THESE présentée en vue de l'obtention de grade de
Doctorat de l'Université de Toulouse
délivrée par l'INSA de Toulouse

Spécialité : **Système Automatique**

PAR M. : **Vincent BOYER**

INTITULE :

Contribution à la programmation en nombres entiers

SOUTENUE LE : **14/12/2007**

JURY :

Saïd HANAFI	LAMIH (Valenciennes)	Président du jury
Mhand HIFI	LARIA (Amiens)	Rapporteur/Examinateur
Gérard PLATEAU	LIPN (Paris Nord)	Rapporteur/Examinateur
Jean-Bernard LASSERRE	LAAS-CNRS (Toulouse)	Directeur de thèse
Didier EL BAZ	LAAS-CNRS (Toulouse)	Co-directeur de thèse
Moussa ELKIHEL	LAAS-CNRS (Toulouse)	Co-directeur de thèse

ECOLE DOCTORALE : **Ecole Doctorale Système Automatique**

LABORATOIRE : **Laboratoire d'analyse et d'architecture des systèmes**

Résumé

Le problème du sac à dos à plusieurs contraintes est un problème classique de l'optimisation appartenant à la classe des problèmes NP-difficiles. On le retrouve notamment sous la forme de sous-problème de nombreux problèmes d'optimisation combinatoire. Les méthodes classiques de résolution exacte telles que la programmation dynamique ou le branch-and-bound ont été traitées abondamment dans la littérature. Elles présentent néanmoins des faiblesses si elles sont utilisées telles quelles, d'où l'idée de faire coopérer ces méthodes en tirant profit de leurs spécificités afin de proposer soit des méthodes heuristiques performantes, soit des méthodes exactes plus efficaces.

Les approches heuristiques que nous proposons sont comparées à d'autres heuristiques de la littérature. Notre méthode coopérative est, quant à elle, comparée à un algorithme de branch-and-bound. L'ensemble de ces tests numériques ont été menés pour diverses instances plus ou moins difficiles de la littérature ainsi que sur des instances engendrées aléatoirement.

Enfin, nous proposons deux techniques pour engendrer des problèmes difficiles. Ces dernières sont basées sur des problèmes en contraintes égalités et sur l'analyse de la transformée en $\mathbb{Z}$ du sac à dos.

Mots clés : optimisation combinatoire, problème du sac à dos multidimensionnel, méthodes coopératives, heuristiques.

Abstract

The Multi-Knapsack Problem is a traditional problem of optimization belonging to the class of the NP-hard problems. This problem occurs in particular as a subproblem of many combinatorial optimization problems. Traditional methods, such as dynamic programming or branch-and-bound, used for the exact solution have been treated abundantly in the literature. They have nevertheless weaknesses if they are used alone. The cooperative methods permit one to benefits from their specificities and to propose powerful heuristics or efficient exact methods.

Heuristics approaches are proposed and compared with other heuristics from the literature. A cooperative method is compared with a branch-and-bound algorithm. Numerical tests were carried out on various difficult problems from the literature and on randomly generated problems.

At last, we consider in particular two techniques in order to generate difficult problems. They are based on problems with equality constraints and the analyzis of the Z transform of the knapsack problem.

Key words : combinatorial optimization, multi-knapsack problem, cooperative methods, heuristics.

Table des matières

Introduction **11**

I Le problème du sac-à-dos **15**

I.1 Introduction 16

I.2 Historique 17

I.3 Les instances de la littérature 18

I.4 Les instances engendrées aléatoirement 19

I.4.1 Problèmes à données non-corrélées 19

I.4.2 Problèmes à données corrélées 19

I.4.3 Problèmes dérivés d'un problème combinatoire en contrainte égalité . . . 20

I.4.4 Instances engendrées à partir de l'analyse de la transformée en $\mathbb{Z}$ 21

I.5 Conclusion 24

II Etat de l'art **25**

II.1 Introduction 26

II.2 Les méthodes de relaxations 26

II.2.1 La relaxation continue 26

II.2.2 La relaxation lagrangienne 27

II.2.3 La relaxation surrogate ou agrégée 27

II.2.4 La relaxation surrogate continue 28

II.2.5 La relaxation Composite 28

II.2.6 Choix des multiplicateurs 29

II.3 Les méthodes de réduction 29

II.3.1 Les fixations de variables 30

II.3.2 Les réductions de contraintes 32

II.3.3 L'algorithme RAMBO 32
II.3.4 La rotation de contraintes 35
II.4 Les méthodes heuristiques 37
II.4.1 AGNES 37
II.4.2 L'heuristique ADP-BH 38
II.4.3 SMA : un algorithme simple multiniveaux 41
II.4.4 Les autres heuristiques existantes 44
II.5 Les méthodes de résolution exacte 44
II.5.1 Branch and Bound 44
II.5.2 La Programmation Dynamique 49
II.5.3 Les solveurs existants 56
II.6 Les méthodes coopératives 57
II.6.1 Méthode coopérative pour le problème (SSP) 58
II.6.2 Méthode coopérative à trois phases pour le problème (UKP) 59
II.7 Conclusion 60

III Une méthode coopérative multi-étape 63

III.1 Introduction 64
III.2 La relaxation surrogate 65
III.2.1 Les heuristiques étudiées 65
III.2.2 Résultats numériques 70
III.3 L'heuristique HDP 74
III.3.1 Un algorithme de programmation dynamique amélioré pour les problèmes uni-contraints 75
III.3.2 Résolution du problème surrogate par la programmation dynamique hybride 75
III.3.3 Comparaison des heuristiques 80
III.3.4 HDP augmenté 81
III.4 Une méthode coopérative de résolution exacte 83
III.4.1 Calcul des bornes 84
III.4.2 La stratégie de branchement 85
III.4.3 La stratégie de parcours 85
III.4.4 Algorithme 85

III.5 La phase de branch and bound limité 87
III.5.1 *BB_lim_N* : limitation du nombre de noeuds dans $\mathcal{L}_{sec}$ 88
III.5.2 *BB_lim_T* : limitation du temps de calcul 89
III.5.3 Comparaison des deux approches 90
III.6 Conclusion 90

IV Les résultats numériques **93**
IV.1 Introduction 94
IV.2 Présentation des instances considérées 94
IV.3 Tests numériques sur les grandes instances de la littérature 94
IV.3.1 *HDP* et *HDP+HDP_augmenté* 95
IV.3.2 *HDP+HDP_augmenté+BB_lim_T* 96
IV.3.3 Les méthodes coopératives 97
IV.4 Tests numériques sur les instances aléatoires 97
IV.4.1 *HDP+HDP_augmenté* 98
IV.4.2 Les méthodes coopératives 98
IV.5 Conception de nouvelles instances difficiles 99
IV.5.1 Instances engendrées à partir d'un problème en contrainte égalité 100
IV.5.2 Instances engendrées à partir de l'analyse de la transformée en $\mathbb{Z}$ 102
IV.6 Conclusion 102

Conclusions et Prospectives **105**

Références bibliographiques **107**

Table des figures

II.1 Arbre généré par décomposition du sous-problème initial 46
II.2 Représentation des listes de la programmation dynamique. 52
II.3 Application du principe de dominance. 53

III.1 Représentation graphique des différentes contraintes surrogate pour (Ex_1) . . 69
III.2 Représentation graphique des différentes contraintes surrogate pour (Ex_2) . . 70

Liste des tableaux

II.1 Génération des listes sans décomposition du problème. 55

II.2 Génération des listes par décomposition du problème. 55

III.1 Relaxation surrogate : Ecart moyen à l'optimum. 72

III.2 Relaxation surrogate : Nombre moyen de contraintes non-respectées. 72

III.3 Relaxation surrogate : Temps de calcul du multiplicateur. 72

III.4 Relaxation surrogate : Amélioration par l'algorithme G. 74

III.5 Relaxation surrogate : Amélioration par rotation de contraintes. 74

III.6 Comparaison de deux algorithmes gloutons. 79

III.7 Ecart à l'optimalité des algorithmes HDP. 81

III.8 Optimalité avec les algorithmes HDP. 81

III.9 Temps de calcul des algorithmes HDP. 81

III.10 Ecart à l'optimalité des algorithmes *HDP+HDP_augmenté*. 82

III.11 Optimalité avec les algorithmes *HDP+HDP_augmenté*. 83

III.12 Temps de calcul des algorithmes *HDP+HDP_augmenté*. 83

III.13 Temps de calcul des algorithmes *HDP+HDP_augmenté+B&B*. 86

III.14 Pourcentage du temps passé dans *B&B* avec *HDP+HDP_augmenté+B&B*. . . . 87

III.15 Ecart à l'optimalité des algorithmes *HDP+HDP_augmenté+BB_lim_N*. 88

III.16 Optimalité avec les algorithmes *HDP+HDP_augmenté+BB_lim_N*. 88

III.17 Temps de calcul des algorithmes *HDP+HDP_augmenté+BB_lim_N*. 89

III.18 Ecart à l'optimalité des algorithmes *HDP+HDP_augmenté+BB_lim_T*. 89

III.19 Optimalité avec les algorithmes *HDP+HDP_augmenté+BB_lim_T*. 90

III.20 Temps de calcul des algorithmes *HDP+HDP_augmenté+BB_lim_T*. 90

IV.1 Paramètres des instances aléatoires. 94

IV.2 HDP : les grandes instances de la littérature (Écart relatif (%)). 95
IV.3 HDP : les grandes instances de la littérature (Temps de calcul (s)). 96
IV.4 BB_lim_T : les grandes instances de la littérature. 96
IV.5 Méthode coopérative : les grandes instances de la littérature. 97
IV.6 HDP : Instances non-corrélées (Ecart relatif (%)). 98
IV.7 HDP : Instances non-corrélées (Temps de calcul (s.)). 99
IV.8 HDP : Instances corrélées (Ecart relatif (%)). 100
IV.9 HDP : Instances corrélées (Temps de calcul (s)). 100
IV.10 Méthode coopérative : Instances non-corrélées. 101
IV.11 Méthode coopérative : Instances corrélées. 101
IV.12 Temps de calcul (en s.) pour $I = [1, 10]$. 102
IV.13 Temps de calcul (en s.) pour $I = [1, 1000]$. 102
IV.14 I_Z : Temps de calcul (en s.). 102

Introduction

Le problème du sac à dos multidimensionnel, ou MKP pour Multi Knapsack Problem, est un problème classique d'optimisation combinatoire appartenant à la classe des problèmes NP-complets. Il fait l'objet d'un grand intérêt de la part de la communauté scientifique, souligné par la création d'un groupe de travail dédié : KSO (KnapSack et Optimisation) du GDR Recherche Opérationnelle, RO. Les applications dans ce domaine sont en effet très nombreuses et concernent des problèmes de logistique, de productique, d'économie ou de finances. En particulier, les problèmes de sac à dos multidimensionnels interviennent dans les domaines suivants :
– Gestion des chaînes logistiques (problème de répartition)
– Ordonnancement de production
– Gestion des ressources humaines (emploi du temps)
– Maintenance (service après vente, ordre des visites, minimisation de parcours)
– Gestion des sièges de trains, avions en fonction des tarifs
– Gestion de ressources (départ, arrivée, placement des avions)
– Gestion des containers dans les ports
Par ailleurs, le problème MKP intervient comme un sous-problème de nombreux problèmes d'optimisation combinatoire.

Les travaux présentés dans ce mémoire s'inscrivent dans la continuité des travaux de Elkihel, Viader et Authié ([VIA 98], [ELK 02]) ainsi que ceux de Lasserre ([LAS 03a], [LAS 03b] et [LAS 04c]). Nous nous sommes intéressés aux méthodes de résolution approchées et exactes du problème MKP.

Les méthodes les plus utilisées pour résoudre des programmes linéaires en variables 0-1 sont la programmation dynamique, les méthodes de branch-and-bound et les méthodes algébriques. L'application directe des méthodes classiques de résolution au problème multi knapsack peut engendrer des temps de résolution importants. Elkihel, Viader et Authié ([VIA 98], [ELK 02]), dans la lignée de Plateau et Elkihel [PLA 85], ont proposé une méthode coopérative pour le problème du sac à dos à une contrainte. Les tests numériques présentés montrent son efficacité par rapport aux méthodes classiques telles que la programmation dynamique et le branch-and-bound. Un des objectifs de nos travaux est de contribuer à l'extension des méthodes coopératives au cas des problèmes à plusieurs contraintes.

Parallèlement à cette étude sur les méthodes coopératives est née une réflexion sur ce qui fait qu'un problème multi knapsack est difficile, et sur diverses techniques pour engendrer des problèmes ardus. En effet, il existe peu de travaux sur la nature de la difficulté en optimisation combinatoire. Notre approche est basée sur les travaux de Lasserre ([LAS 03a], [LAS 03b] et [LAS 04c]), ainsi que ceux de El Baz, Elkihel et Gely [ELB 05a]. Une partie des travaux de Lasserre porte sur la transformée en $\mathbb{Z}$ du problème du sac à dos en contrainte égalité et à variables non-bornées. El Baz, Elkihel et Gely [ELB 05a] ont proposé une méthode de résolution de problème difficile en contrainte en égalité à partir de formulations en contraintes inégalités.

Nous proposons deux techniques pour construire des instances difficiles.

Dans le premier chapitre, nous commençons par définir le problème MKP ainsi que les notations et le vocabulaire utilisés. Dans la littérature de nombreux auteurs ont proposé des instances, qui sont en libre accès sur internet, ou des méthodes pour engendrer des instances de manière aléatoire. Nous présentons ces différentes instances ainsi que celles engendrées par les deux nouvelles techniques que nous proposons, à savoir :
– instances dérivées de problèmes combinatoires en contrainte égalité et
– instances engendrées à partir de l'analyse de la transformée en $\mathbb{Z}$.

Le chapitre 2 est dédié à l'état de l'art. Tout d'abord, nous présentons les méthodes de relaxation, avec en particulier la relaxation surrogate introduite par Glover [GLO 65] que nous utiliserons par la suite. Ensuite, nous aborderons les méthodes de réduction. Elles contribuent à réduire la taille du problème en fixant des variables ou en éliminant des contraintes à partir de différents tests. On citera notamment l'algorithme RAMBO de Fréville et Plateau [FRE 94]. Enfin, nous considérons diverses méthodes de résolution proposées dans la littérature :
– les méthodes approchées, qui permettent d'obtenir une solution approchée de la fonction objectif qui satisfait l'ensemble des contraintes. On s'intéresse en particulier aux heuristiques AGNES de Fréville et Plateau [FRE 94], ADP-based heuristics approach de Bertsimas et Demir [BER 02] et SMA de Hanafi, Fréville et El Abdellaoui [HAN 96] ;
– les méthodes classiques de résolution exacte, qui sont la programmation dynamique et le branch-and-bound ;
– les méthodes coopératives de résolution exacte dans le cas de problème à une contrainte, avec les méthodes proposées par Martello et Toth [MAR 82], Plateau et Elkihel [PLA 85], ainsi que Authié, Elkihel et Viader ([VIA 98] et[ELK 02]).
Ces méthodes coopératives ont montré leur efficacité pour des problèmes à une seule contrainte, mais aucune méthode efficace n'a été proposée pour le cas à plusieurs contraintes.

Au chapitre 3, nous présentons notre contribution aux méthodes de résolution du problème MKP. L'ensemble de ces méthodes est construit sur une première heuristique, nommée HDP (Hybrid Dynamic Programming). HDP est basée sur la résolution du problème MKP en appliquant un algorithme de programmation dynamique sur sa relaxation surrogate. Cet algorithme est modifié par rapport à la méthode classique afin de garantir la réalisabilité de la solution obtenue pour le problème MKP. Lors de ce processus, des états éliminés par la programmation dynamique sont conservés et peuvent être réutilisés pour construire :
– des heuristiques augmentées qui permettent d'améliorer la solution obtenue avec HDP,
– une méthode coopérative de résolution exacte.
Nous proposons deux heuristiques augmentées, qui correspondent à différentes recherches au voisinage des états conservés. La première se base sur un algorithme glouton et la deuxième sur un algorithme de branch-and-bound. De plus, nous présentons une méthode coopérative de résolution exacte combinant la programmation dynamique de HDP et le branch-and-bound. Nous avons donc été amenés à faire des choix concernant, tout d'abord, le calcul des multiplicateurs de la relaxation surrogate, et, aussi, les algorithmes envisagés (calcul des bornes, stratégie de parcours, classement de variables,...). Différents tests numériques préliminaires sur des instances de la littérature de petite taille viennent illustrer les implications de ces différents choix.

Enfin, le chapitre 4 est consacré aux tests numériques. Ces derniers ont été menés sur des instances de taille plus importante issues de la littérature ou engendrées de manière aléatoire. Nos heuristiques ont été comparées avec les heuristiques AGNES, ADP-based heuristics approach et SMA. Nous avons comparé les résultats obtenus avec notre méthode coopérative à ceux d'une méthode classique, le branch-and-bound. L'ensemble de ces tests nous permet-

tra d'évaluer les performances des différentes méthodes proposées. Nous présentons aussi les résultats préliminaires obtenus avec nos deux techniques pour engendrer des instances aléatoires difficiles. Pour cela, nous les avons résolues de manière exacte avec CPLEX 9.0 et avons comparé les temps de calcul avec les instances obtenues avec les techniques classiques.

Chapitre I

Le problème du sac-à-dos

I.1 Introduction

Dans ce chapitre, nous définirons le problème MKP ainsi que les notations et le vocabulaire qui seront utilisés dans la suite. De plus, nous verrons les différentes instances que l'on peut trouver dans la littérature, ainsi que des méthodes pour engendrer des instances de manière aléatoire, à savoir les instances à données non-corrélées, faiblement corrélées et fortement corrélées. Nous présentons ces différentes instances ainsi que celles engendrées par deux nouvelles techniques :
– des instances dérivées de problèmes combinatoires en contrainte égalité et
– des instances engendrées à partir de l'analyse de la transformée en $\mathbb{Z}$.

Le problème du sac à dos est un problème classique de l'optimisation combinatoire appartenant à la classe des problèmes NP-difficiles [FRE 04]. Un problème d'optimisation combinatoire est caractérisé par :
– un ensemble fini de solutions Ω,
– une fonction objectif $f : \Omega \rightarrow \mathbb{R}$ associant une valeur à chaque solution admissible. Ce problème consiste à déterminer la solution $x^* \in \Omega$ maximisant f.

$$x^* = argmax\{f(x) \mid x \in \Omega\}. \tag{I.1}$$

Le problème du sac à dos multidimensionnel est formulé comme suit :

$$(MKP) \begin{cases} \max \sum_{j=1}^{n} p_j.x_j, \\ \text{s.c. } \sum_{j=1}^{n} w_{i,j}.x_j \leq c_i, \ i \in \{1,...,m\}, \\ x_j \in \{0,1\}, \ j \in \{1,...,n\}, \end{cases} \tag{I.2}$$

avec p_j, c_i, et $w_{i,j}$ entiers positifs pour $i \in \{1,...,m\}$ et $j \in \{1,...,n\}$.

Ou plus simplement :

$$(MKP) \begin{cases} \max p^t.x, \\ \text{s.c. } W.x \leq c, \\ x \in \{0, \ 1\}^n, \end{cases} \tag{I.3}$$

avec $p = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix}$, $c = \begin{pmatrix} c_1 \\ \vdots \\ c_m \end{pmatrix}$ et $W = \begin{pmatrix} w_{1,1} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{m,1} & \cdots & w_{m,n} \end{pmatrix}$.

De plus, nous nous placerons sous les hypothèses (h1) et (h2) qui écartent des cas triviaux :

– (h1) : $\max_{j \in \{1,...,n\}} \{w_{i,j}\} \leq c_i$ pour $i \in \{1,...,m\}$ (exclusion d'une variable) ,

– (h2) : $\forall i \in \{1,...,m\}$, $c_i \geq \sum_{j=1}^{n} w_{i,j}$ (contrainte toujours réalisée).

Nous adopterons les notations et les appellations suivantes :

– p_j est appelé le **profit** associé à la variable x_j,

– W_i est la **i$^{\text{ème}}$ ligne de la matrice W**,
– $\sum_{j=1}^{n} w_{i,j}.x_j \leq c_i$ est la **i$^{\text{ème}}$ contrainte de poids**,
– $c = (c_1, ..., c_m)^t$ est la **capacité** du sac à dos,
– on note $W^j = \begin{pmatrix} w_{1,j} \\ \vdots \\ w_{n,j} \end{pmatrix}$ la **j$^{\text{ème}}$ colonne de la matrice W** soit le vecteur poids associé à la variable x_j,
– la taille d'un problème sera donné sous la forme $\mathbf{n} \times \mathbf{m}$,
– $v(S)$ représente la **valeur optimale du critère** associée au problème (S) et
– $\overline{v}(S)$ et $\underline{v}(S)$ sont, respectivement, un majorant et un minorant de $v(S)$.

On rencontre le problème du sac à dos multidimensionnel lorsqu'on cherche à optimiser l'usage d'une entité qui consomme des ressources multiples. Il correspond au modèle général de type allocation de ressources où les données ont les significations suivantes :

n : nombre d'articles,

m : nombre de ressources,

p_j : le profit lié au choix de l'article j,

c_i : la quantité de ressource i disponible,

$w_{i,j}$: la quantité de ressource i consommée par l'article j,

x_j : la variable de décision qui vaut 1 si le projet j est accepté, 0 sinon.

L'objectif est de maximiser le profit dans la limite des ressources disponibles.

La première utilisation de ce modèle concerne les problèmes d'investissement :
– p_j représente le bénéfice du projet j,
– $w_{i,j}$ la demande en capital du projet j dans la période i,
– c_i le capital disponible dans la période i et
– m le nombre de périodes considérées.
Il a été utilisé aussi dans :
– des problèmes de découpes.
– des problèmes de chargement.
Ce problème apparaît encore comme un sous-problème dans les modèles destinés à attribuer les processeurs et les bases de données dans un système informatique distribué.

I.2 Historique

La communauté scientifique s'est intéressée au problème du sac à dos multidimensionnel pour la première fois dans les années 50. Depuis, un grand nombre d'algorithmes et de techniques a été présenté pour trouver des solutions exactes et approchées : les méthodes de relaxations, le branch-and-bound, la programmation dynamique, les méthodes heuristiques, les algorithmes parallèles, etc.

Historique :

Années 50 : Premier algorithme de Programmation Dynamique (R. BELLMAN [BEL 57]) ;
Borne supérieure de la valeur optimale du critère (G.B. DANTZIG [DAN 57]) ;

Années 50 : Amélioration de la Programmation Dynamique (P.C. GILMORE et R.E. GOMORY [GIL 61] [GIL 62] [GIL 65]) ;
Premier Branch-and-Bound (P.J. KOLESAR [KOL 67]) ;

Années 70 : Algorithme de B&B revu (E. HOROWITZ, S. SAHNI [HOR 74]) ;
Première procédure de réduction du nombre de variables (G.P. INGARGIOLA, J.F. KORSH [ING 73]) ;
Première approximation en temps polynomial (S. SAHNI [SAH 75]) ;
Approximation en temps polynomial (O.H. IBARRA et C.E. KIM [IBA 75]) ;
Borne supérieure dominant celle obtenue par la relaxation continue (MARTELLO, P. TOTH [MAR 77]) ;

Années 80 : Résultats sur la solution des problèmes de très grande taille où le classement de variables prend la plupart du temps ;
Travaux portant sur le noyau du problème (E. Balas et E. Zemel, [BAL 80], G. Plateau et M. Elkihel [PLA 85]) ;

Années 90 : Algorithmes parallèles (W. LOOTS, SMITH T.H.C. [LOO 92]) ;
Approche par réseau de neurones (Ohlsson M., Peterson C. et Soderberg B. [OHL 93]) ;
Approche par des techniques d'Intelligence Artificielle (Ko I. [KO 93]) ;
Résolution du problème par échantillonnage (PENN M. et HASSON D. [PEN 94]).

I.3 Les instances de la littérature

Le problème du sac à dos étant étudié depuis maintenant plus de 50 ans, un certain nombre d'auteurs a proposé des problèmes types dans leurs publications qui se retrouvent maintenant largement diffusées grâce à internet. Les problèmes présentés ici et qui nous ont permis de mener une grande partie de nos tests numériques sont disponibles via la OR-library tenue par Beasley :

$$http://people.brunel.ac.uk/\sim mastjjb/jeb/orlib/mknapinfo.html$$

Les instances que l'on y trouve sont les suivantes :
– instance de Petersen [PET 67] : 7 problèmes de tailles différentes (de 6×10 à 50×5),
– instance de Weingartner et Ness [WEI 67] : 8 problèmes (de 28×2 à 105×2),
– instance de Senyu et Toyoda [SEN 68] : 2 problèmes (60×30),
– instance de Shi [SHI 79] : 30 problèmes dont 5 problèmes de tailles 30×5, 4 de 40×5, 4 de 50×5, 4 de 60×5, 4 de 70×5, 4 de 80×5 et 5 de 90×5.
– instance de Fréville et Plateau [FRE 90] : 8 problèmes de tailles différentes (de 27×4 à 40×30) et
– instances de Chu et Beasley [CHU 98] : 9 instances de 30 problèmes de taille 100×5 à 500×30

Il est à noter que pour les instances de Chu et Beasley, contrairemant aux autres instances, les bornes optimales des problèmes ne sont pas connues du fait de leurs tailles importantes, c'est pourquoi nous ferons souvent la distinction suivante :
– les petites instances de la littérature, à savoir ceux de Petersen, Weingartner et Ness, Senyu et Toyoda, Shi et Fréville et Plateau et
– les grandes instances de la littérature, à savoir ceux de Chu et Beasley.

I.4 Les instances engendrées aléatoirement

La création d'instances de façon aléatoire pose le problème de la manière d'engendrer des instances difficiles. Cette question est d'autant plus intéressante sachant que peu d'études ont été faites là-dessus.

Bien entendu, la difficulté de résolution d'un problème dépend de la taille de ce dernier ou encore de la taille des coefficients, mais pas uniquement. On se rend compte qu'en menant des tests sur des problèmes aléatoires de tailles identiques, certains sont beaucoup plus durs à résoudre que des problèmes de tailles bien plus grandes (voir par exemple [BEI 03]).

Dans cette section nous présenterons quelques pistes de recherche sur le moyen d'engendrer des instances difficiles.

I.4.1 Problèmes à données non-corrélées

Lorsque l'on étudie la complexité des méthodes de résolution exacte, on se rend compte qu'elle est liée d'une part à sa taille, mais aussi à la capacité du problème. Nous pouvons considérer les problèmes suivants :

$$(MKP(\epsilon)) \begin{cases} \max \sum_{j=1}^{n} p_j.x_j, \\ \text{s.c.} \sum_{j=1}^{n} w_{i,j}.x_j \leq \epsilon.\sum_{j=1}^{n} w_{i,j}, \ i \in \{1,...,m\}, \\ x_j \in \{0,1\}, \ j \in \{1,...,n\}, \end{cases} \tag{I.4}$$

avec $\epsilon \in]0,1[$ et pour $j \in \{1,...,n\}$ et $i \in \{1,...,m\}$, les coefficients p_j, c_i et $w_{i,j}$ sont tirés de manière uniforme dans, respectivement, $[1,U_p]$ et $[1,U_w]$ où ϵ, U_p et U_w sont des paramètres fixés par l'utilisateur (U_p et U_w sont généralement pris égaux à 1000).

Ceci constitue la méthode la plus classique pour engendrer des problèmes de façon aléatoire ; elle est d'ailleurs à la base de la génération des instances de Chu et Beasley [CHU 98] pour lesquelles ϵ prend les valeurs 0,25, 0,50 et 0,75.

Les instances les plus difficiles sont crées pour $\epsilon = 0,5$. En effet, pour un ϵ petit, le sac à dos ayant une faible capacité se remplit vite, et pour un ϵ grand, un changement de variable de type $x \rightarrow 1-x$ permet de se ramener au cas précédent.

I.4.2 Problèmes à données corrélées

Contrairement au cas précédent, la génération du vecteur des profits p est liée à la matrice des poids W. On distingue les problèmes :
– faiblement corrélés : pour $j \in \{1,...,n\}$ et $i \in \{1,...,m\}$, les coefficients $w_{i,j}$ sont tirés de

manière uniforme dans $[1, U_w]$ et $p_j = \max\{1,\ \hat{p}_j\}$ avec $\hat{p}_j$ tiré de manière uniforme dans :

$$\left[\frac{\sum_{k=1}^{m} w_{k,j}}{m} - wc, \frac{\sum_{k=1}^{m} w_{k,j}}{m} + wc \right],$$

où wc est un paramètre spécifié par l'utilisateur ;

– fortement corrélés : pour $j \in \{1, ..., n\}$ et $i \in \{1, ..., m\}$, les coefficients $w_{i,j}$ sont tirés de manière uniforme dans $[1, U_w]$ et

$$p_j = \frac{\sum_{k=1}^{m} w_{k,j}}{m} + sc,$$

où sc est un paramètre spécifié par l'utilisateur.

Quant au vecteur des capacités c, il est généralement engendré de la même façon qu'avec les problèmes non-corrélés, à savoir $c = \epsilon.\sum_{j=1}^{n} W^j$, $\epsilon \in]0, 1[$.

Les problèmes fortement corrélés sont généralement plus difficiles à résoudre que leurs homologues non-corrélés (cf. [PAN 93], [MAR 97], [PIS 95]).

I.4.3 Problèmes dérivés d'un problème combinatoire en contrainte égalité

Cette méthode se base sur les travaux de El Baz, Elkihel et Gely [ELB 05b] qui ont présenté une méthode de résolution du problème du sac à dos multidimensionnel en contrainte égalité basée sur la résolution d'une série de problèmes en contrainte inégalité. Cette approche est une généralisation de celle de Plateau et Elkihel [PLA 85] au cas multidimensionnel.

On considère les problèmes équivalents :

$$(E) \begin{cases} \max p^t.x, \\ \text{s.c. } W.x = c, \\ x \in \{0,\ 1\}^n, \end{cases} \quad \Leftrightarrow \quad (E) \begin{cases} \max p^t.x, \\ \text{s.c. } \begin{cases} W.x \leq c, \\ W.x \geq c, \end{cases} \\ x \in \{0,\ 1\}^n. \end{cases} \tag{I.5}$$

Une relaxation Lagrangienne sur la contrainte $W.x \geq c$ donne alors :

$$(\widetilde{E}(\Lambda)) \begin{cases} \max p^t.x + \Lambda^t.(W.x - c), \\ \text{s.c. } W.x \leq c, \\ x \in \{0,\ 1\}^n, \end{cases} \tag{I.6}$$

où $\Lambda \geq 0$.

Λ est pris tel que $\Lambda = \lambda.\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$ avec $\lambda \in \mathbb{R}^+$. On peut alors reformuler $\widetilde{E}(\Lambda)$ de la façon suivante :

$$(\widetilde{E}(\lambda))\begin{cases}\max p^t.x+\lambda.\begin{pmatrix}1 & \dots & 1\end{pmatrix}.(W.x-c),\\ \text{s.c. } W.x\leq c,\\ x\in\{0,\ 1\}^n,\end{cases} \tag{I.7}$$

Pour un λ^* suffisamment grand on a $v(\widetilde{E}(\lambda^*))=v(E)$ [ELB 05b], autrement dit il y a équivalence entre E et $\widetilde{E}(\lambda^*)$. Ceci nous mène donc à engendrer des problèmes de la forme de $\widetilde{E}(\lambda)$ sachant que plus le paramètre λ sera grand, plus le problème relâché est proche du problème en contrainte égalité. En prenant $c=\epsilon.\sum_{j=1}^{n}W^j$, $\epsilon\in]0,1[$, on obtient d'une façon générale des problèmes de la forme :

$$(MKP(\epsilon,\lambda))\begin{cases}\max p^t.x+\lambda.\begin{pmatrix}1 & \dots & 1\end{pmatrix}.(W.x-\epsilon.\sum_{j=1}^{n}W^j),\\ \text{s.c. } W.x\leq\epsilon.\sum_{j=1}^{n}W^j,\\ x\in\{0,\ 1\}^n,\end{cases} \tag{I.8}$$

avec $\epsilon\in]0,1[$, $\lambda\geq 0$ et pour $j\in\{1,...,n\}$ et $i\in\{1,...,m\}$, les coefficients p_j, c_i et $w_{i,j}$ sont tirés de manière uniforme dans, respectivement, $[1,U_p]$ et $[1,U_w]$. ϵ, U_p et U_w sont des paramètres fixés par l'utilisateur.

Plus le paramètre λ est grand, plus le problème en inégalité est ardu car la partie purement liée au critère est écrasée par la satisfaction des contraintes. Ceci nous amène à étudier l'influence de ce paramètre λ sur la difficulté de résolution de problèmes engendrés ainsi.

I.4.4 Instances engendrées à partir de l'analyse de la transformée en $\mathbb{Z}$

On considère le problème général en nombre entier suivant :

$$\mathbb{P}_d : f_d(c,p)=\max\{p^t.x\mid Wx=c, x\in\mathbb{N}^n\}, \tag{I.9}$$

ainsi que le problème associé suivant (voir Lasserre [LAS 03b]) :

$$\begin{aligned}\mathbb{I}_d : \hat{f}_d(c,p)&=\sum_{x\in\Omega(c)}e^{p^t.x},\\ \Omega(c)&=\{x\in\mathbb{N}^n\mid W.x=c\}.\end{aligned} \tag{I.10}$$

Une partie des travaux de Lasserre ([LAS 03a], [LAS 03b] et [LAS 04c]) suggère de chercher un dual de $\mathbb{P}_d$ à partir de la transformée en $\mathbb{Z}$ de $\mathbb{I}_d$. Sachant que $\mathbb{P}_d$ et $\mathbb{I}_d$ sont liés par la relation suivante :

$$f_d(c,p)=\lim_{r\to\infty}\frac{1}{r}.ln\left(\hat{f}_d(c,r.p)\right). \tag{I.11}$$

La transformée en $\mathbb{Z}$ de $\mathbb{I}_d$, par sa transformée inverse, mène à une nouvelle formulation équivalente du problème $\mathbb{I}_d$. A partir de ce dual de $\mathbb{I}_d$, l'utilisation de l'équation (I.11) permet de construire un dual pour le problème $\mathbb{P}_d$. La transformée en $\mathbb{Z}$ de $\mathbb{I}_d$ nous donne donc (voir Lasserre [LAS 03a]) :

$$\hat{F}_d(z,p) = \sum_{y\in\mathbb{Z}^m} z^{-y}.\hat{f}_d(y,p) = \prod_{k=1}^{n} \frac{1}{(1-e^{p_k}.z^{-W^k})}, \\ avec\ pour\ k\in\{1,...,n\},\ z^{-W^k} = z_1^{-w_{1,k}}.z_2^{-w_{2,k}}...z_m^{-w_{m,k}}. \tag{I.12}$$

On constate que $\hat{F}_d(z,p)$ est bien définie si $pour\ k\in\{1,...,n\},\ |z^{-W^k}| > e^{p_k}$. L'intégration de cette fonction (transformée en $\mathbb{Z}$ inverse) sera d'autant plus difficile qu'elle a un grand nombre de pôles. L'idée est donc d'engendrer un problème $\mathbb{P}_d$ conduisant à un grand nombre de pôles.

On considère $\sigma = \{j_1,...,j_m\} \subset \{1,...,m\}$ tel que la matrice $W_\sigma = (W^{j_1},...,W^{j_m})$ soit une base du problème linéaire associé. Pour un tel σ, nous avons le système à m équations suivant :

$$Pour\ j\in\sigma,\ z_1^{-w_{1,j}}.z_2^{-w_{2,j}}...z_m^{-w_{m,j}} = e^{p_j}. \tag{I.13}$$

Ce système a $\rho(\sigma) = det(W_\sigma)$ solutions de la forme (voir Lasserre [LAS 03a]) :

$$z(k) = e^{\lambda}.e^{2\imath\pi\theta(k)},\ k\in\{1,...,\rho(\sigma)\} \\ avec\ \theta(k)\in\mathbb{R}^m\ et\ \lambda = W_\sigma^{-1}.p_\sigma. \tag{I.14}$$

Donc, pour avoir un grand nombre de pôles, on doit engendrer des matrices de base W_σ ayant un déterminant suffisamment grand. Plus les rayons, $\lambda = W_\sigma^{-1}.p_\sigma$, des multi-disques de $\mathbb{C}^n$ sur lesquels sont ces pôles, sont proches les uns des autres plus l'intégration est difficile.

Dans le cas des problèmes uni-contraints, le calcul de l'ensemble des λ, pour toutes les bases admissibles, nous donne :

$$\lambda_k = \frac{p_k}{w_k}\ avec\ k\in\left\{ j\in\{1,...,n\} \mid w_j\neq 0\ et\ \frac{p_j}{w_j}\geq 0\right\}. \tag{I.15}$$

Dans le cas à une contrainte, plus les rapports profits sur poids sont proches les uns des autres, plus le problème à une contrainte est difficile à résoudre. Cette conclusion peut-être généralisée au cas multidimensionnel grâce à l'utilisation de la transformée en $\mathbb{Z}$ en prenant les $W_\sigma^{-1}.p_\sigma$ proches :

$$\left\| W_\sigma^{-1}.p_\sigma - W_{\sigma'}^{-1}.p_{\sigma'}\right\| \leq \epsilon,\ \epsilon\ \in\mathbb{R}_+. \tag{I.16}$$

A partir du problème en égalité ainsi construit, on remplace simplement les égalités par des inégalités pour obtenir le problème en contraintes inégalité désiré.

Application au cas à deux contraintes

Pour étudier le temps de résolution d'un problème engendré de cette façon, nous avons appliqué, dans un premier temps, cette méthode au cas des problèmes à deux contraintes. Ceci nous permettra de valider cette approche afin d'envisager son extension au cas multidimensionnel.

Afin de simplifier les calculs, ces problèmes seront pris sous la forme :

$$(MKP)\begin{cases} max\ p_0.x_1 + (p_0+\delta_p).x_2 + ... + (p_0+(n-1).\delta_p).x_n, \\ s.c.\begin{cases} a_0.x_1 + (a_0+\delta_a).x_2 + ... + (a_0+(n-1).\delta_a).x_n \leq a, \\ b_0.x_1 + (b_0+\delta_b).x_2 + ... + (b_0+(n-1).\delta_b).x_n \leq b, \\ x_j\in\{0,1\},\ j\in\{1,...,n\}, \end{cases} \end{cases} \tag{I.17}$$

avec $n >= 3$.

Avec $\{j1, j2\} \subset \{0, ..., n-1\}$, considérons la matrice carré :

$$W_{j1,j2} = \begin{pmatrix} a_0 + j1.\delta_a & a_0 + j2.\delta_a \\ b_0 + j1.\delta_b & b_0 + j2.\delta_b \end{pmatrix} \tag{I.18}$$

$W_{j1,j2}$ est une base de $\mathbb{R}^2$ si :

$$|det(W_{j1,j2})| = |j1 - j2|.|\delta_a.b_0 - \delta_b.a_0| \neq 0, \tag{I.19}$$

or $|det(W_{j1,j2})| \geq |\delta_a.b_0 - \delta_b.a_0|$, il suffit alors de prendre $|\delta_a.b_0 - \delta_b.a_0| = DET \neq 0$. Toute les matrices $W_{j1,j2}$ sont ainsi des matrices de base de $\mathbb{R}^2$.

Le respect de l'équation (I.16) nous donne la condition suivante :

$$\left\| (W_{j1,j2})^{-1}. \begin{pmatrix} p_o + j1.\delta_p \\ p_o + j2.\delta_p \end{pmatrix} - (W_{j3,j4})^{-1}. \begin{pmatrix} p_o + j3.\delta_p \\ p_o + j4.\delta_p \end{pmatrix} \right\| \leq \epsilon, \tag{I.20}$$

avec $\{j1, j2\} \subset \{0, ..., n-1\}$, $\{j3, j4\} \subset \{0, ..., n-1\}$ et $\{j1, j2\} \neq \{j3, j4\}$.

Afin de simplifier les calculs, cette condition sera imposée pour $j1 = j \in \{0, ..., n-3\}$, $j2 = j+1$, $j3 = j2 = j+1$ et $j4 = j3 + 1 = j + 2$, ce qui donne :

$$\left\| (W_{j,j+1})^{-1}. \begin{pmatrix} p_o + j.\delta_p \\ p_o + (j+1).\delta_p \end{pmatrix} - (W_{j+1,j+2})^{-1}. \begin{pmatrix} p_o + (j+1).\delta_p \\ c_o + (j+2).\delta_p \end{pmatrix} \right\| \leq \epsilon, \tag{I.21}$$

ce qui donne :

$$\alpha.\delta_p^2 + 2.\beta.\delta_p + \gamma \leq (r.DET)^2, \tag{I.22}$$

avec
- $\alpha = 2.(a_0 - b_0 + \delta_a)^2 + (\delta_b - \delta_a)^2(8.j^2 + 12.j + 5) - 2.(\delta_b - \delta_a).(a_0 - b_0 + \delta_a).(4.j + 3)$,
- $\beta = -p_0.(\delta_b - \delta_a).\,(2.(a_0 - b_0 + \delta_a) - (\delta_b - \delta_a).(4.j + 3))$ et
- $\gamma = 2.p_0^2.(\delta_b - \delta_a)^2$.

En se plaçant sous les conditions suivantes :
- $\delta_b - \delta_a \leq 0$ et
- $2.(a_0 - b_0 + \delta_a) - (\delta_b - \delta_a).(4.(n-3) + 3) \leq 0$,

nous pouvons faire les majorations suivantes :
- $\alpha \leq \alpha' = 2.(a_0 - b_0 + \delta_a)^2 + (\delta_b - \delta_a)^2(8.(n-3)^2 + 12.(n-3) + 5)$ et
- $\beta \leq \beta' = -p_0.(\delta_b - \delta_a).\,(2.(a_0 - b_0 + \delta_a) - (\delta_b - \delta_a).(4.(n-3) + 3))$.

Ainsi en prenant $\alpha'.\delta_p^2 + 2.\beta'.\delta_p + \gamma \leq (r.DET)^2$, l'inéquation (I.22) est vérifiée quelque soit $j \in \{0, ..., n-3\}$. De plus la fonction $f : \delta_p \rightarrow \alpha'.\delta_p^2 + 2.\beta'.\delta_p + \gamma$ est minimale pour $\delta_p^* = -\frac{\beta'}{\alpha'}$. Nous devons donc avoir $(r.DET)^2 \geq f(\delta_p^*)$. On se placera alors dans le cas extrême où $\delta_p = \delta_p^*$ (les λ sont pris aussi proches que possible). Les conditions prises précédemment nous garantissent $\delta_p^* \geq 0$.

De manière synthétique :
- δ_a et δ_b sont tirés aléatoirement dans $[1, U_\delta]$ tel que $\delta_b - \delta_a \leq 0$,
- p_0 est tiré aléatoirement dans $[1, U_p]$,
- a_0 et b_0 sont construits tels que

$$\begin{cases} \delta_a.b_0 - \delta_b.a_0 = DET \geq 0 \; et \\ 2.(a_0 - b_0 + \delta_a) - (\delta_b - \delta_a).(4.(n-3) + 3) \leq 0, \end{cases}$$

- $\delta_p = -\frac{\beta'}{\alpha'}$ et
- les capacités a et b sont construites de la même façon que dans le cas des problèmes non corrélés.

I.5 Conclusion

Dans ce chapitre, nous avons défini le problème du sac à dos multidimensionnel ainsi que les notations et le vocabulaire qui seront utilisés par la suite. De plus nous avons introduit :
– les instances de la OR-librairy de Beasley qui sont en libre accès sur internet et
– les différentes techniques pour engendrer de manière aléatoire des instances.
Ces instances seront utilisées dans la suite pour illustrer et tester les différentes méthodes de résolution que nous avons développées pour la résolution du MKP.

Nous avons aussi étudié deux nouvelles techniques pour engendrer aléatoirement des instances difficiles :
– instances dérivées de problèmes combinatoires en contrainte égalité et
– instances engendrées à partir de l'analyse de la transformée en $\mathbb{Z}$.
Les tests numériques sur cette étude préliminaire seront présentés dans le chapitre IV où nous comparerons les temps de résolution pour ces instances avec ceux d'instances engendrées par des techniques classiques.

Chapitre II

Etat de l'art

II.1 Introduction

Depuis les années 50, de nombreuses d'études ont été faites sur le problème du sac à dos. Dans ce chapitre nous faisons un état de l'art des travaux qui ont été menés.

Nous commencerons par les méthodes de relaxation, avec en particulier la relaxation surrogate introduite par Glover [GLO 65] que nous utiliserons par la suite. Ensuite, nous aborderons les méthodes de réduction. Ces méthodes essaient de réduire la taille du problème en fixant des variables ou en éliminant des contraintes. On citera notamment l'algorithme RAMBO de Fréville et Plateau [FRE 94]. Enfin, nous considérons diverses méthodes de résolution proposées dans la littérature :

- les méthodes approchées, qui permettent d'obtenir une solution approchée de la fonction objectif qui satisfait l'ensemble des contraintes. On s'intéresse en particulier aux heuristiques AGNES de Fréville et Plateau [FRE 94], ADP-based heuristics approach de Bertsimas et Demir [BER 02] et SMA de Hanafi, Fréville et El Abdellaoui [HAN 96] ;
- les méthodes classiques de résolution exacte, qui sont la programmation dynamique et le branch-and-bound ;
- les méthodes coopératives de résolution exacte dans le cas de problème à une contrainte, avec les méthodes proposées par Martello et Toth [MAR 82], Plateau et Elkihel [PLA 85], ainsi que Authié, Elkihel et Viader ([VIA 98] et[ELK 02]).

Les méthodes coopératives ont montré leur efficacité pour des problèmes à une seule contrainte.

II.2 Les méthodes de relaxations

les méthodes de relaxation sont généralement utilisées afin d'estimer un majorant de la valeur optimale du problème (MKP) considéré en se ramenant sur des problèmes pour lesquelles des méthodes de résolutions efficaces existent. Dans cette section nous présentons les principales relaxations existantes.

II.2.1 La relaxation continue

La relaxation continue consiste à relâcher la contrainte d'intégralité des variables :

$$(LP)\begin{cases} \max p^t.x, \\ \text{s.c. } W.x \leq c, \\ x \in [0,1]^n. \end{cases} \tag{II.1}$$

Le problème (LP) est résolu grâce à l'utilisation de la méthode du Simplexe ([DAN 48], [DAN 61]). Cependant, du fait que les variables sont bornées, il existe une variante de la méthode du simplexe (voir [DAN 67]) qui permet de diminuer l'occupation mémoire nécessaire pour la résolution numérique de (LP).

II.2.2 La relaxation lagrangienne

Une méthode très employée, qui n'a cependant pas été retenue dans le cadre de cette étude, se formule de la manière suivante :

$$(L_Q(\lambda)) \begin{cases} \max \sum_{j=1}^{n} (p_j - \sum_{i \in Q} \lambda_i . w_{i,j}) . x_i + \sum_{i \in Q} \lambda_i . c_i, \\ \text{s.c.} \sum_{j=1}^{n} w_{i,j} . x_j \leq c_i, \ i \in \{1, ..., m\} - Q, \\ x_j \in \{0, 1\}, \ j \in \{1, ..., n\}, \end{cases} \tag{II.2}$$

avec $Q \subset \{1, ..., m\}$, $q = |Q| \neq 0$ et $\lambda = (\lambda_i)_{i \in Q} \in \mathbb{R}_+^q$.

Geoffrion [GEO 74] et Fisher [FIS 81] ont montré que cette relaxation pouvait être utilisée de manière efficace pour la résolution de (MKP). De plus, en notant $v(L_Q) = \min_{\lambda \geq 0}(v(L_Q(\lambda)))$, on a $v(LP) \geq v(L_Q)$ et nous avons égalité pour $Q = \{1, ..., m\}$.

Pour $Q = \{1, ..., m\}$, notons :

$$(L(\lambda)) \begin{cases} \max p^t . x + \lambda^t (c - W.x), \\ x \in \{0, 1\}^n. \end{cases} \tag{II.3}$$

Ce problème a l'avantage d'être très facile à résoudre, avec une complexité de $o(n)$. En effet nous avons :

$$Pour \ j \in \{1, ..., n\}, \ si \ p_j - \sum_{i=1}^{m} \lambda_i . w_{i,j} \geq 0, \ x_j^*(\lambda) = 1, \ sinon \ x_j^*(\lambda) = 0 \ et$$

$$v(L(\lambda)) = \sum_{j=1}^{n} (p_j - \sum_{i=1}^{m} \lambda_i . w_{i,j}) . x_j^*(\lambda).$$

II.2.3 La relaxation surrogate ou agrégée

Une autre solution approchée peut être déterminée par la résolution du sac à dos unidimensionnel obtenu par la relaxation surrogate de (MKP), notée (S(μ)) introduite pour la première fois par Glover [GLO 65]. La relaxation surrogate consiste à combiner l'ensemble des contraintes en une seule de la manière suivante :

$$(S(\mu)) \begin{cases} \max p^t . x, \\ \text{s.c. } \mu^t . W . x \leq \mu^t . c, \\ x \in \{0, 1\}^n, \end{cases} \tag{II.4}$$

où $\mu^t = (\mu_1, \ldots, \mu_m) \geq 0$.

Cette relaxation a fait l'objet de nombreuses études afin de proposer des méthodes de calcul du multiplicateur et son utilisation pour la résolution du problème (MKP). On notera les travaux de Glover ([GLO 75] et [GLO 68]), Greenberg et Pierskela [GRE 70], Dyer [DYE 80], Gavish et Pirkul [GAV 85], Fréville et Plateau [FRE 93] et plus récemment Osario, Glover et Hammer [OSO 02] et Hanafi et Glover [HAN 07].

En notant, $v(S) = \min_{\mu \geq 0}\{v(S(\mu))\}$, Greenberg and Pierskalla [GRE 70] ont montré que dans tous les cas nous avons $v(LP) \geq v(S)$ et que l'inégalité stricte apparaît le plus souvent.

II.2.4 La relaxation surrogate continue

Une autre solution approchée, moins bonne que les deux précédentes mais peu coûteuse en calcul, est donnée par résolution du problème relâché, noté $\overline{S(\mu)}$:

$$(\overline{S(\mu)}) \begin{cases} \max p^t.x, \\ \text{s.c. } \mu^t.W.x \leq \mu^t.c, \\ x \in [0,1]^n, \end{cases} \tag{II.5}$$

où $\mu^t = (\mu_1, \ldots, \mu_m)$, $\mu_i \geq 0$ pour $i \in \{0, .., m\}$.

$(\overline{S(\mu)})$ peut être résolu via la méthode de Dantzig [DAN 57] :

Algorithme II.1 *Algorithme de Dantzig :*

Si les données sont classées selon :

$$\frac{p_1}{\mu^t.W^1} \geq \ldots \geq \frac{p_j}{\mu^t.W^j} \geq \ldots \geq \frac{p_n}{\mu^t.W^n},$$

La solution $\bar{x}$ *de* $(\overline{S(\mu)})$ *est :*
– $\overline{x_j} = 1$ *pour* $j \in \{1, \ldots, k-1\}$,
– $\overline{x_j} = 0$ *pour* $j \in \{k+1, \ldots, n\}$ *et*
– $\overline{x_k} = \frac{\bar{c}}{\mu^t.W^k}$,
avec $k = \max\{j \mid \sum_{p=1}^{j-1} \mu^t.W^p \leq c\}$ *et* $\bar{c} = c - \sum_{j=1}^{k-1} \mu^t.W^j$.
k correspond à l'indice de la variable de base.

Martello et Toth [MAR 90] ont proposé une évolution cet algorithme afin d'obtenir un meilleur majorant de la valeur optimale du problème (MKP). Ils fixent la variable de base x_k à 1, puis à 0, les deux sous-problèmes résultants sont résolus par l'algorithme précédent et la valeur retournée la plus petite est conservée.

En notant, $v(\overline{S}) = \min_{\mu \geq 0}\{v(\overline{S(\mu)})\}$, d'aprés Glover [GLO 75] nous avons $v(LP) = v(\overline{S})$.

II.2.5 La relaxation Composite

La relaxation composite est la combinaison de la relaxation lagrangienne et de la relaxation surrogate. Elle fut introduite par Greenberg and Pierskalla [GRE 70] :

$$(C(\lambda, \mu)) \begin{cases} \max \sum_{j=1}^{n} p_j.x_j + \lambda^t.(c - W.x), \\ \text{s.c. } \mu^t.W.x \leq \mu^t.c, \\ x_j \in \{0,1\} \text{ pour } j \in \{1, \ldots, n\}. \end{cases} \tag{II.6}$$

On note $v(C) = \min_{\lambda \geq 0, \mu \geq 0} \{v(C(\lambda, \mu))\}$. La relaxation lagrangienne (L) et la relaxation surrogate (S) sont clairement des cas particuliers de la relaxation composite. Nous avons donc, $v(L) \geq v(C)$ et $v(S) \geq v(C)$.

Crama et Mazzola [CRA 94] ont montré que :

$$v(LP) - \max_{j \in \{1,\ldots,n\}} \{p_j\} \leq v(C(\lambda, \mu)) \; et \; \frac{1}{2}.v(LP) \leq v(C(\lambda, \mu)). \tag{II.7}$$

Autrement dit, la relaxation composite et donc les relaxations lagrangienne et surrogate ne peuvent améliorer la borne de (LP) plus que p_{max}, le plus grand profit de la fonction objectif.

Hanafi et Fréville [HAN 00] ont proposé une méthode de résolution pour la relaxation composite.

II.2.6 Choix des multiplicateurs

L'utilisation de ces différentes méthodes de relaxation nécessite le calcul des multiplicateurs. Un multiplicateur μ est dit meilleur que le multiplicateur μ' lorsqu'il fournit une meilleure borne supérieure. Autrement dit, le multiplicateur optimal est celui qui minimise la borne supérieure.

Les multiplicateurs optimaux sont obtenus par résolution des problèmes duaux :
- $\min_{\lambda \geq 0}\{v(L(\lambda))\} = v(L)$ dans le cas d'une relaxation lagrangienne,
- $\min_{\mu \geq 0}\{v(S(\mu))\} = v(S)$ dans le cas d'une relaxation surrogate,
- $\min_{\mu \in \geq 0}\{v(\overline{S(\mu)})\} = v(\overline{S})$ dans le cas d'une relaxation surrogate continue,
- $\min_{\lambda \geq 0, \mu \geq 0}\{v(C(\lambda, \mu))\} = v(C)$ dans le cas d'une relaxation Composite.

Nous avons la relation suivante entre le bornes des problèmes duaux :

$$v(MKP) \leq v(C) \leq v(S) \leq v(L) \leq v(LP) = v(\overline{S}). \tag{II.8}$$

II.3 Les méthodes de réduction

Les méthodes présentées ici essaient à travers différentes techniques de diminuer la complexité de résolution du problème du sac à dos. Deux approches sont abordées :
- la réduction de la taille du problème (c'est-à-dire le nombre de variables et de contraintes du problème), car la complexité des algorithmes de résolution y est souvent liée,
- la réduction du domaine de définition de l'ensemble des solutions réalisables en continu, ce qui permet notamment l'amélioration de la qualité des bornes calculées.

Pour la première approche, Fréville et Plateau [FRE 94] ont proposé un ensemble de méthodes efficaces pour la réduction de la taille du problème. Ces méthodes, dont les performances sont liées à la qualité de l'estimation d'une borne inférieure, s'accompagnent de l'heuristique AGNES (voir page 37).

En ce qui concerne la deuxième approche, les méthodes de rotation de contraintes, introduites par Kianfar [KIA 71], permettent de modifier les contraintes originales du problème afin de resserrer le domaine continu sur l'ensemble des solutions entières admissibles.

Une autre méthode très utilisée pour réduire le domaine de définition des solutions en continu est l'utilisation des méthodes de coupes. Ces méthodes sont aujourd'hui largement utilisées dans les solveurs industriels tel que CPLEX ou XPRES. Ici, le principe est basé sur l'ajout de contraintes additionnelles afin d'éliminer des solutions non-entières. Ces coupes sont généralement issues de l'analyse du dernier tableau de la méthode du simplexe, obtenu par la résolution de la relaxation continue de (MKP). Si une solution non-entière est obtenue, des coupes peuvent être ajoutées afin d'éliminer cette solution du domaine continu et d'essayer de faire converger la valeur optimale en continue vers la valeur optimale en discret, voir même d'obtenir une solution entière, dans ce cas la valeur optimale en continue est aussi optimale en discret.

Les méthodes de coupes nécessitent souvent une grande précision de calcul, nous nous sommes donc, pour l'instant, peu intéressés à ces méthodes, et nous ne nous étendrons pas plus sur elles. On peut citer notamment les travaux de Gomory [GOM 58] [GOM 63], de Cook, Kannan et Schrijver [COO 90], de Nemhauser et Wolsey [NEM 88] [NEM 90], de Balas, Ceria et Cornuéjols [BAL 93], de Cornuejols, Li et Vandenbussche [COR 03], de Andersen, Cornuejols et Li [AND 05] et de Cornuejols [COR 07].

II.3.1 Les fixations de variables

Ces méthodes permettent de fixer de façon définitive certaines variables à 0 ou à 1.

Réductions à partir des bornes supérieures

A chaque variable x_j ($j \in \{1, \ldots, n\}$), on associe deux bornes supérieures, à savoir :
– $\overline{v}(x_j = 1)$, un majorant de la valeur optimale de (MKP) lorsque x_j est fixé à 1,
– $\overline{v}(x_j = 0)$, un majorant de la valeur optimale de (MKP) lorsque x_j est fixé à 0.
Notons alors $\underline{v}$ un minorant de la valeur optimale du problème (MKP), et $\underline{x}^t = (\underline{x}_1, \ldots, \underline{x}_n)$, la solution associée. Nous avons donc le schéma de réduction suivant :

Proposition II.1 - *Réduction de variable 1 :*
Pour $j \in \{1, \ldots, n\}$,
– *si* $\overline{v}(x_j = 0) \leq \underline{v}$, *alors on peut fixer* x_j *à 1,*
– *si* $\overline{v}(x_j = 1) \leq \underline{v}$, *alors on peut fixer* x_j *à 0.*

Le calcul de ces bornes supérieures peut se faire au travers d'une méthode de relaxation, sachant que la relaxation continue présente un avantage du point de vue de temps de résolution du fait de l'utilisation de la méthode du simplexe.

De plus, on peut noter ici que pour un problème (MKP) donné, si on souhaite intégrer cette méthode de réduction à un algorithme de résolution tel que le branch-and-bound, nous devons calculer $2.n$ bornes supérieures, ce qui peut représenter des temps de calcul non négligeables. Afin d'essayer d'optimiser cette étape, nous pouvons remarquer la chose suivante :

Soit $\overline{v}$, une borne supérieure du problème (MKP), et $\overline{x}^t = (\overline{x}_1, \ldots, \overline{x}_n)$, la solution associée. Notons alors $J_1 = \{j \in \{1, ..., n\} \mid \overline{x}_j = 1\}$ et $J_0 = \{j \in \{1, ..., n\} \mid \overline{x}_j = 0\}$. Alors, nous avons :
– pour $j \in J_1$, $\overline{v}(x_j = 1) = \overline{v}$ et
– pour $j \in J_0$, $\overline{v}(x_j = 0) = \overline{v}$.
Ce qui donne $(2.n - |J_1| - |J_0| + 1)$ bornes supérieures à calculer.

Réductions à partir des profits réduits

Cette méthode se base sur la résolution exacte de la relaxation continue (II.1) de (MKP) par la méthode du simplexe. Pour cela nous devons la transformer pour avoir que des contraintes égalités, nous devons donc introduire des variables d'écarts $(e_1, ..., e_{n+m})$:

$$(LP) \begin{cases} \max \sum_{j=1}^{n} p_j.x_j, \\ \text{s.c.} \begin{cases} \sum_{j=1}^{n} w_{i,j}.x_j + e_i = c_i \text{ pour } i \in \{1, ..., m\}, \\ x_j + e_{m+j} = 1 \text{ pour } j \in \{1, ..., n\}, \\ x_j \geq 0 \text{ pour } j \in \{1, ..., n\} \text{ et } e_j \geq 0 \text{ pour } j \in \{1, ..., n+m\}. \end{cases} \end{cases} \tag{II.9}$$

Ce qui peut se noter sous forme matricielle :

$$(LP) \begin{cases} \max \tilde{p}^t.\tilde{x}, \\ \text{s.c. } \tilde{W}.\tilde{x} = \tilde{c}, \\ \tilde{x} \geq 0, \end{cases} \tag{II.10}$$

avec :

$$\tilde{x} = \begin{pmatrix} x \\ e_1 \\ \vdots \\ e_{n+m} \end{pmatrix}, \quad \tilde{p} = \begin{pmatrix} p \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \tilde{c} = \begin{pmatrix} c \\ 1 \\ \vdots \\ 1 \end{pmatrix} \text{ et}$$

$$\tilde{W} = \begin{pmatrix} W & 1_m & O_{(m,n)} \\ 1_n & O_{(n,m)} & 1_n \end{pmatrix}.$$

où :
– 1_m et 1_m représentent les matrices unités de dimension respective $m \times m$ et $n \times n$,
– $O_{(m,n)}$, la matrice nulle de dimension $m \times n$ et
– $O_{(n,m)}$, la matrice nulle de dimension $n \times m$.
Si l'on note alors $\tilde{x}_B$, les variables de bases, et $\tilde{x}_H$, les variables hors-bases, auxquelles on associe les vecteurs profits correspondants, $\tilde{p}_B$ et $\tilde{p}_H$, ainsi que les matrices de poids, $\tilde{W}_B$ et $\tilde{W}_H$, nous pouvons écrire (II.10) sous la forme :

$$\begin{aligned} (LP) &\begin{cases} \max\ \tilde{p}_B^t.\tilde{x}_B + \tilde{p}_H^t.\tilde{x}_H, \\ \text{s.c. } \tilde{W}_B.\tilde{x}_B + \tilde{W}_H.\tilde{x}_H = \tilde{c}, \\ \tilde{x} \geq 0. \end{cases} \\ \Leftrightarrow (LP) &\begin{cases} \max\ \tilde{p}_B^t.\tilde{W}_B^{-1}.\tilde{c} + (\tilde{p}_H^t - \tilde{p}_B^t.\tilde{W}_B^{-1}.\tilde{W}_H).\tilde{x}_H, \\ \text{s.c. } \tilde{x}_B + \tilde{W}_B^{-1}.\tilde{W}_H.\tilde{x}_H = \tilde{W}_B^{-1}.\tilde{c}, \\ \tilde{x} \geq 0. \end{cases} \end{aligned} \tag{II.11}$$

On définit alors $p_{red} = (p_{red1}, ..., p_{redn})^t$, le vecteur des profits réduits associé à la variable de décision x, par :

$$p_{red} = (\tilde{p}_H^t - \tilde{p}_B^t.\tilde{W}_B^{-1}.\tilde{W}_H). \tag{II.12}$$

Remarque: *Le profit réduit associé à une variable de base vaut* 0.

Reprenons alors les notations précédentes, à savoir, $\overline{v}$, une borne supérieure du problème (MKP) qui correspond à la valeur optimale du problème (LP) (équation II.1), $\overline{x}^t = (\overline{x}_1, \ldots, \overline{x}_n)$ une solution associée, et $\underline{v}$ une borne inférieure du problème (MKP). Nous avons alors le schéma de réduction suivant :

Proposition II.2 - *Réduction de variable 2 :*
Pour $j \in \{1, ..., n\}$, *si* $\overline{x}_j \in \{0, 1\}$ *et* $\overline{v} - |p_{redj}| \leq \underline{v}$, *alors on peut fixer de manière définitive* x_j *à* $\overline{x}_j$.

II.3.2 Les réductions de contraintes

L'objectif ici est d'écarter certaines contraintes redondantes, c'est-à-dire que leur élimination n'altère pas la valeur optimale du problème. Nous détaillons ci-dessous trois méthodes classiques de réduction.

Proposition II.3 - *Réduction de contraintes 1 :*
Pour $i \in \{1, \ldots, m\}$, *si* $\sum_{j=1}^{n} w_{i,j} \leq c_i$, *alors la* $i^{ème}$ *contrainte peut être éliminée.*

Cette réduction nous permet de nous placer dans le cas de l'hypothèse (h2) vu précédemment (page 16).

Proposition II.4 - *Réduction de contraintes 2 :*
Pour $i \in \{1, \ldots, m\}$, *si* $\exists k \in \{1, \ldots, i-1, i+1, \ldots, m\}$ *tel que* $\forall j \in \{1, \ldots, n\}$, $w_{i,j} \leq w_{k,j}$ *et* $c_k \leq c_i$, *alors la* $i^{ème}$ *contrainte peut être éliminée.*

En effet, le respect de la contrainte k implique celui de la contrainte j et nous conservons la contrainte la plus forte.

Proposition II.5 - *Réduction de contraintes 3 :*
Pour $i \in \{1, \ldots, m\}$, *si* $\exists k \in \{1, \ldots, i-1, i+1, \ldots, m\}$ *tel que* $v(B_i^k) \leq c_i$, *alors la* $i^{ème}$ *contrainte peut être éliminée, avec*

$$(B_i^k) \begin{cases} \max\ W_i.x, \\ s.c.\ W_k.x \leq c_k, \\ x \in \{0,1\}^n. \end{cases}$$

Ce test est un peu plus coûteuse en temps de calcul que les précédents car il nécessite la résolution exacte de problèmes du sac à dos à une contrainte. On pourrait éventuellement estimer à chaque fois qu'une borne supérieure pour (B_i^k), mais on dégraderait les performances de cette méthode de réduction.

II.3.3 L'algorithme RAMBO

L'algorithme RAMBO (Reduction algorithm for the Multidimenssional Binary variables Optimization problem) de Fréville et Plateau [FRE 94] est constitué d'un ensemble d'étapes successives visant à réduire efficacement la taille du problème. Ces différentes étapes sont :

– Étape 1 : bonne formulation du problème,
– Étape 2 : réduction de variables,
– Étape 3 : élimination de contraintes.
Ils reprennent une partie des méthodes présentées précédemment.

Étape 1 : bonne formulation du problème

Cette première étape vérifie simplement la bonne formulation du problème à savoir :

$$\begin{cases} \forall i \in \{1,...,m\},\ \max\limits_{j\in\{1,...,n\}}\{w_{i,j}\} \leq c_i \leq \sum\limits_{j=1}^{n} w_{i,j}, \\ \forall j \in \{1,...,n\},\ \exists i \in \{1,...,n\}\ tel\ que\ w_{i,j} \neq 0. \end{cases} \tag{II.13}$$

On effectue pour cela les tests suivants :

[R1] $\forall j \in \{1,...,n\},\ si\ \exists i \in \{1,...,m\}\ tel\ que\ w_{i,j} > c_i,\ alors\ x_j = 0,$

[R2] $\forall i \in \{1,...,m\},\ si \sum\limits_{j=1}^{n} w_{i,j} \leq c_i,\ alors\ la\ contrainte\ i\ peut-être\ éliminée,$

[R3] $\forall j \in \{1,...,n\},\ si\ \forall i \in \{1,...,m\}\ w_{i,j} = 0,\ alors\ x_j = 1.$

La deuxième partie de cette étape essaie de remplacer les contraintes originales par des contraintes plus fortes, c'est-à-dire définissants un domaine contenant les mêmes solutions entières avec moins de solutions réelles (Bragley, Hammer et Wolsey [BRA 74]) :

[R4] $\forall i \in \{1,...,m\},\ si\ c_i \geq \sum\limits_{j=1}^{n} w_{i,j} - \max\limits_{j\in\{1,...,n\}}\{w_{i,j}\},\ on\ définit\ alors\ K_i \subseteq N = \{1,...,n\}\ tel$
$que,\ \forall k \in K_i,\ w_{i,k} \geq \delta_i = \sum\limits_{j=1}^{n} w_{i,j} - c_i.$
$Si\ K_i \neq \emptyset,\ alors\ la\ contrainte\ i\ doit\ être\ remplacée\ par :$

$$\delta_i.\sum_{j\in K_i} x_j \ + \sum_{j\in N-K_i} w_{i,j}.x_j \leq (|K_i|-1).\delta_i \ + \sum_{j\in N-K_i} w_{i,j}.$$

Ce dernier test a une complexité temporelle de $O(m.n)$.

Étape 2 : réduction de variables

Cette étape se base sur la proposition II.1. Son efficacité est fortement liée à la qualité de la borne inférieure, mais aussi à celle de la borne supérieure. Les bornes supérieures vont être estimées avec une complexité de plus en plus importante afin de réduire un maximum de variables.

On considère la relaxation surrogate $(S(\mu))$ de (MKP), que l'on notera par souci de simplification :

$$(S(\mu)) :\ \max\left\{p^t.x\ s.c.\ a.x \leq a_0,\ x \in \{0,1\}^n\right\},\ avec\ a = \mu^t.W\ et\ a_0 = \mu^t.c.$$

μ est calculé via un algorithme subgradient. Il est initialisé par un multiplicateur quelconque. Itérativement, l'algorithme essaie d'améliorer ce multiplicateur afin de converger vers un multiplicateur optimal. Pour cela, $(S(\mu))$ est résolu de façon exacte et le multiplicateur μ est perturbé

en fonction des contraintes violées et des contraintes respectées, puis on résout à nouveau la nouvelle relaxation et ainsi de suite. L'algorithme s'arrête lorsque $v(S(\mu))$ ne décroît plus (voir l'algorithme III.3, page 73).

Le premier test se base sur la relaxation lagrangienne $(L(\lambda))$, $\lambda \in \mathbb{R}$, de $(S(\mu))$. D'après les résultats de Hammer, Padberg et Peled [HAM 75], pour fixer une variable x_j à partir de $(L(\lambda))$, il suffit de prendre $\lambda \in \Lambda$:

$$\Lambda = \left\{ \frac{p_j}{a_j} \mid j \in \{1, ..., n\} \right\}.$$

On note :
– $\overline{(S(\mu))}$, la relaxation continue de $(S(\mu))$,
– j^* l'indice de base de $\overline{(S(\mu))}$ définit par la méthode de Dantzig et
– $\underline{v}(MKP)$ une borne inférieure de (MKP),
Un multiplicateur optimal de la relaxation lagrangienne de $\overline{(S(\mu))}$ est alors défini par $\lambda_{j^*} = \frac{p_{j^*}}{a_{j^*}}$.

Le test V1 s'énonce alors comme suit :

[V1] $\forall j \in \{1, ..., n\},\ j \neq j^*,$ *si* $|p_j - \lambda_{j^*}.a_j| \geq v(L(\lambda_{j^*})) - \underline{v}(MKP),$ *alors* $x_j = 1,$ *respectivement* $x_j = 0,$ *si* $p_j - \lambda_{j^*}.a_j > 0,$ *respectivement* $p_j - \lambda_{j^*}.a_j < 0.$

Le test V2 s'intéresse aux autres multiplicateur de lagrange. Cependant, afin de limiter la complexité temporelle qui est $O(n^2)$, au lieu de considérer $\lambda \in \Lambda$, on prendra $\Lambda \cap [\lambda^0_{j^*}, \lambda^1_{j^*}]$ avec :
– $\lambda^0_{j^*}$ le multiplicateur optimal de la relaxation lagrangienne de $\overline{(S(\mu) \mid x_{j^*} = 0)}$ et
– $\lambda^1_{j^*}$ le multiplicateur optimal de la relaxation lagrangienne de $\overline{(S(\mu) \mid x_{j^*} = 1)}$.

[V2] $\forall j \in \{1, ..., n\},$ *si* $\exists i \in \{1, ..., n\},\ i \neq j,$ *tel que* $p_j - \lambda_i.a_j \geq v(L(\lambda_i)) - \underline{v}(MKP),$ *alors* $x_j = 1,$ *respectivement* $x_j = 0,$ *si* $p_j - \lambda_i.a_j > 0,$ *respectivement* $p_j - \lambda_i.a_j < 0.$

Dans le test V3, les bornes sont améliorées en faisant une séparation sur la variable de base j^* :

[V3] $\forall j \in \{1, ..., n\},\ j \neq j^*,$ *si* $\exists (i_0, i_1) \in \{1, ..., n\}^2,$ *tel que* :
– $(p_j - \lambda_{i_0}.a_j)$ *et* $(p_j - \lambda_{i_1}.a_j)$ *ont le même signe,*
– $|p_j - \lambda_{i_0}.a_j| \geq v(L(\lambda_{i_0}) \mid x_{j^*} = 0) - \underline{v}(MKP)$ *et*
– $|p_j - \lambda_{i_1}.a_j| \geq v(L(\lambda_{i_1}) \mid x_{j^*} = 1) - \underline{v}(MKP),$
alors $x_j = 1,$ *respectivement* $x_j = 0,$ *si le signe est positif, respectivement négatif.*

En pratique, pour $j \in \{1, ..., n\},\ j \neq j^*$, ce test est effectué pour $i_0 = i^*_0$ et $i_1 = i^*_1$, représentant, respectivement, les indices de base des sous-problèmes $(S(\mu) \mid x_{j^*} = 0, x_j = \epsilon)$ et $(S(\mu) \mid x_{j^*} = 1, x_j = \epsilon)$, $\epsilon \in \{0, 1\}$.

Le test V4, qui s'appuie sur les propriétés des profits réduits, s'énonce comme suit :

[V4] $\forall j \in \{1, ..., n\}$:
– *choisir* $\lambda_{i(j)}$ *tel que* :

$$v(L(\lambda_{i(j)})) - \underline{v}(MKP) - |p_j - \lambda_{i(j)}.a_j| = \min_{i \in \{1,...,n\}} \{v(L(\lambda_i)) - \underline{v}(MKP) - |p_j - \lambda_i.a_j|\};$$

– *hypothèse* : $x_j = \epsilon,$ *avec* $\epsilon = 0$ *si* $p_j - \lambda_{i(j)}.a_j < 0,$ $\epsilon = 1$ *si* > 0;
– *choisir un autre* $\lambda_i,$ $i \neq i(j),$ *selon le même critère que précédemment et construire* X^0_j *et* X^1_j *l'indice des variables fixées respectivement à* 0 *et à* 1, *sous la condition* $x_j = \epsilon$: *si* $v(L(\lambda_i) \mid x_j = \epsilon,\ x_l = \eta,\ x_k = 0,\ \forall k \in X^0_j,\ x_k = 1,\ \forall k \in X^1_j) \leq \underline{v}(MKP),$ *alors* $X^0_j = X^0_j \cup \{l\},$ *si* $\eta = 1,$ $X^1_j = X^1_j \cup \{l\},$ *si* $\eta = 0$;

- *si* $(MKP \mid x_k = 0,\ \forall k \in X_j^0,\ x_k = 1,\ \forall k \in X_j^1)$ *ne possède aucune solution réalisable alors* $x_j = 1 - \epsilon$;
- *si* $v(L(\lambda_i) \mid x_j = \epsilon,\ x_k = 0,\ \forall k \in X_j^0,\ x_k = 1,\ \forall k \in X_j^1) \leq \underline{v}(MKP)$, *alors* $x_j = 1 - \epsilon$.

Et enfin le test V5 consiste à résoudre $(S(\mu) \mid x_j = \epsilon)$, $\epsilon \in \{0,1\}$, mais seulement pour les variables x_j vérifiant une des inégalités du test V3 :

[V5] *Pour* $j \in \{1,...,n\}$ *et* $\epsilon \in \{0,1\}$, *si* $v(S(\mu) \mid x_j = \epsilon) \leq \underline{v}(MKP)$, *alors* $x_j = 1 - \epsilon$.

Étape 3 : élimination de contraintes

On considère les problèmes suivant :

$$pour\ k \in \{1,...,m\},\ (K_k) :\ \max\{W_k.x\ s.c.\ x \in E_k\},$$

avec Ek définit selon les deux possibilités suivantes :
(i) $E_k = \{x \in \{0,1\}^n \mid W_p.x \leq c_p,\ p \neq k\}$,
(ii) $E_k = \{x \in \{0,1\}^n \mid \mu.W.x \leq \mu.c,\ \mu \in \mathbb{R}_+^m,\ u_k = 0\}$,
où μ est défini par une technique duale et p est tel que $\mu_p = \max\limits_{i \in \{1,...,m\}} \{\mu_i\}$.

Dans un premier temps on s'intéresse à la relaxation continue de (K_k), $\overline{(K_k)}$.

[C1] *Si* $\left\lfloor v(\overline{(K_k)}) \right\rfloor \leq c_k$, *alors la contrainte k peut être éliminée.*

[C2] *Soit* x_{j^*} *la variable de base de* $\overline{(K_k)}$:
si $\alpha = \max\left\{ \lfloor v(\overline{(K_k \mid x_{j^*} = 0)}) \rfloor,\ \lfloor v(\overline{(K_k \mid x_{j^*} = 1)}) \rfloor \right\} \leq c_k$, *alors la contrainte k peut être éliminée.*

Si $\alpha > c_k$, mais $\lfloor v(\overline{(K_k \mid x_{j^*} = 1 - \epsilon)}) \rfloor \leq c_k$, avec $\epsilon \in \{0,1\}$, alors on effectue le test suivant :

[C3] *Soit* $x_{j^*(\epsilon)}$ *la variable de base de* $\overline{(K_k \mid x_{j^*} = \epsilon)}$:
si $\alpha = \max\left\{ \lfloor v(\overline{(K_k \mid x_{j^*} = \epsilon,\ x_{j^*(\epsilon)} = 0)}) \rfloor,\ \lfloor v(\overline{(K_k \mid x_{j^*} = \epsilon,\ x_{j^*(\epsilon)} = 0)}) \rfloor \right\} \leq c_k$, *alors la contrainte k peut être éliminée.*

Le dernier test, C4, résout de façon exacte (K_k) :

[C4] *Si* $v((K_k)) \leq c_k$, *alors la contrainte k peut être éliminée.*

II.3.4 La rotation de contraintes

Les méthodes de rotation de contraintes (Elkihel et Plateau [ELK 84a] [ELK 86a] [ELK 86b], Kianfar [KIA 71], [KIA 76] et Williams [WIL 74]) permettent de reformuler les contraintes d'un problème (MKP) avec exactement le même ensemble de solutions réalisables. Le but étant de resserrer le domaine continu défini par les contraintes sur le domaine défini par les solutions entières réalisables.

Walukkiewicz et Kaliszewski [WAL 76] et Williams [WIL 74] ont montré que la résolution est d'autant plus facile que le domaine de définition est réduit.

Cette méthode consiste donc à déplacer les contraintes de (MKP) de sorte qu'elles passent par au moins autant de points entiers que les originales. Deux approches sont mises en avant :

– la rotation sans ajout de contrainte additionnelle : Méthode de Kianfar [KIA 71],
– la rotation avec ajout de contrainte additionnelle : Méthode de Elkihel et Plateau [ELK 84a] [ELK 86a] [ELK 86b].

Rotation sans contraintes additionnelles [KIA 71]

Soit $p \in \{1, ..., m\}$, et la $p^{ème}$ contrainte $W_p.x \leq c_p$ de (MKP).

La rotation de la $p^{ème}$ contraintes de (MKP) consiste donc à déplacer l'hyperplan $W_p.x = c_p$ afin que le nouvel hyperplan $\widehat{W}_p.x = c_p$ ainsi construit satisfasse :

$$\{x \in \{0,1\}^n \mid W_p.x = c_p\} \subset \{x \in \{0,1\}^n \mid \widehat{W}_p.x = c_p\}. \tag{II.14}$$

Soit $r \in \{1, ..., n\}$. On va considérer le problème suivant :

$$(B_p^p(x_r = 1)) \begin{cases} \max\ w_{p,r} + \sum\limits_{\substack{j=1 \\ j \neq r}}^{n} w_{p,j}.x_j, \\ \text{s.c.} \sum\limits_{\substack{j=1 \\ j \neq r}}^{n} w_{p,j}.x_j \leq c - w_{p,r}, \\ x_j \in \{0,\ 1\},\ j \in \{1, ..., n\}. \end{cases} \tag{II.15}$$

Les poids du nouvel hyperplan sont alors construites itérativement de la manière suivante :

$$\widehat{w}_{p,r} = w_{p,r} + c_p - v(B_p^p(x_r = 1)). \tag{II.16}$$

Ainsi si $\widehat{w}_{p,r} > w_{p,r}$, le nouvel hyperplan, $\widehat{w}_{p,r}.x_r + \sum\limits_{\substack{j=1 \\ j \neq r}}^{n} w_{p,j}.x_j = c_p$, passe par le nouveau point entier $\widehat{x} = \{\widehat{x}_1, ..., \widehat{x}_n\}$:

$$\widehat{x}_r = 1 \ et\ \widehat{x}_j = x_j^* \ pour\ j \in \{1, ..., n\} - \{r\},$$

avec x^* une solution optimale de $(B_p^p(x_r = 1))$.

L'enjeu est alors de choisir une bonne direction pour la rotation à savoir dans quel ordre sera calculé les poids $\widehat{w}_{p,r}$. En effet, de cette direction dépendra la qualité de la rotation (voir Elkihel [ELK 84a] et Walukiewicz et Kaliszewski [WAL 76]).

Rotation avec contraintes additionnelles [ELK 84a] [ELK 86a] [ELK 86b]

La méthode consiste à faire une rotation avec l'ajout d'une contrainte additionnelle. Elle permet de réduire davantage le domaine de définition de l'ensemble des solutions admissibles de (MKP).

Soit $(p,q) \in \{1,...,m\}^2$ et $r \in \{1,...,n\}$. Le principe est ici le même, nous allons effectuer la rotation de la contrainte $W_p.x \leq c_p$ sous la contrainte additionnelle $W_q.x \leq c_q$. Nous allons donc considérer le problème suivant :

$$(B^p_{p,q}(x_r=1)) \begin{cases} \max\ w_{p,r} + \sum\limits_{\substack{j=1 \\ j\neq r}}^{n} w_{p,j}.x_j, \\ \text{s.c.} \begin{cases} \sum\limits_{\substack{j=1 \\ j\neq r}}^{n} w_{p,j}.x_j \leq c - w_{p,r}, \\ \sum\limits_{\substack{j=1 \\ j\neq r}}^{n} w_{q,j}.x_j \leq c - w_{q,r}, \end{cases} \\ x_j \in \{0,\ 1\},\ j \in \{1,...,n\}. \end{cases} \tag{II.17}$$

Ainsi, nous construisons de la même manière les poids du nouvel hyperplan :

$$\widehat{w}_{p,r}(q) = w_{p,r} + c_p - v(B^p_{p,q}(x_r=1)). \tag{II.18}$$

La problématique est alors de choisir la contrainte additionnelle q en fonction de la contrainte p sur laquelle nous voulons faire une rotation, à savoir celle qui induira le maximum de déviation. Elkihel et Plateau suggèrent de choisir q tel que :

$$\frac{c_q}{\sum\limits_{j=1}^{n} w_{q,j}}.\cos^2(W_p,W_q) = \min_{\forall i\neq p}\left\{\frac{c_i}{\sum\limits_{j=1}^{n} w_{i,j}}.\cos^2(W_p,W_i)\right\}. \tag{II.19}$$

II.4 Les méthodes heuristiques

Les heuristiques proposées seront comparées à des heuristiques de la littérature :
– AGNES de Fréville et Plateau [FRE 94] ;
– ADP-based heuristics approach de Bertsimas et Demir [BER 02] ;
– Simple Multistage Algorithm (SMA) de Hanafi, Fréville et El Abdellaoui [HAN 96].
Notre choix s'est porté sur ces heuristiques pour leurs performances et/ou leur construction proche des heuristiques que nous proposons. Nous allons faire le point sur ces différentes approches.

II.4.1 AGNES

L'heuristique AGNES de Fréville et Plateau [FRE 94] se base sur différentes méthodes proposées par Glover ([GLO 75] et [GLO 77]). AGNES génère une série de solutions réalisables à partir de différentes techniques exploitant les informations fournies par plusieurs types de contraintes surrogate.

L'heuristique est initialisée avec un multiplicateur $\mu \in \mathbb{R}_+^m$. AGNES est décomposée en les étapes suivantes :

- AGNES0 génère une première solution par une méthode gloutonne avec les variables classées selon les rapports $\frac{p_j}{\mu^t.W^j}$ décroissants.
- AGNES1 construit une solution à partir de la solution optimale $\overline{x}$ d'une relaxation surrogate continue perturbée $\overline{S_\epsilon(\mu)}$: max $p^t.x$ *s.c.* $\mu^t.W.x \leq (1+\epsilon).\mu^t.c$ *et* $x \in [0,1]^n$. Les variables j, tel que $\overline{x}_j = 1$, sont, dans un premier temps, temporairement fixées à 1 et les autres à 0. Si la solution obtenue n'est pas réalisable, une partie des variables fixées à 1 sont libérées par une méthode gloutonne, en se basant sur une procédure décrite par Senju et Toyoda [SEN 68] (voir algorithme II.5, page 42), afin de revenir dans le domaine des solutions réalisables. Les variables qui sont restées à 1 sont définitivement fixées. La procédure est alors réitérée sur le sous-problème défini par les variables libres. Plusieurs solutions sont générées en prenant différentes valeurs de ϵ dans $[-0,2; 0,2]$.
- AGNES2 tente de fixer itérativement les variables à partir des profits réduits. La procédure fixe $\lfloor \alpha.n \rfloor$ variables, $\alpha \in]0,1[$, à partir des profits réduits de la relaxation surrogate continue $\overline{S(\mu)}$: max $p^t.x$ *s.c.* $\mu^t.W.x \leq \mu^t.c$ *et* $x \in [0,1]^n$. Si le profit réduit est positif, respectivement négatif, la variable correspondante est fixée à 1, respectivement à 0. La procédure est alors réitérée sur le sous-problème définie par les variables libres. Plusieurs solutions sont générées en prenant différentes valeurs de α dans $\left\{\frac{1}{3}, \frac{1}{4}, \frac{1}{5}\right\}$.
- AGNES3 est un *k-interchange heuristic* qui essaye de trouver une meilleure solution parmi les solutions, $\underline{x}$, retournées par les précédentes heuristiques. Cette solution est recherchée dans le voisinage $\left\{ x \in \{0,1\}^n \;\middle|\; \sum_{j \in J} |x_j - \underline{x}_j| = k,\ J = \{j \mid \underline{x}_j = 1\} \right\}$. Afin d'avoir des temps de calcul petits k a été fixé à 1 par les auteurs.

L'ensemble de ces heuristiques constitue l'heuristique AGNES. Elle est exécutée plusieurs fois avec une initialisation différente du multiplicateur μ qui est calculé successivement avec :

- la méthode structurale (voir page 68),
- le dual du simplexe (voir page 66),
- la solution optimale du dual surrogate $(S) : \min_{\mu \geq 0}\{v(S(\mu))\}$.

AGNES a été comparée avec les heuristiques MKHEUR de Pirkul [PIR 87] et GP-procedure de Gavish et Pirkul [GAV 85]. Les tests numériques ont montré qu'AGNES permet d'avoir un bon compromis entre temps de calcul et qualité de la borne.

II.4.2 L'heuristique ADP-BH

L'ADP-BH, ADP pour Approximate Dynamic Programming et BH pour Base Heuristics, de Bertsimas et Demir [BER 02] est, comme son nom l'indique, basé sur un algorithme de programmation dynamique. Nos heuristiques étant aussi basées sur le même type d'algorithme, il était intéressant de pouvoir comparer nos résultats avec celui là.

Si l'on considère le sous-problème $(MKP(k, c'))$ qui comprend les k premières variables de (MKP) avec la capacité c' :

$$(MKP(k,c'))\begin{cases} \max \sum_{j=1}^{k} p_j.x_j, \\ \text{s.c.} \sum_{j=1}^{k} w_{i,j}.x_j \leq c'_i, \ \forall i \in \{1,...,m\}, \\ x_j \in \{0,1\}, \ \forall j \in \{1, \ ..., \ k\}, \end{cases} \tag{II.20}$$

On peut donc établir les deux relations récursives suivantes dans le cadre de la résolution par programmation dynamique pour $k \in \{2, \ ..., \ n\}$:

$$v(MKP(k,c')) = max\{v(MKP(k-1,c')), \ v(MKP(k-1,c'-W^k)+p_k)\}, \tag{II.21}$$

$$x_k^* = argmax_{x_k \in \{0, \ 1\}}\{v(MKP(k-1,c'-W^k.x_k))+p_k.x_k\}. \tag{II.22}$$

Cependant, dans l'ADP-BH, au lieu de considérer les valeurs optimales dans l'équation II.22 ($v(MKP(k-1,c'-W^k.x_k))$), ce qui reviendrait à faire une programmation dynamique classique, Bertsimas et Demir ont utilisé des méthodes approchées pour estimer cette borne. Ceci qui permet alors d'évaluer x_k^* à chaque étape k de l'algorithme (équation II.22). Cette étape est appelée *Variable assignment*.

Plusieurs choix ont été proposés et testés pour cette heuristique, appelée Base Heuristique par Betsimas et Démir, notamment :

– *Dual Gradient Algorithm* de Senju et Toyoda [SEN 68] qui construit, à partir d'une solution non-réalisable, une solution réalisable en fixant les variables ayant le plus petit gradient à 0. Le gradient de la variable j est défini comme suit :

$$G_j = \frac{p_j}{(W^j)^t.S} \ avec \ S = \left(\sum_{k=1}^{n} w_{i,k} - c_i\right)_{\{i \in \{1,...,m\}\}}.$$

– *Primal Gradient Algorithm* de Toyoda [TOY 75] qui à partir de la solution nulle (sac à dos vide) fixe itérativement les variables à 1 selon leur gradient décroissant.
– *Greedy-like Heuristics* de Loulou et Michaelides [LOU 79] qui est une variation du Primal Gradient Algorithm avec un gradient plus évolué.
– *Incremental Heuristic* de McCarl, Kochembergs et Wymann [MCC 74] qui est une autre variation du Primal Gradient Algorithm.
– *Adaptative Fixing* proposé par Bertsimas et Demir [BER 02] (voir Algorithme II.3).

De plus, afin de diminuer les temps de calcul, en plus de la fixation de variables grâce aux profits réduits (Proposition II.2 page 32), ils autorisent la fixation d'un certain nombre de variables x_j avec $j \in \{k-l, k-l+1, \ ..., \ k+l\}$ avec $l = \left\lfloor \frac{k}{lag-time} \right\rfloor$ où *lag-time* est un paramètre fixé par l'utilisateur. Cette méthode, bien qu'elle a tendance à dégrader la qualité de la borne, permet d'obtenir une borne inférieure de (MKP) dans des temps de calculs raisonnables. Cette méthode est nommée *Lag variable fixing*.

Algorithme II.2 - *ADP-BH*

Initialisation :
k=n, c'=c, F=∅ (indice des variables fixées) ;
early-terminaison=false.

Appliquer BH sur (MKP(k,c')) et récupérer la solution $x^{BH}(k,c')$ *et la borne associée* $H(k,c')$ *;*
Vérifier le critère d'arrêt et mettre à jour early-terminaison ;
$x^{BEST} = x^{BH}(k,c')$ *et* $z^{BEST} = H(k,c')$.

Réduire les variables à partir des profits réduits ;
$F = F \cup \{j \in \{1,...,n\} \mid x_j \ fixé\}$ *et mettre à jour* $x^F = \{x_j = 0 \ ou \ 1 \mid j \in F\}$.

Tant que $k > 2$ *et early-terminaison=false :*
Si $k \in F$ *alors* $x_k^{ADP} = x_k^F$ *;*
Sinon
Utiliser Variable assignement pour calculer x_k^{ADP} *;*
Mettre à jour x^{BEST} *et* z^{BEST} *;*
Vérifier le critère d'arrêt et mettre à jour early-terminaison ;
Fixer les variables via la procédure Lag variable fixing et mettre à jour F et x^F *;*
$c' = c' - W^k.x_k^{ADP}$ *;*
k=k-1.

Fixer x_1^{ADP} *à la solution optimale de* $MKP(1,c')$ *;*

Retourner x^{BEST} *et* $z^{BEST} = H(k,c')$ *;*

Le critère d'arrêt utilisé dans l'algorithme II.2 permet de fixer la variable *early-terminaison* à *true* ou à *false*. L'objectif est d'arrêter la procédure avant la fin de l'itération si une solution optimale est détectée.

A une étape k de ADP-BH, on considère :
– le sous problème $MKP(k,c')$,
– $x^{BH}(k,c')$ sa solution par l'heuristique *BH*,
– $H(k,c')$ sa borne associée et
– $U(k,c')$ une borne supérieure.
Le critère d'arrêt consiste alors à vérifier :
– si $x^{BH}(k-1,c')$ et $x^{BH}(k-1,c'-W_k)$ sont,respectivement , des solutions optimales du sous-problème $(MKP(k-1,c'))$ et $(MKP(k-1,c'-W^k))$ (une solution $x^{BH}(k',c'')$ est considérée comme optimale si $H(k',c'') = U(k',c'')$),
– si $H(k-1,c') \geq U(k-1,c'-W^k) + p_k$ et $H(k-1,c') = U(k-1,c')$, alors $x^{BH}(k,c') = (x^{BH}(k-1,c'),0)$ est une solution optimale,
– de même, si $H(k-1,c'-W^k) + p_k \geq U(k-1,c')$ et $H(k-1,c'-W^k) = U(k-1,c'-W^k)$, alors $x^{BH}(k,c') = (x^{BH}(k-1,c'-W^k),1)$ est une solution optimale.
Si l'une de ces conditions est vérifiée, la variable *early-terminaison* prend la valeur *true*.

Algorithme II.3 - *Adaptative Fixing :*

Initialisation :
– $\gamma = 0,25$*, paramètre utilisateur ;*
– $X_1 = \emptyset$*, indice des variables fixées à* 1 *;*
– $X_0 = \emptyset$*, indice des variables fixées à* 0 *;*

Résoudre (LP^C) *et récupérer sa solution* x^{LP^C}*, avec, en notant* $K = \{1, ..., n\} - X_1 \cup X_0$ *:*

$$(LP^C) \begin{cases} \max \sum_{j \in X_1} p_j \ + \ \sum_{j \in K} p_j.x_j, \\ s.c. \sum_{j \in K} w_{i,j}.x_j \leq c_i - \sum_{j \in X_1} w_{i,j}, \ i \in \{1, ..., m\}, \\ x_j \in \{0,1\} \text{ pour } j \in K \ (x_j = 1 \text{ pour } j \in X_1 \text{ et } x_j = 0 \text{ pour } j \in X_0). \end{cases}$$

Mettre à jour X_1 *et* X_0 *:*
- $X_1 = X_1 \cup \{j \mid x_j^{LP^C} = 1\}$,
- $X_0 = X_0 \cup \{j \mid 0 \leq x_j^{LP^C} < \gamma\}$.

Tant que x^{LP^C} *contient des valeurs fractionnaires :*
Résoudre (LP^C) *et récupérer sa solution* x^{LP^C} *;*
$j^* = min\{j \in \{1, ..., n\} \mid 0 < x_j^{LP^C} < 1\}$ *;*
$X_1 = X_1 \cup \{j \mid x_j^{LP^C} = 1\}$ *et* $X_0 = X_0 \cup \{j \mid x_j^{LP^C} = 0\} \cup \{j^*\}$ *;*
Fin tant que.

La solution retournée par l'algorithme est alors $x^{AF} = (x_1^{AF}, ..., x_n^{AF})$ *tel que :*
- $x_j^{AF} = 1$*, pour* $j \in X_1$ *et*
- $x_j^{AF} = 0$*, pour* $j \in X_0$.

Les tests numériques menés par Bertsimas et Demir suggèrent que la dernière heuristique, Adaptative Fixing, permet d'obtenir les meilleurs résultats non seulement en terme de temps de calcul, mais aussi en ce qui concerne la qualité de la borne. Pour ces tests, le paramètre *lag-time* a été fixé à 10. C'est cette variante de L'ADP-BH que nous avons retenue.

II.4.3 SMA : un algorithme simple multiniveaux

Comme AGNES, le Simple Multistage Algorithm combine différentes approches heuristiques. Du fait de sa flexibilité, Hanafi, Fréville et El Abdellaoui [HAN 96] en ont proposé différentes variantes.

Cette heuristique commence par, selon la terminologie adoptée sur la recherche Tabou, un processus de diversification. Pour cela, un ensemble de solutions, réalisables ou non, est généré de façon aléatoire, complété par les deux solutions extrêmes, à savoir (0, 0, ..., 0) et (1, 1, ..., 1). Cet ensemble est le point de départ pour une recherche au voisinage. La flexibilité de leur heuristique réside dans le fait que plusieurs méthodes de recherche au voisinage peuvent être facilement utilisées telle qu'une méthode gloutonne ou un recuit simulé.

Algorithme II.4 - *SMA :*

Générer aléatoirement un ensemble P *contenant* S *chaînes binaires* $x^s \in \{0,1\}^n$*,* $s \in \{1, ..., S\}$ *;*
$P = P \cup \{x^0, x^{s+1}\}$ *avec* $x^0 = (0, 0, ..., 0)$ *et* $x^{s+1} = (1, 1, ..., 1)$.

$\underline{x} = x^0$ *;*
Pour s=0 à S+1
Si x^s *est non réalisable, alors Projection*(x^s) *fin Si ;*
Local_search(x^s) *;*

Repeated_drop_add(x^s) ;
Si $p^t.x^s > p^t.\underline{x}$, alors $\underline{x} = x^s$ fin Si ;
Fin Pour.

La procédure *Projection* permet de "projeter" les solutions non réalisables dans l'ensemble des solutions réalisables. Elle est basée sur une méthode gloutonne de Senju et Toyoda [SEN 68].

Algorithme II.5 - *Projection(x) :*

feasible=false ;
Calculé Δ représentant les ressources consommées par x :

$$pour\ i \in \{1, ..., m\},\ \Delta_i = \sum_{j=1}^{n} w_{i,j}.x_j - b_i;$$

Déterminer le sous-ensemble I(x) des contraintes violées par x :

$$I(x) = \{i \mid \Delta_i > 0,\ i \in \{1, ..., m\}\};$$

Tant que feasible=false :
Choisir $j^* = argmin\left\{\frac{p_j}{v_j} \mid x_j = 1,\ j \in \{1, ..., n\}\right\}$ où v_j représente un facteur de pénalité de la variable j, à savoir $v_j = \sum_{i \in I(x)} w_{i,j}.\Delta_i$;
$x_{j^*} = 0$;
Mettre à jour Δ : pour $i \in \{1, ..., m\},\ \Delta_i = \Delta_i - w_{i,j^*}$;
Mettre à jour I(x) ;
Si $I(x) = \emptyset$ alors feasible=true fin Si ;
Fin Tant que.

La procédure *Repeated_drop_add* est une procédure d'intensification. Elle essaie d'améliorer la borne obtenue par la recherche au voisinage en utilisant de façon répétée une recherche gloutonne appelée la procédure *Add*. Cette méthode utilise une relaxation surrogate pour le classement des variables, et nécessite donc le calcul d'un multiplicateur μ qui peut être :
– la solution optimale d'un dual du problème (MKP),
– obtenu par un calcul direct proposé par Fréville et Plateau [FRE 86].

Algorithme II.6 - *Add($\underline{x}$) :*

$q = |J_0(\underline{x})|$ avec $J_0(\underline{x}) = \{j \mid \underline{x}_j = 0\}$;
Classer les indices $j \in J_0(\underline{x})$ tel que :

$$\frac{p_{j1}}{\mu^t.W^{j1}} \geq ... \geq \frac{p_{jk}}{\mu^t.W^{jk}} \geq ... \geq \frac{p_{jq}}{\mu^t.W^{jq}};$$

$x=\underline{x}$;

Pour k=1 à q
Si $\forall i \in \{1, ..., m\}\ w_{j,jk} + \sum_{j=1}^{n} w_{i,j}.x_j \leq c_i$, alors $x_{jk} = 1$ fin Si ;
Fin Pour

Si $p^t.x > p^t.\underline{x}$, *alors* $\underline{x} = x$ *fin Si.*

Algorithme II.7 - *Repeated_drop_add*($\underline{x}$) :

On considère $J_1(\underline{x})$ *le sous-ensemble des indices de* $\underline{x}$ *valant 1 :*

$$J_1(\underline{x}) = \{j \mid x_j = 1,\ j \in \{1, ..., n\}\};$$

$x^{\#} = \underline{x}$;

Pour $j \in J_1(\underline{x})$
    $x = x^{\#}$; $x_j = 0$;
    Add(x) ;
    Si $p^t.x > p^t.\underline{x}$, *alors* $\underline{x} = x$ *fin Si* ;
Fin Pour.

En ce qui concerne la procédure *Local_search*, plusieurs implémentations ont été proposées par les auteurs :
– un algorithme glouton,
– un algorithme de recuit simulé,
– *Threshold Accepting* de Dueck et Scheuer ([DUE 89] et [DUE 90]),
– *Noising method* de Charon et Hudry [CHA 93a] et [CHA 93b].
Les tests numériques menés ont montré que, comparée aux autres variantes de l'algorithme, la méthode de *Threshold Accepting* permet d'avoir les meilleurs temps de calcul, mais aussi la meilleure borne. Nous avons donc implémenté cette version.

Au niveau de la formulation *Threshold Accepting* est très proche d'un algorithme de recuit simulé. *Threshold Accepting* accepte une solution candidate comme nouvelle solution temporaire si il ne détériore pas trop la valeur courante de la fonction objective. Quant au recuit simulé, les plus mauvaises solutions sont acceptées avec une probabilité relativement basse. Dueck et Scheuer ont montré que leur algorithme permet d'avoir une meilleure borne avec un temps de calcul plus faible que le recuit simulé.

Algorithme II.8 - *Local_search*($\underline{x}$) :

Initialiser les paramètres T et r ;
change=true ; $x = \underline{x}$;

Tant que change=true
    change=false ;
    Répéter r fois
        Move(x,x') ;
        Threshold_Accepting(x,x',T,change)
        Si $p^t.x > p^t.\underline{x}$, *alors* $\underline{x} = x$ *fin Si ;*
    Fin Répéter
    Mettre à jour T et r ;
Fin Tant que

La procédure *Move* permet de générer une nouvelle solution réalisable x' à partir de x. Les paramètres T et r sont fixés par l'utilisateur, r représentant la taille du voisinage exploré, et T la dégradation tolérée de la fonction objective.

Algorithme II.9 - *Threshold_Accepting(x,x',T,change) :*

Si $\delta = p^t.x' - p^t x > -T$, *alors*
    x=x';
    change=true
Fin Si;

II.4.4 Les autres heuristiques existantes

Bien entendu, il existent beaucoup d'autres heuristiques, sans parler de métaheuristiques, dans la littérature. Nous proposons ici de faire un rapide tour d'horizon de ces méthodes.

Heuristiques Primales : Les heuristiques primales débutent avec la solution nulle (sac à dos vide). Selon une règle donnée, une succession de variables candidates sont affectées à 1 tant que la solution reste réalisable (voir McCarl, Kochenberg et Wynmann [MCC 74], Toyoda [TOY 75] et Loulou et Michaelides [LOU 79]).

Heuristiques Duales : Les heuristiques duales débutent avec un vecteur solution égal à 1 (tous les objets sont pris). De même que précédemment, selon une règle donnée, une succession de variables candidates sont affectées à 0 jusqu'à l'obtention d'une solution réalisable (voir Senju et Toyoda [SEN 68]).

Heuristiques basées sur une relaxation continue : Ces méthodes essaient de construire une solution réalisable pour (MKP) à partir de l'analyse de la solution de sa relaxation continue (voir Hillier [HIL 69] et celui de Balas et Martin [BAL 80]).

Zanackis [ZAN 77] a comparé l'algorithme de McCarl, Kochenberg et Wynmann [MCC 74], Senju et Toyoda [SEN 68] et celle de Hillier [HIL 69] et a montré qu'aucun d'entre eux ne dominait les autres.

Heuristiques basées sur les autres relaxations : Ces heuristiques se basent sur une ou plusieurs relaxations du problème de départ (relaxation lagrangienne, surrogate et/ou composite). AGNES de Fréville et Plateau [FRE 94] en font parti (voir aussi Magazine et Oguz [MAG 84] et Osorio, Glover et Hammer [OSO 02]).

Métaheuristiques : De nombreuses métaheuristiques ont été testées pour la résolution de (MKP). Parmi elles nous pouvons citer :
- le recuit simulé (voir [KIR 83]), avec Drexl [DRE 88],
- la recherche tabou (voir [GLO 89], [GLO 90] et [GLO 97]), avec Hanafi et Fréville [HAN 98] et Vasquez et Hao [VAS 01a] [VAS 01b],
- l'algorithme génétique (voir [GOL 89]), avec Chu et Beasley [CHU 98].

II.5 Les méthodes de résolution exacte

II.5.1 Branch and Bound

L'algorithme du branch-and-bound [LAN 60] est basé sur une méthode arborescente de recherche d'une solution optimale par séparations et évaluations. reconnue pour résoudre les problèmes d'optimisation combinatoire NP-difficiles.

Dans la littérature, plusieurs auteurs ont présentés des algorithmes de branch-and-bound pour le problème (MKP), notamment Shih [SHI 79] suivi de Gavish et Pirkul [GAV 85] qui ont obtenu des temps calcul meilleurs que leur prédécesseur.

Le problème posé pourra être, d'une façon générale, formulé de la façon suivante :

$$\begin{cases} \max f(x), \\ s.c.\ x \in X. \end{cases} \tag{II.23}$$

On recherche un élément $x^* \in X$, une solution optimale, tel que $f(x^*) \geq f(x)$, $\forall x \in X$ où :
– f(.) est une fonction d'utilité,
– X est un ensemble fini, défini par un ensemble de contraintes sur les variables.
On s'intéresse au cas où l'énumération de l'ensemble des éléments de X est impossible en raison de l'importance de son cardinal. La méthode de branch-and-bound cherche alors à éliminer de l'espace de recherche des sous-ensembles qui ne peuvent pas fournir de solution optimale.

Présentation de l'algorithme

L'algorithme consiste à séparer de manière récursive le problème en sous-problèmes de cardinalité inférieure tant que la résolution de ces problèmes reste difficile. Le cardinal de l'ensemble à explorer est réduit en imposant à cet ensemble des contraintes supplémentaires (réduction du domaine). Une série de tests, appliquée à tous les sous-problèmes permet de supprimer de l'espace de recherche les sous-problèmes qui ne peuvent pas engendrer de solution optimale.

Cette recherche par décomposition de l'ensemble des solutions peut être représentée graphiquement par un arbre. C'est de cette représentation que vient le nom de "méthode de recherche arborescente" .
– Chaque sous-problème créé au cours de l'exploration est symbolisé par un noeud de l'arbre (ou sommet), le noeud racine représentant le problème initial.
– Les branches de l'arbre symbolisent le processus de séparation, ils représentent la relation entre les noeuds.
– Les noeuds non séparés, appelés noeuds pendants (par exemple, (S_1) , (S_3) et (S_4), de la figure II.1)
Plus précisément, le branch-and-bound est basé sur trois principes :
– le principe de séparation
– le principe d'évaluation
– la stratégie de parcours.

Principe d'évaluation :

Le principe d'évaluation permet de diminuer l'espace de recherche. L'objectif est d'essayer d'évaluer l'intérêt de l'exploration d'un sous-ensemble de l'arborescence.

Le branch-and-bound utilise, à des fins d'élagage (élimination de branches dans l'arborescence de recherche), deux fonctions :
– une borne inférieure de la fonction d'utilité du problème initial, qui résulte d'une fonction d'évaluation et
– une borne supérieure de la fonction d'utilité des solutions d'un sous-ensemble.
La connaissance d'une borne inférieure du problème et d'une borne supérieure de la fonction d'utilité de chaque sous-problème permet de stopper l'exploration d'un sous-ensemble de solu-

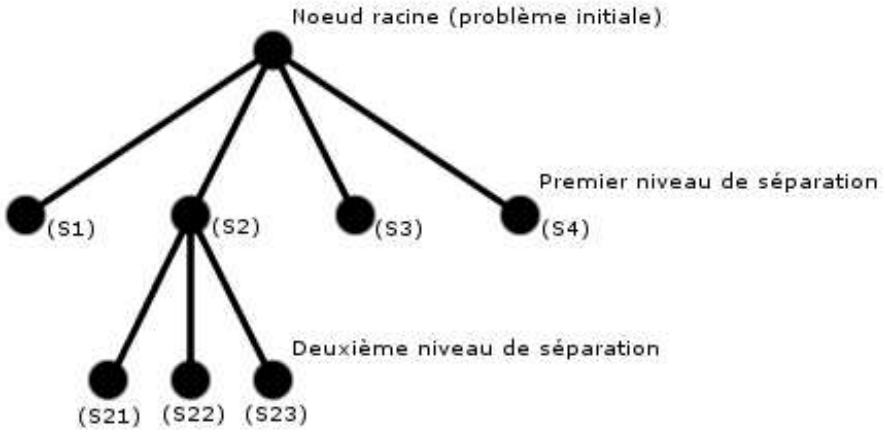


FIG. II.1 – Arbre généré par décomposition du sous-problème initial

tions ne pouvant pas contenir de solutions candidates à l'optimalité : si pour un sous-problème la borne supérieure est plus petite que la borne inférieure du problème, l'exploration du sous-ensemble correspondant est inutile. D'autre part, lorsque le sous-ensemble est suffisamment « petit », on procède à une évaluation dite exacte : on résout alors le sous-problème correspondant.

Le principe de séparation :

Le principe de séparation permet d'établir l'ensemble des règles qui vont régir la séparation d'un ensemble en sous-ensembles. Afin d'assurer l'optimalité de la solution fournie par le branch-and-bound, certaines règles doivent être respectées.

- Règle 1 : Aucune solution optimale ne doit être écartée lors d'une séparation. On utilise la règle suivante qui garantit la règle 1 :
- Règle 1' : La réunion des sous-ensembles obtenus lors d'une séparation doit être égale à l'ensemble séparé, si S_i est séparé en $S_{i,1}$, ..., $S_{i,n}$ alors $\cup_{j=1}^{n} S_{i,j} = S_i$.
- Règle 2 : Le cardinal d'un sous-ensemble doit être inférieur à celui de son père.
- Règle 3 : Un sous-ensemble qui ne peut être séparé doit être sondable.

Remarque: *Un ensemble sondable est un ensemble qu'on ne peux plus séparer parce que :*

- *on connaît la meilleure solution de l'ensemble,*
- *on connaît une solution meilleure que toutes celles de l'ensemble,*
- *on sait que l'ensemble ne contient aucune solution admissible.*

Stratégie de parcours :

La stratégie de parcours est la règle suivant laquelle est choisi le sommet devant être séparé parmi tous les sommets pendants de l'arborescence. Parmi les stratégies de parcours les plus connues, on peut citer :

- La profondeur d'abord :
  L'exploration privilégie les sous-problèmes obtenus par le plus grand nombre de séparations appliquées au problème de départ, c'est-à-dire aux sommets les plus éloignés de la racine

(de profondeur la plus élevée). L'obtention rapide d'une solution optimale et le peu de place mémoire nécessaire en sont les avantages. L'inconvénient est l'exploration de sous-problèmes peu prometteurs à l'obtention d'une solution optimale.

– La largeur d'abord :
Cette stratégie favorise les sous-problèmes obtenus par le moins de séparations du problème de départ, c'est à dire les sommets les plus proches de la racine. Il est à noter que cette stratégie est peu utilisée car elle présente une efficacité plus faible que les deux autres stratégies présentées.
– Le meilleur d'abord :
Cette stratégie favorise l'exploration des sous-problèmes possédant la plus grande borne supérieure. Elle dirige la recherche là où la probabilité de trouver une meilleure solution est la plus grande. Elle permet aussi d'éviter l'exploration de tous les sous-problèmes qui possèdent une évaluation inférieure à la valeur optimale.

Application au sac à dos multidimensionnel (MKP)

Plusieurs méthodes de calcul de borne supérieure sont abordées. Nous présentons également plusieurs fonctions heuristiques pour l'obtention d'une borne inférieure.

Rappel de la formulation du problème :

$$(MKP)\begin{cases} \max \sum_{j=1}^{n} p_j.x_j, \\ \text{s.c. } \sum_{j=1}^{n} w_{i,j}.x_j \leq c_i, \ i \in \{1,...,m\}, \\ x_j \in \{0,1\}, \ j \in \{1,...,n\}, \end{cases} \tag{II.24}$$

avec p_j, c_i, et $w_{i,j}$ entiers positifs pour $i \in \{1,...,m\}$ et $j \in \{1,...,n\}$.

Obtention des bornes supérieures :

Les méthodes permettant d'obtenir les bornes supérieures sont les méthodes de relaxation du problème. En élargissant l'espace des solutions réalisables, nous acceptons de nouvelles solutions hors de l'espace d'origine. La plus grande valeur de la fonction objective associée à ces solutions nous conduit à une borne supérieure.

On s'est donc intéressé au problème relâché suivant :

$$(LP)\begin{cases} \max \sum_{j=1}^{n} p_j.x_j, \\ \text{s.c. } \sum_{j=1}^{n} w_{i,j}.x_j \leq c_i, \ i \in \{1,...,m\}, \\ x_j \in [0,1], \ j \in \{1,...,n\}, \end{cases} \tag{II.25}$$

La méthode de résolution choisie est celle du simplexe. Afin de diminuer la complexité spatiale (occupation mémoire), nous avons opté pour une version de cet algorithme spécifique au cas où les variables sont bornées ([NEM 88]). Pour les problèmes à une seule contrainte, on utilisera l'algorithme de Martello et Toth [MAR 90] (voir page 28).

Obtention des bornes inférieures :

Une borne inférieure doit avoir une propriété importante, elle doit correspondre à une solution réalisable pour le problème initial. Il s'agit donc de construire une procédure permettant de trouver assez rapidement des solutions approchées du problème.

Plusieurs méthodes heuristiques sont proposées dans la littérature. Ces heuristiques peuvent être classées en quatre groupes :
- Les heuristiques, dites gloutonnes, qui fixent à chaque itération au moins une variable x_j en fonction de sa valeur relative $\frac{p_j}{v_j}$. Le facteur v_j caractérise chaque méthode. Deux sous-groupes peuvent être distingués selon la solution prise pour initialiser l'heuristique :
  - les méthodes se basant sur une solution réalisable mais non optimale et
  - les méthodes se basant sur la solution non réalisable $x_j = 1$ pour $j \in \{1, ..., n\}$.
- Les heuristiques utilisant l'arrondi de la solution fournie par la méthode du simplexe ou des variantes de celles-ci. La solution obtenue est à améliorer si possible par un algorithme glouton
- Les heuristiques à contraintes surrogate.
- D'autres méthodes partiellement basées sur une recherche aléatoire ou ayant des heuristiques à structures particulières.

Nous présentons ici deux heuristiques de type glouton couramment utilisées :

Algorithme II.10 - *L'algorithme Glouton :*
Les indices des variables étant classées selon :

$$\frac{p_1}{u^t.W^1} \geq \ldots \geq \frac{p_j}{u^t.W^j} \geq \ldots \geq \frac{p_n}{u^t.W^n}.$$

$u \in \mathbb{R}^m_+$ *étant un vecteur choisit par l'utilisateur.*

Initialisation :
$\underline{x} = (0, ..., 0)^t$;
$\underline{v} = 0$;
$Q = c$, *vecteur représentant la place restante dans le sac à dos ;*

Pour $j \in \{1, ..., n\}$
  Si $Q - W^j \geq 0$
    $\underline{x}_j = 1$;
    $\underline{v} = \underline{v} + p_j$;
    $Q = Q - W^j$;
  Fin Si ;
Fin Pour ;

Retourner $\underline{v}$ *et* $\underline{x}$.

u est généralement pris égal à $(1, ..., 1)^t$.

Algorithme II.11 - *Heuristique basée sur une méthode du simplexe et le Glouton (Simplexe-Glouton) :*
Soit $x^* = (x_1^*, \ldots, x_n^*)$ *une solution optimale du problème* (LP) *obtenue par une méthode du simplexe.*
Sans perdre de généralité, notons :
- $R = \{1, ..., k\}$, *l'indice des variables de* x^* *appartenant à* $]0, 1[$ *et*
- $E = \{k + 1, ..., n\}$, *l'indice des variables de* x^* *appartenant à* $\{0, 1\}$.

On applique alors l'algorithme du Glouton au problème réduit :

$$(MKP^{red}) \begin{cases} \max \sum_{j \in R} p_j.x_j, \\ s.c. \sum_{j \in R} w_{i,j}.x_j \leq c_i - \sum_{j \in E} w_{i,j}.x_j^* \text{ pour } i=\{1,...,m\}, \\ x_j \in \{0,1\} \text{ pour } j \in R. \end{cases}$$

Notons alors x^{red} *la solution de* (MKP^{red}) *obtenue.*

Une borne inférieure de (MKP) *est donnée par :* $\underline{x} = \begin{pmatrix} x^{red} \\ x_{k+1}^* \\ \vdots \\ x_n^* \end{pmatrix}$ *et* $\underline{v}(MKP) = p^t.x^{red}$.

II.5.2 La Programmation Dynamique

La programmation dynamique [BEL 54] [BEL 57] est une autre méthode classique de résolution exacte qui peut-être utilisée pour la résolution du problème du sac à dos multidimensionnel en variables 0-1. C'est une méthode d'optimisation faisant intervenir des décisions par étapes.

Le formalisme :

Pour l'étape k, avec $k \in \{1, \ldots, n\}$ (n est le nombre d'étapes)
- X_k est l'ensemble des « décisions » que l'on peut prendre à l'étape k,
- E_k est l'ensemble des « états » dans lesquels le système peut se trouver à l'étape k,
- $C(e_{k-1}, x_k)$ est le profit de la décision x_k qui fait passer le système de l'état e_{k-1} à l'état e_k,
- $f(k, e_k) = \sum_{j=1}^{k} C(e_{j-1}, x_j)$ est le profit total de la séquence de décisions $x = (x_1, x_2, \ldots, x_k)$ qui fait passer le système de l'état e_0 à e_k,
- $F(e_{k-1}, x_k) = e_k$: transition d'état.

La programmation dynamique repose sur les principes décrits ci-après.

Principe 1 : propriété Markovienne

Le profit associé à la décision x_k lorsqu'on se trouve à l'état e_{k-1} ne dépend que de ces deux éléments et non de toutes les décisions antérieures qui ont permis d'aboutir à l'état e_{k-1}.

Principe 2 :

On cherche une évolution du système de profit global (minimum ou maximum) égal à la somme des profits de transition.

Principe 3 : principe d'optimalité

Soit une trajectoire optimale partant du point A et arrivant au point C, la portion de cette trajectoire partant de n'importe quel point intermédiaire B, appartenant à cette trajectoire, et arrivant au point C doit être la trajectoire optimale entre les points B et C.

Conséquences :

Quelle que soit la décision optimale prise à l'étape k qui amène le système de l'état e_{k-1} à l'état e_k, la portion de la politique entre e_0 et e_{k-1} doit être optimale.

La résolution du problème revient donc à trouver une séquence optimale $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$ qui à partir de l'état initial e_0 nous amène à l'état e_n tout en maximisant la fonction profit :

$$f(n, e_n) = max \left\{ \sum_{j=1}^{n} C(e_{j-1}, x_j) \,|\, x_j \in X_j;\, e_j = F(e_{j-1}, X_j);\, j \in \{1, \ldots, n\} \right\} \tag{II.26}$$

En appliquant le principe d'optimalité, nous pouvons calculer de proche en proche $f(n, e_n)$ en utilisant l'équation suivante :

$$f(k, e_k) = \max_{x_k \in X_k; e_k = F(e_{k-1}, X_k)} \{C(e_{k-1}, x_k) + f(k-1, e_{k-1})\} \tag{II.27}$$

où $f(k, e_k)$ représente le profit optimal du problème sur un horizon de k périodes.

La programmation dynamique pour le problème du sac à dos (une dimension)

Nous allons nous intéresser à l'application de la programmation dynamique au problème à une seule contrainte. En voici la formulation :

$$(UKP) \begin{cases} \max \sum_{j=1}^{n} p_j.x_j, \\ \text{s.c.} \sum_{j=1}^{n} w_j.x_j \leq c, \\ x_j \in \{0,1\},\ j \in \{1, ..., n\}, \end{cases} \tag{II.28}$$

avec p_j, c, et w_j entiers positifs pour $j \in \{1, ..., n\}$.

On définit des familles de problèmes pour chaque couple (k, g) , $k \in \{1, \ldots, n\}$ et $g \in \{0, \ldots, c\}$:

$$(UKP(k,g)) \begin{cases} \max \sum_{j=1}^{k} p_j.x_j, \\ \text{s.c.} \sum_{j=1}^{k} w_j.x_j \leq g, \\ x_j \in \{0,1\},\ j \in \{1, ..., n\}. \end{cases} \tag{II.29}$$

La solution du problème initial est donnée par la solution de $(UKP(n\ , c))$. Le profit obtenu en chargeant le sac à dos de capacité g avec les k premiers objets est $f(k, g)$.

Dans ce problème on a :
– l'ensemble des « décisions » que l'on peut prendre à l'étape k :

$$X_k = \{0, 1\}, \tag{II.30}$$

– l'ensemble des « états » dans lesquels le système peut se trouver à l'étape k :

$$E_k = \left\{ e_k = g \,|\, g \in \left\{ 0, 1, \ldots, min \left\{ c, \sum_{j=1}^{k} w_j \right\} \right\} \right\}, \tag{II.31}$$

– le profit de la décision x_k qui fait passer le système de l'état e_{k-1} à l'état e_k :

$$C(e_{k-1}, x_k) = p_k.x_k, \tag{II.32}$$

– le profit total de la séquence de décisions $x = (x_1, \ldots, x_k)$ qui fait passer le système de l'état e_0 à e_k :

$$\sum_{j=1}^{k} C(e_{j-1}, x_j) = \sum_{j=1}^{k} p_j.x_j, \tag{II.33}$$

– la transition d'état :

$$F(e_{k-1}, x_j) = e_k = e_{k-1} + w_k.x_k. \tag{II.34}$$

L'ensemble,E_k , des états dans lesquels le système peut se trouver à l'étape k, associés chacun à la séquence de décisions optimales $x_g = (x_{g,1}, \ldots, x_{g,k})$ qui maximise le profit pour le poids g correspondant constitue la liste de la programmation dynamique à l'étape k. On a alors pour chaque $g \in \{1, \ldots, c\}$:

$$f(k, g) = \sum_{j=1}^{k} C(e_{k-1}, x_{g,j}) \tag{II.35}$$

Le principe d'optimalité nous permet ensuite de déduire la relation qui relie l'ensemble des éléments de la liste de la programmation dynamique à l'étape k aux éléments de la liste à l'étape $k-1$. Cette relation nous permet de construire itérativement la liste contenant l'ensemble des solutions des problèmes $f(k, g)$ pour un k fixé :

$$f(k, g) = \begin{cases} f(k-1, g) \ pour \ g \in \{0, ..., w_k\}, \\ \max\{f(k-1, g),\ c_k + f(k-1, g - w_k)\} \ pour \ g \in \{0, ..., \underline{c}\}, \end{cases} \tag{II.36}$$

avec $\underline{c} = \min \left\{ c, \sum_{j=1}^{k} w_j \right\}$.

Le concept de listes

Pour chaque entier positif k, on définit la liste récursive de programmation dynamique L_k comme l'ensemble suivant de couples :

$$L_k = \left\{ (w, p) \mid w = \sum_{j=1}^{k} w_j.x_j \leq c,\ p = \sum_{j=1}^{k} p_j.x_j,\ avec \ \forall j \in \{1, .., k\}\ x_j \in \{0, 1\} \right\}. \tag{II.37}$$

Exemple :

L'exemple suivant illustre un problème à $n = 5$ objets $[(8,9),\ (7,3),\ (6,2),\ (5,3),\ (4,1)]$ avec une capacité $c = 20$:

$$(Ex)\begin{cases} \max\ 9.x_1 + 3.x_2 + 2.x_3 + 3.x_4 + 1.x_5, \\ s.c.\ 8.x_1 + 7.x_2 + 6.x_3 + 5.x_4 + 4.x_5 \leq 20, \\ x_j \in \{0,\ 1\}\ pour\ j \in \{1, ..., 5\}. \end{cases} \tag{II.38}$$

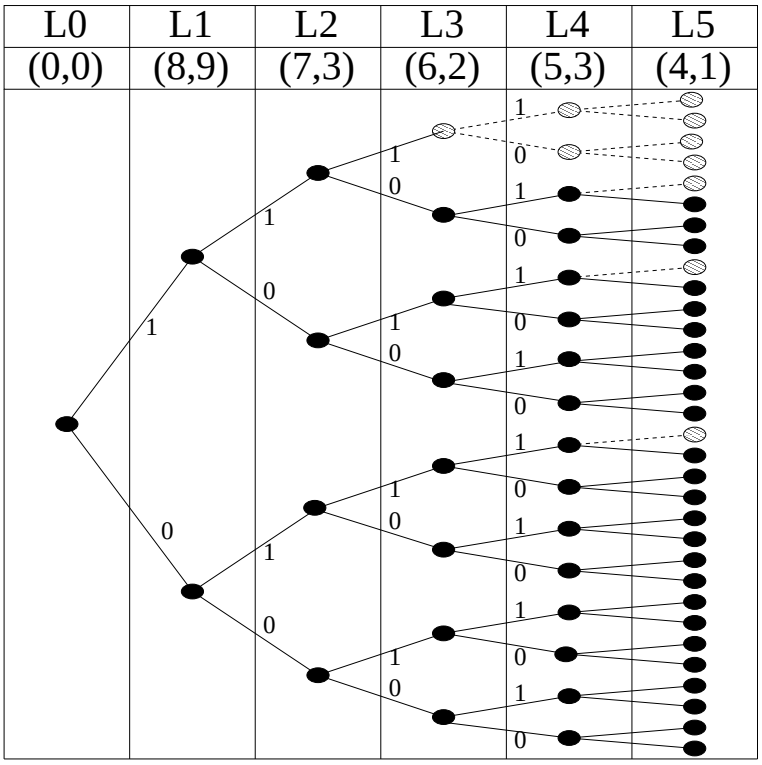


FIG. II.2 – Représentation des listes de la programmation dynamique.

Comme on peut le voir sur la figure II.2, qui représente la construction des listes de la programmation dynamique, nous avons une structure d'arbre binaire. Les états représentés par un point rayé correspondent à des états non-réalisables pour le problème (Ex) (équation II.38).

Les listes correspondent chacune à un niveau de l'arbre, et l'élaboration de ces listes équivaut à une descente en largeur. Cette méthode de parcours est indispensable pour l'utilisation du principe de dominance décrit ci-après.

Le principe de la dominance

Proposition II.6 - *Le principe de la dominance :*
Soit un couple ***(w,p)****.*
Si $\exists$ ***(w',p')*** *tel que* $\mathbf{p} \geq \mathbf{p}'$ *et* $\mathbf{w} \leq \mathbf{w}'$*, alors on dit que le couple* ***(w,p)*** *domine le couple* ***(w',p')****.*

On retire par conséquent de la liste de la programmation dynamique les éléments dominés. Le

principe de dominance découle en fait directement du principe d'optimalité. La figure II.3 montre la suppression des éléments dominés dans l'arbre de recherche.

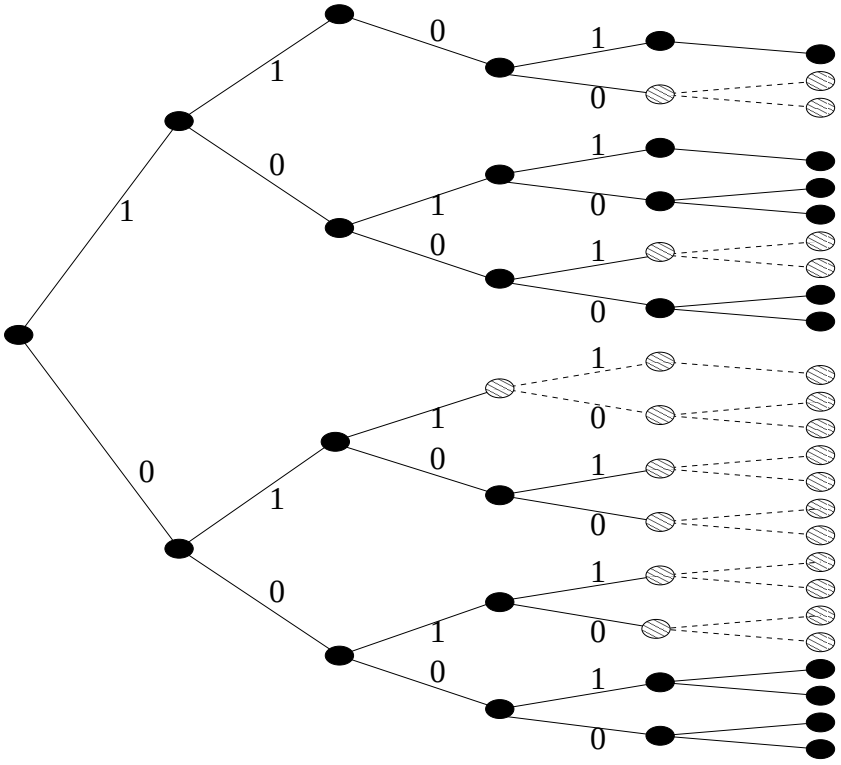


FIG. II.3 – Application du principe de dominance.

Dans l'exemple le couple (13,5) est dominé par le couple (8,9) car il a un poids supérieur mais un profit inférieur. Tous les autres couples en pointillés dans l'arbre sont ceux qui ont été supprimés par dominance.

On voit que ce concept permet de supprimer un grand nombre de couples, réduisant ainsi considérablement le travail de recherche d'une solution optimale.

L'utilisation de la programmation dynamique avec le concept de liste et celui de la dominance nous permet d'avoir une complexité temporelle égale à $O(min(2^n, n.c))$ (Ahrens et Finke [AHR 75], Horowitz et Sahni [HOR 74], Toth [TOT 80]) et Plateau et Elkihel [ELK 84a].

Décomposition du problème

Afin de réduire la complexité temporelle de la programmation dynamique, Horowitz et Sahni [HOR 74] ont proposé une méthode basée sur la décomposition de (UKP) en deux sous-problèmes (UKP_1) et (UKP_2), de $n_1 = \lfloor n/2 \rfloor$ et de $n_2 = n - n_1$ variables :

$$(UKP_1)\begin{cases}\max \sum_{j=1}^{n_1} p_j.x_j, \\ \text{s.c.} \sum_{j=1}^{n_1} w_j.x_j \leq c, \\ x_j \in \{0,1\},\ j \in \{1,...,n_1\},\end{cases} \quad et \quad (UKP_2)\begin{cases}\max \sum_{j=n_1+1}^{n} p_j.x_j, \\ \text{s.c.} \sum_{j=n_1+1}^{n} w_j.x_j \leq c, \\ x_j \in \{0,1\},\ j \in \{n_1+1,...,n\}.\end{cases} \tag{II.39}$$

Ces deux sous-problèmes sont résolus via la programmation dynamique avec le principe de dominance. Nous obtenons alors deux listes contenant l'ensemble des états non dominés, à la $n_1^{ème}$ étape pour (UKP_1) et à la $n_2^{ème}$ étape pour (UKP_2), que nous noterons $\mathcal{L}_1$ et $\mathcal{L}_2$.

Il faut alors rechercher parmi l'ensemble des combinaisons réalisables possibles entre les états des deux listes celle qui maximise le profit. Afin d'éviter l'énumération de toutes ces combinaisons, la liste $\mathcal{L}_1$ est parcourue selon les poids décroissants et $\mathcal{L}_1$ selon les poids croissants. Ainsi en un seul parcours des deux listes nous vérifions l'ensemble des combinaisons réalisables. Cette étape est appelée la fusion des listes.

Pour cette recherche on se sert d'un pivot P qui sera alternativement dans $\mathcal{L}_1$ puis dans $\mathcal{L}_2$. $P = (w, p)$ étant dans $\mathcal{L}_1$, on parcours $\mathcal{L}_2$ jusqu'à (w', p') tel que $w + w' \geq c$. Le nouveau pivot est alors $P = (w', p')$. On continue alors le parcours de la liste $\mathcal{L}_1$ à partir de (w', p') jusqu'à (w'', p'') tel que $w' + w'' \leq c$ et on prend $P = (w'', p'')$ et ainsi de suite. la recherche s'arrête lorsque la fin d'une des deux listes est atteinte.

Cette méthode permet de diminuer la complexité temporelle et la complexité spatiale de l'algorithme de programmation dynamique. En effet, la complexité temporelle de construction des deux listes est de $O(min(2^{\frac{n}{2}}, \frac{n}{2}.c))$, et celle de la fusion des deux listes est de $O(min(2^{\frac{n}{2}}, c))$. En ce qui concerne la complexité spatiale, elle passe de $O(min(2^n, c))$ à $O(min(2^{\frac{n}{2}}, c))$.

Exemple sur la décomposition du problème

$$(Ex)\begin{cases}\max\ 2.x_1 + 4.x_2 + 7.x_3 + 10.x_4, \\ s.c.\ 3.x_1 + 5.x_2 + 8.x_3 + 10.x_4 \leq 15, \\ x_j \in \{0,\ 1\}\ pour\ j \in \{1,...,4\}.\end{cases} \tag{II.40}$$

En appliquant le principe de la décomposition, on construit les deux sous-problèmes suivant :

$$(Ex_1)\begin{cases}\max\ 2.x_1 + 4.x_2, \\ s.c.\ 3.x_1 + 5.x_2 \leq 15, \\ x_j \in \{0,\ 1\}\ pour\ j \in \{1,2\}.\end{cases} \quad et \quad (Ex_2)\begin{cases}\max\ 7.x_3 + 10.x_4, \\ s.c.\ 8.x_3 + 10.x_4 \leq 15, \\ x_j \in \{0,\ 1\}\ pour\ j \in \{3,4\}.\end{cases} \tag{II.41}$$

Dans le tableau II.1, nous avons représenté la génération des listes sans décomposition du problème Ex. Le tableau II.2 présente la génération des listes $\mathcal{L}_1$ et $\mathcal{L}_2$ correspondant respectivement au problème (Ex_1) et (Ex_2). Dans les deux tableaux, tous les états dominés ont été éliminés. La valeur de la variable de décision x est représentée par une conversion binaire en décimal (Ex : $x = (1,1,0)^t \Leftrightarrow x = 1.2^0 + 1.2^1 + 0.2^2 = 2$) pour une meilleure lisibilité.

On peut alors appliquer l'algorithme de fusion :
– Itération 1 : $P = (8,6,3) \in \mathcal{L}_1$ et on s'arrête sur l'état $(8,7,4) \in \mathcal{L}_2 \Rightarrow\ \underline{v}(Ex) = 6 + 0 = 6$,

– Itération 2 : $P = (8, 7, 4) \in \mathcal{L}_2$ et on s'arrête sur l'état $(5, 4, 2) \in \mathcal{L}_1 \Rightarrow \underline{v}(Ex) = 7 + 4 = 13$,
– Itération 3 : $P = (5, 4, 2) \in \mathcal{L}_1$ et on s'arrête sur l'état $(10, 10, 8) \in \mathcal{L}_2 \Rightarrow \underline{v}(Ex) = 4+10 = 14$,
– Stop : on a atteint la fin d'une des deux listes.
La valeur optimale du problème est alors $v(Ex) = 14$, avec :

$$x = 2 + 8 = 10 = 2^1 + 2^3 \Leftrightarrow x = (0, 1, 0, 1)^t.$$

De plus, on peut remarquer que la méthode de décomposition a nécessité la construction de 11 états contre 19 pour la méthode classique.

k=1			k=2			k=3			k=4		
w	p	x	w	p	x	w	p	x	w	p	x
									15	14	10
						13	11	6	13	12	9
						11	9	5	10	10	8
			8	6	3	8	7	4	8	7	4
			5	4	2	5	4	2	5	4	2
3	2	1	3	2	1	3	2	1	3	2	1
0	0	0	0	0	0	0	0	0	0	0	0

TAB. II.1 – Génération des listes sans décomposition du problème.

$\mathcal{L}_1$

k=1			k=2		
w	p	x	w	p	x
			8	6	3
			5	4	2
3	2	1	3	2	1
0	0	0	0	0	0

$\mathcal{L}_2$

k=4			k=3		
w	p	x	w	p	x
10	10	8			
8	7	4	8	7	4
0	0	0	0	0	0

TAB. II.2 – Génération des listes par décomposition du problème.

La programmation dynamique pour le MKP :

Nous avons ici appliqué le principe de la programmation dynamique au problème du sac à dos à une seule contrainte. Ce principe peut être appliqué au problème à m contraintes de façon très similaire : les capacités ne sont alors plus des entiers mais des vecteurs d'entiers de dimension m.

La programmation dynamique est efficace pour résoudre le problème à une seule contrainte : la complexité spatiale de l'algorithme est linéaire par rapport à la capacité du sac c. De plus, la possibilité d'exploiter le principe de dominance accélère encore les traitements. Lors du passage à m contraintes, la complexité spatiale est de l'ordre de $O(c_1.c_2 \dots c_m)$. L'application du principe de dominance se trouve limité elle aussi par le fait qu'un élément n'est dominé par un autre que si chacun des poids du premier vecteur est dominé par celui qui lui correspond dans le second.

De la même manière, on pourrait appliquer le principe de décomposition décrit précédemment. Cependant du fait de la multitude de contraintes, qui rendait par ailleurs le principe de dominance moins efficace, nous ne pouvons ranger les éléments d'une liste comme pour le cas à une contrainte. De ce fait la procédure de fusion des listes n'a pas la même complexité temporelle qui sera ici de l'ordre de $O(min(2^{\frac{n}{2}}, c_1.c_2 \dots c_m))$.

Tout ceci participe au fait que le problème du sac à dos multidimensionnel est un problème plus difficile à résoudre par la programmation dynamique que son homologue à une seule contrainte.

II.5.3 Les solveurs existants

Il existe de nombreux solveurs qui ont été développés pour la résolution du problème du sac à dos. Dans cette section nous allons présenter les principaux d'entre eux. Nous distinguerons les logiciels libres mis à disposition de la communauté notamment par le biais d'internet et les logiciels payants.

Les logiciels libres

Une des plus ferventes communautés du logiciel libre regroupée sous le Projet GNU propose un ensemble d'outils libre pour l'optimisation de programmes linéaires. Ils sont regroupés sous le nom de GLPK (GNU linear programming kit) et est disponible sur le site internet :

$$http://www.gnu.org/software/glpk/glpk.html.$$

GLPK se présente sous forme de librairie pouvant être utilisée par des programmes en langage C/C++. Il intègre les composants suivants :
– la méthode du simplexe révisé,
– la méthode du point intérieure primal-dual,
– la méthode Branch-and-bound,
– un solveur de programme linéaire et programme mixte en nombres entiers.
GLPK parait donc comme un outil complet, surtout pour la résolution de programme linéaire, cependant il montre ses faiblesses sur la résolution de problèmes en nombres entiers car il n'intègre pas d'outils évolués comme par exemple les méthodes de coupes.

Bernard Le Cun et François Galea du laboratoire PRISM de l'Université de Versailles ont développé et continuent de faire évoluer un outil appelé Bob++. Bob++ est une bibliothèque programmée en langage C++ d'aide au développement d'applications de type Séparation et Évaluation. Il permet de créer un environnement permettant de faciliter la programmation de méthodes comme la branch-and-bound ou la programmation dynamique.

Bob++ est disponible via le site internet :

$$http://www.prism.uvsq.fr/\ blec/BOBO/main.html.$$

Bob++ comprend, entre autres, les composants suivants :
– différents algorithmes de recherche,
– différentes méthodes heuristiques,
– une gestion automatique des différents paramètres,
– une gestion des erreurs,

– une gestion transparente du parallélisme.
Bob++ est en constante évolution afin d'intégrer les dernières avancées de la communauté scientifique et son utilisation est aisée dans le cadre de développement d'algorithme.

Les logiciels commerciaux

Les logiciels commerciaux présentent souvent des performances supérieures aux solveurs libres. Ils disposent de plus de mises à jour régulières et d'une assistance en cas de problèmes. Ce sont des outils aboutis et complets présentant une bonne flexibilité (intégration dans différents langages de programmation sous la forme de librairie et une interface utilisateur plus conviviale).

Les deux principaux logiciels existants sont :
– CPLEX de ILOG :

$$http://www.ilog.com/products/cplex$$

– XPRESS-MP de Dash Optimisation :

$$http://www.dashoptimization.com/home/products/products\_optimizer.html$$

Ces logiciels étant payants, les méthodes utilisées ici ne sont pas accessibles. Les algorithmes utilisés sont à priori robustes et de traiter une grande variété de problèmes différents. Cependant dans le cas du MKP, ils ne peuvent résoudre des instances de taille 1000×2.

II.6 Les méthodes coopératives

Les méthodes coopératives font parties des nouvelles pistes de recherches pour le développement d'un algorithme de résolution efficace du problème du sac à dos multidimensionnel. Elles visent à faire coopérer différentes méthodes de résolution existantes afin d'en construire une plus robuste.

En effet on constate que selon le problème à résoudre les différentes méthodes de résolution exacte donnent des temps de calcul différents. Il n'existe pour le moment aucun algorithme coopératif efficace pour la résolution de (MKP). Cependant, dans le cadre de la résolution de problèmes uni-contraints, on peut citer :
– Martello et Toth [MAR 82],
– Elkihel [ELK 84a] et Elkihel et Plateau [ELK 84b] [PLA 85],
– Viader [VIA 98] et Elkihel, Viader et Authié [ELK 02]
Martello et Toth et Plateau et Elkihel ont proposé une méthode coopérative le Subset-Sum Problem :

$$(SSP)\begin{cases} \max \sum_{j=1}^{n} w_j.x_j, \\ \text{s.c. } \sum_{j=1}^{n} w_j.x_j \leq c, \\ x_j \in \{0,1\},\ j \in \{1,...,n\}, \end{cases} \tag{II.42}$$

Plateau et Elkihel [ELK 84a] [ELK 84b] ont proposé aussi une méthode coopérative pour le problème du sac à dos à contrainte générale (UKP), cette méthode fut reprise et améliorée par Viader [VIA 98] [ELK 02] :

$$(UKP)\begin{cases} \max \sum_{j=1}^{n} p_j.x_j, \\ \text{s.c.} \sum_{j=1}^{n} w_j.x_j \leq c, \\ x_j \in \{0,1\},\ j \in \{1,...,n\}, \end{cases} \tag{II.43}$$

Toutes ces méthodes essaient de faire coopérer efficacement la programmation dynamique et le branch-and-bound. Dans cette section nous allons présenter brièvement une méthode coopérative pour le problème (SSP) (Martello et Toth [MAR 82]) qui fut une des premières méthodes coopératives, puis nous décrirons une méthode coopérative à trois phases pour le problème (UKP) (Elkihel, Authié et Viader [VIA 98] [ELK 02]) avec laquelle les auteurs ont obtenu de bons résultats.

II.6.1 Méthode coopérative pour le problème (SSP)

La méthode de Martello et Toth a été construite pour la résolution du problème (SSP). Les deux principales étapes sont :
– phase 1 : application la programmation dynamique sur une partie des variables du problème,
– phase 2 : Application le branch-and-bound sur le reste des variables.
Soit les entiers p et q et le réél c', tel que :

$$p < q < n \ et \ c' < c.$$

Phase 1

On construit alors via la programmation dynamique les deux listes suivantes :
– $\mathcal{L}_1 = \left\{ w \mid w = \sum_{j=q}^{k} w_j.x_j \leq c \ avec \ \forall j \in \{q,...,k\} \ x_j \in \{0,1\} \ et \ k \in \{q,...,n\} \right\}$: la liste des combinaisons réalisables des $n-q+1$ dernières variables,
– $\mathcal{L}_2 = \left\{ w \mid w = \sum_{j=p}^{k} w_j.x_j \leq c' \ avec \ \forall j \in \{p,...,k\} \ x_j \in \{0,1\} \ et \ k \in \{p,...,n\} \right\}$: la liste des combinaisons réalisables des $n-p+1$ dernières variables avec une capacité c'.
Les paramètres p et q et c' sont fixés en fonction de la difficulté du problème à résoudre. Les auteurs suggèrent pour les problèmes ayant beaucoup de solutions satisfaisantes l'égalité $\sum_{j=1}^{n} w_j.x_j = c$, il vaut mieux favoriser l'énumération explicite de la phase 2 en prenant p et q grand et c' petit, et réciproquement dans le cas contraire.

Phase 2

On applique une énumération explicite sur les q premières variables. Soit w un état issu de cette énumération partielle.

On complète alors w par w' tel que $w + w' \leq c$, en prenant pour w' la valeur :
– $w'_1 = \max\{w' \in \mathcal{L}_1 \mid w + w' \leq c\}$ ou

– $w'_2 = \max\{w' \in \mathcal{L}_2 \mid w + w' \leq c \text{ et } w' \neq w\}$.
Soit $\delta = \max\{w'_1, w'_2\}$. Si $w + \delta = c$ alors le problème (SSP) est résolu, et dans le cas contraire la valeur optimale est donnée par le majorant de l'ensemble des combinaisons possibles.

On constate que les performances de cette méthode dépend des paramètres choisis. La méthode d'Elkihel pour résoudre le problème (SSP) ne nécessite aucun paramètre, la phase programmation dynamique n'est appliquée que sur le noyau du problème (voir [BAL 80]).

II.6.2 Méthode coopérative à trois phases pour le problème (UKP)

Lorsque l'on résout une instance donnée par la programmation dynamique et puis par le branch-and-bound, on se rend compte que les temps de calcul pour la première méthode sont proches les unes des autres, ce qui n'est pas le cas pour la deuxième. Elkihel, Authié et Viader [VIA 98] [ELK 02] se sont intéressés à ce problème et ont essayé de pallier aux défauts de chacune d'entre elle.

On s'intéresse ici à la résolution de (UKP). Soit j^* l'indice de base définit par Dantzig (voir page 28). Les profits réduits sont alors définis comme suit :

$$Pour\ j \in \{1, ..., n\},\ d_j = p_i - p_{j^*}.\frac{w_j}{w_{j^*}}. \tag{II.44}$$

On considère alors les variables classés comme suit :

$$|d_1| \leq |d_2| \leq ... \leq |d_n|.$$

Balas et Zemel [BAL 80] ont constaté que l'explosion combinatoire pour la méthode de branch-and-bound avec la stratégie meilleure d'abord se faisait sur les 25 dernières variables du problème. Ces variables font partie de ce que l'on appelle le noyau du problème. Ces variables sont donc traitées au préalable par la programmation dynamique.

Phase 1

Soit $p \in \{1, ..., n\}$. La phase 1 traite les p dernières variables par la programmation dynamique en utilisant le principe de dominance (voir proposition II.6 page 52). Une nouvelle heuristique pour le calcul des bornes basées sur la connaissance des trajectoires optimales de la programmation dynamique est utilisée.

En notant $\mathcal{P}$ l'indice des variables déjà traitées par la programmation dynamique, $\mathcal{B} = \{1, ..., n\} - \mathcal{P}$ et $\overline{x} = \{\overline{x}_1, ..., \overline{x}_n\}$ une solution optimale de la relaxation continue du problème de départ, cette heuristique consiste à résoudre les deux problèmes suivants :

$$(B_1)\begin{cases} \displaystyle\sum_{j\in\mathcal{B}} p_j.\overline{x}_j + \max \sum_{j\in\mathcal{P}} p_j.x_j, \\ \text{s.c.} \displaystyle\sum_{j\in\mathcal{P}} w_j.x_j \leq c - \sum_{j\in\mathcal{B}} w_j.\overline{x}_j, \\ x_j \in \{0,1\},\ j \in \mathcal{P}, \end{cases} \quad et \quad (B_2)\begin{cases} \max \displaystyle\sum_{j\in\mathcal{P}} p_j.x_j, \\ \text{s.c.} \displaystyle\sum_{j\in\mathcal{P}} w_j.x_j \leq c, \\ x_j \in \{0,1\},\ j \in \mathcal{P}, \end{cases} \tag{II.45}$$

La résolution de ces problèmes utilise la liste construite par la programmation dynamique, il suffit de récupérer l'état correspondant au profit le plus grand et satisfaisant les contraintes de

(B_1), d'une part, et de (B_2) d'autre part. (B_1) essaie de construire une solution optimale en se basant sur la solution du problème relâché, alors que (B_2) récupère l'état ayant le meilleur profit.

Une troisième borne est calculée via un algorithme glouton sur $\mathcal{B}$. Au final, le meilleur de ces trois bornes est retenue pour le calcul de la borne inférieure.

Il en résulte de cette première phase :
– une liste d'état $\mathcal{L}$ dont la trajectoire optimale sur les p dernières variables est connue,
– une borne inférieure $\underline{v}$ pour (UKP).

Phase 2

L'étape branch-and-bound est initialisée avec la liste $\mathcal{L}$ et la borne $\underline{v}$. Le traitement des p dernières variables par la phase 1 réduit la profondeur maximale de l'arbre de recherche.

L'enjeu de cette méthode est la détermination du paramètre p qui répartit la charge de travail entre les deux phases. En se basant sur une analyse de complexité et le temps de calcul théorique de chacune des phases, les auteurs suggèrent de prendre $p = 2.log_2 n$.

Phase 3

Avec la méthode coopérative construite sur les deux précédentes phases, les auteurs montrent, à travers les tests numériques,une amélioration du temps de calcul de deux à trois fois, en moyenne, par rapport au branch-and-bound. Cependant, dans certain cas, la programmation dynamique donne de meilleurs temps de calcul que le branch-and-bound.

Il est donc quelque fois préférable d'arrêter le branch-and-bound dans la phase 2 afin de revenir sur une résolution par la programmation dynamique. Pour cela, les auteurs proposent de fixer à un nombre q le nombre maximal de séparation acceptée dans le branch-and-bound. Au delà de ce nombre de séparation, il est préférable d'utiliser la programmation dynamique.

En fixant $q = \frac{c}{n}$, les temps de calcul obtenus sont quasiment égaux à la programmation dynamique dans les cas défavorables au branch-and-bound. De plus cette dernière phase ne détériore pas les résultats obtenus précédemment.

II.7 Conclusion

Nous avons présenté dans ce chapitre un état de l'art des différents travaux portant sur le problème du sac à dos multidimensionnel. Nous avons abordé les principales méthodes :
– de relaxation, avec en particulier la relaxation surrogate qui nous servira par la suite,
– de réduction (fixation de variables, réduction de contrainte et du domaine continu),
– de résolution approchée, avec en particulier les trois heuristiques AGNES, SMA et ADP,
– de résolution exacte, à savoir le branch-and-bound et la programmation dynamique, ainsi que les méthodes coopératives appliquées au sac à dos à une contrainte.

le problème du sac à dos multidimensionnel a fait l'objet de nombreuses études aussi bien sur l'aspect réduction que résolution, cependant il n'existe pour le moment aucun algorithme de complexité polynomiale pour le résoudre. Les bons résultats obtenus avec les méthodes coopératives

dans le cas à une contrainte motivent leur extension au cas à plusieurs contraintes.

Dans le chapitre suivant nous proposons des méthodes de résolution approchées ou exactes combinant différentes techniques de résolution.

Chapitre III

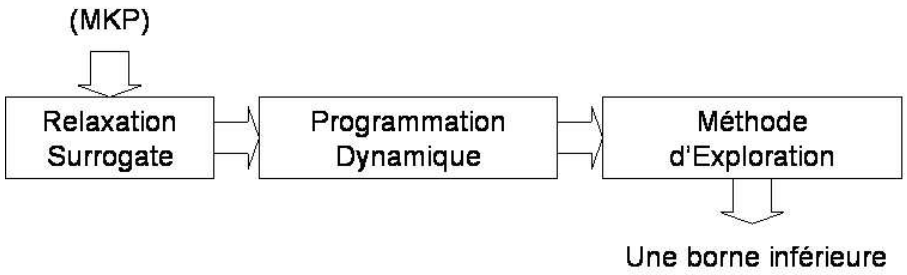
Une méthode coopérative multi-étape

III.1 Introduction

Le point de départ de ce travail était l'étude d'une méthode coopérative pour la résolution du problème du sac à dos multidimensionnel. L'idée était de résoudre une relaxation surrogate du problème (MKP) afin d'obtenir un problème à une seule contrainte pour, d'une part, avoir des temps de résolution plus faibles et, d'autre part, pouvoir utiliser une résolution par un algorithme de programmation dynamique.

Le principe de dominance dans la programmation dynamique permet d'écarter des états durant l'exploration, de manière définitive si on se place du point de vue du problème relaxé, ou de manière temporaire du point de vue du problème original. Ces états peuvent alors être explorés via d'autres méthodes de résolution telles que le branch-and-bound. Cette approche coopérative permet de faire coopérer différentes méthodes de résolution et permet de construire une méthode de résolution exacte pour le problème (MKP). On peut aussi dégager de cette trame une heuristique. En effet, comme on va le voir, la première phase de notre algorithme qui consiste à résoudre une relaxation surrogate via la programmation dynamique, nous fournit non seulement une liste d'états, mais aussi une borne inférieure pour le problème (MKP). Cette phase peut donc aisément être interprétée comme une heuristique laissant le choix à l'utilisateur d'aller plus loin dans la résolution. Cette phase peut être aussi interprétée comme un processus de diversification, qui est, comme on va le voir, une méthode très utilisée dans les heuristiques. Elle nous fournit une famille de solutions réalisables qui pourront être explorées ou bien par une méthode de résolution exacte, comme proposé ci-dessus, ou bien par une méthode de recherche au voisinage afin de construire une nouvelle heuristique permettant d'essayer d'améliorer la dernière borne obtenue.

On peut schématiser les différentes étapes des méthodes proposées :



Le choix de la méthode d'exploration à la dernière étape déterminera le caractère exact ou approché de la méthode de résolution.

Dans ce chapitre sont présentés les principales étapes des algorithmes étudiés pour la construction d'une méthode de résolution exacte et de méthodes de résolutions approchées.

Les tests numériques présentés ont été faits sur des instances de la littérature dont les valeurs optimales sont connues :
– instances de Petersen [PET 67],
– instances de Weingartner et Ness [WEI 67],
– instances de Shi [SHI 79],
– instances de Senju et Toyoda [SEN 68].
Cette série de problèmes de taille relativement petite nous ont permis d'avoir une première appréciation sur le comportement des algorithmes proposés. Des tests plus poussés seront présentés à la fin de ce mémoire.

III.2 La relaxation surrogate

Cette première étape, qui est à la base des algorithmes étudiés ici, a pour objectif de transformer notre problème (MKP) en un problème possédant qu'une seule contrainte. Nous avons utilisé pour cela la relaxation surrogate :

$$(MKP)\begin{cases}\max p^t.x,\\ \text{s.c. } W.x \leq c,\\ x \in \{0,\ 1\}^n.\end{cases} \Rightarrow \quad (S(\mu))\begin{cases}\max p^t.x,\\ \text{s.c. } \mu^t.W.x \leq \mu^t.c,\\ x \in \{0,\ 1\}^n.\end{cases} \tag{III.1}$$

On rappel que le multiplicateur μ optimale est définit comme suit :

$$\min_{\mu \geq 0}\{v(S(\mu))\} = v(S). \tag{III.2}$$

Résoudre (III.2) de façon exacte est en lui-même un problème NP-difficile. De plus, on n'a aucune garantie que $v(S) = v(MKP)$. Nous nous sommes donc tournés vers des méthodes de calcul approchées du multiplicateur μ. Dans la suite nous décrirons les méthodes étudiées et nous essaierons d'évaluer leurs performances.

III.2.1 Les heuristiques étudiées

Différentes heuristiques ont été proposées pour le calcul du multiplicateur de la relaxation surrogate. Nous nous sommes intéressés tout d'abord à l'heuristique de Garvish et Pirkul [GAV 85] qui semble être une des heuristiques les plus abouties mais aussi à d'autre méthode plus simple à mettre en place ayant l'avantage d'avoir des temps de résolution plus court.

Afin d'évaluer ces différentes heuristiques, nous nous sommes intéressés à la valeur optimale du problème surrogate résultant sur des instances de la littérature.

La méthode de Garvish et Pirkul [GAV 85]

La méthode de Garvish et Pirkul se base sur les travaux de Glover [GLO 75] qui a proposé une procédure pour le calcul optimal des multiplicateurs pour les problèmes à 2 contraintes. Les multiplicateurs seront estimés de manière itérative en considérant une succession de problèmes à 2 contraintes. Deux procédures sont décrites :
– Procédure-1 : permet une recherche approchée des multiplicateurs pour un problème à 2 contraintes,
– Procédure-2 : permet une recherche approchée des multiplicateurs pour un problème à m contraintes de manière itérative par l'utilisation de Procedure-1.
Nous allons dans un premier temps considérer le problème suivant :

$$S(\psi,\ \mu) = Max\{cx \mid (\psi\,\mu).(W.x - C) \leq 0,\ x \in \{0,\ 1\}^n\}, \tag{III.3}$$

avec ψ et μ des scalaires et W une matrice de dimension $2 \times n$.

Algorithme III.1 - *Procédure-1*

Étape 1 : Soit μ_H et μ_L tel que la solution de $S(1,\mu)$ satisfait la première contrainte pour $\mu = \mu_L$ et ne la satisfait pas pour $\mu = \mu_H$.
Étape 2 : Si $\mu_H - \mu_L \leq \epsilon$ STOP.
Étape 3 : Posons $\mu = \mu_L + (\mu_H + \mu_L)/2$ et résoudre $S(1,\mu)$. Si la solution obtenue :
- *satisfait les 2 contraintes, STOP ($(1,\ \mu)$ est un multiplicateur optimal).*
- *ne satisfait que la première contrainte, on pose $\mu_L = \mu$ et retour à l'étape 2.*
- *ne satisfait pas la première contrainte, on pose $\mu_H = \mu$ et retour à l'étape 2.*

Afin de diminuer les temps de calcul de cette procédure Garvish et Pirkul proposent de remplacer la résolution du problème $S(1,\ \mu)$ par la résolution de sa relaxation continue. Cette procédure est appelé Procédure-1A.

Algorithme III.2 - *Procédure-2 : Calcul des multiplicateurs pour un (MKP) quelconque.*
Étape 1 : On cherche la contrainte qui fournit la plus petite valeur de la fonction objective lorsque les autres contraintes sont ignorées. On renumérote cette contrainte comme la contrainte 1.
Étape 2 : On cherche alors la contrainte la plus violée par la solution du problème formé par la fonction objective originale et la contrainte 1. Elle devient alors la contrainte 2.
Étape 3 : On applique Procédure 1 avec les contraintes 1 et 2. Si la fonction objective ne décroît pas, ou si le nombre d'itérations devient trop grand, STOP.
Étape 4 : La contrainte surrogate déterminée à l'étape 3 devient alors la contrainte 1, et on retourne à l'étape 2.

Les résultats numériques montrent que Procédure-2 combinée avec Procédure-1 est la méthode fournissant le meilleur compromis entre temps de calcul et qualité de la borne. Cependant, en comparaison avec la méthode décrite ci-après, ils obtiennent des temps de calculs importants pour une amélioration d'environ 4% de la borne.

Dualité en continu

Revenons sur la relaxation continue en contrainte égalité, (LP), de (MKP) définit page 31 (équation (II.10)), nous avons la dualité suivante :

$$(LP)\begin{cases} \max \tilde{p}^t.\tilde{x}, \\ \text{s.c. } \tilde{W}.\tilde{x} = \tilde{c}, \\ \tilde{x} \geq 0, \end{cases} \quad \Leftrightarrow \quad (LP_{DUAL})\begin{cases} \min \lambda^t.\tilde{c}, \\ \text{s.c. } \lambda^t.\tilde{W} \geq \tilde{p^t}, \\ \lambda \geq 0, \end{cases} \tag{III.4}$$

Nous avons ici équivalence entre (LP) et (LP_{DUAL}), c'est-à-dire que nous avons égalité des valeurs optimales. La résolution de (LP_{DUAL}) peut se faire par l'algorithme dual du simplexe, et nous fournit une solution λ optimale que l'on notera $\lambda^* = (\lambda_1^*, ..., \lambda_{m+n}^*)^t$.

De ce λ^*, nous ne conservons que les m premières composantes correspondant aux m contraintes du problème (MKP), afin de construire le multiplicateur de la relaxation surrogate comme suit :

$$\mu_{DUAL} = (\lambda_1^*, ..., \lambda_m^*)^t.$$

Le calcul de ce multiplicateur se résume donc à une résolution du problème dual de (LP).

En notant $(S_{LP}(\mu_{DUAL}))$, la relaxation continue de $(S(\mu_{DUAL}))$, on a l'égalité suivante :

$$v(LP) = v(S_{LP}(\mu_{DUAL})). \tag{III.5}$$

Démonstration III.1 - $v(MKP) = v(S_{LP}(\mu_{DUAL}))$
Posons $\lambda = \begin{pmatrix} \mu \\ \psi \end{pmatrix}$, *avec* $\mu = (\mu_1, ..., \mu_n)^t \in \mathbb{R}^m$ *et* $\psi = (\psi_1, ..., \psi_n)^t \in \mathbb{R}^n$. *Nous pouvons alors récrire* (LP_{DUAL}) *de la façon suivante :*

$$(LP_{DUAL}) \begin{cases} \min \mu^t.c + \psi_1 + ... + \psi_n, \\ s.c.\ \mu^t.W + \psi^t \geq p, \\ \mu \geq 0 \ et \ \psi \geq 0, \end{cases}$$

Par définition, à l'optimum nous avons $\mu = \mu_{DUAL}$ *et* $\psi = (\lambda^*_{m+1}, ..., \lambda^*_{m+n})^t$ *et*

$$v(LP) = \mu^t_{DUAL}.b + \lambda^*_{m+1} + ... + \lambda^*_{m+n}.$$

Notons alors x^*, *une solution optimale de* $S_{LP}(\mu_{DUAL})$. *Le respect des contraintes entraînent :*

$$\begin{cases} \mu^t_{DUAL}.b \geq \mu^t_{DUAL}.W.x^* \ et \\ \mu^t_{DUAL}.W + (\lambda^*_{m+1}, ..., \lambda^*_{m+n}) \geq p. \end{cases}$$

D'où :

$$\begin{aligned} v(LP) &= \mu^t_{DUAL}.b + \lambda^*_{m+1} + ... + \lambda^*_{m+n}, \\ &\geq (\mu^t_{DUAL}.W.x^* + (\lambda^*_{m+1}, ..., \lambda^*_{m+n}).x^*) - (\lambda^*_{m+1}, ..., \lambda^*_{m+n}).x^* + \lambda^*_{m+1} + ... + \lambda^*_{m+n}, \\ &\geq p^t.x^* + \lambda^*_{m+1}.(1 - x^*_1) + ... + \lambda^*_{m+n}.(1 - x^*_n), \\ &\geq v(S_{LP}(\mu_{DUAL})) \ car \ \forall i \in \{1, ..., n\},\ 1 - x^*_1 \geq 0. \end{aligned}$$

Donc

$$v(LP) \geq v(S_{LP}(\mu_{DUAL})).$$

Or $(S_{LP}(\mu_{DUAL}))$ *étant une relaxation surrogate de* (LP),

$$v(S_{LP}(\mu_{DUAL})) \geq v(LP).$$

D'où le résultat :

$$v(LP) = v(S_{LP}(\mu_{DUAL})).$$

A partir de ce résultat, on peut dire que μ_{DUAL} est le multiplicateur optimale pour le problème (LP), de plus l'inégalité suivante devient alors triviale :

$$v(MKP) \leq v(S(\mu_{DUAL})) \leq v(LP) = v(S_{LP}(\mu_{DUAL})). \tag{III.6}$$

Cette inégalité justifie l'utilisation d'un tel multiplicateur.

Autres heuristiques étudiées

Nous nous sommes aussi intéressé à d'autres méthodes de calcul approché du multiplicateur. A part la méthode M_Unite présenté ci-dessous, les différentes heuristiques proposées ici essayent

de caractériser une contrainte qui semble prépondérante afin de ne conserver qu'elle ou d'évaluer les multiplicateurs en fonction de cette caractérisation.

La méthode structurale : Les multiplicateurs sont calculés ici en tenant compte de l'importance relative de chacune des contraintes :

$$\forall\ i \in \{i, ..., m\},\ \mu_i = \frac{\sum_{j=1}^{n} w_{i,j} - c_i}{\sum_{j=1}^{n} w_{i,j}}. \tag{III.7}$$

La méthode M_Unité :

La méthode M_Unité est la plus simple à mettre en place car elle nécessite aucun calcul. Elle consiste tout simplement à ajouter les contraintes entre elles. Nous avons donc :

$$\forall\ i \in \{1, ..., m\},\ \mu_i = 1. \tag{III.8}$$

La méthode M_Profit :

Au lieu de conserver toutes les contraintes, on n'en conserve qu'une. La manière la plus logique consiste à conserver celle qui minimise le profit lorsque les autres contraintes sont ignorées. Cependant, au lieu de résoudre m problème du sac à dos uni-contraints, nous avons estimé une borne supérieure en considérant leur relaxation continue afin de diminuer les temps de calcul.

On considère donc le problème suivant :

$$Pour\ \ i \in \{1, ..., m\},\ (B_i) \begin{cases} \max p^t.x, \\ \text{s.c. } W_i.x \leq c_i, \\ x \in \{0,\ 1\}^n. \end{cases} \tag{III.9}$$

On note :

$$k = argmin_{i \in \{1,...,m\}}\{\overline{v}(B_i)\},\ avec\ \overline{v}(B_i)\ une\ borne\ supérieure\ de\ (B_i). \tag{III.10}$$

D'où :

$$\forall\ i \in \{1, ..., m\},\ i \neq k,\ \mu_i = 0\ et\ \mu_k = 1. \tag{III.11}$$

La méthode M_Glouton :

On reprend ici la même idée que précédemment sauf que nous allons conserver la contrainte qui minimise le nombre de variables fixées par une heuristique de type gloutonne. Pour cela nous résolvons les m problèmes (B_i), pour $i \in \{1, ..., m\}$ par l'algorithme présenté page 48.

En notant k l'indice du problème pour lequel le nombre de variables fixées par l'algorithme est minimal, nous construisons les multiplicateurs comme décrit par la formule précédente (III.11).

Représentation graphique sur un exemple :

On considère les deux problèmes à 2 variables et 2 contraintes suivant :

$$(Ex\_1)\begin{cases} \max 3.x_1 + 4.x_2, \\ \text{s.c.} \begin{cases} 8.x_1 + 12.x_2 \leq 15, \\ 4.x_1 + 24.x_2 \leq 21, \end{cases} \\ x_1 \in \{0,\ 1\}\ et\ x_2 \in \{0,\ 1\}. \end{cases} \tag{III.12}$$

$$(Ex\_2)\begin{cases} \max 3.x_1 + 4.x_2, \\ \text{s.c.} \begin{cases} 5.x_1 + 4.x_2 \leq 4, \\ 3.x_1 + 6.x_2 \leq 3, \end{cases} \\ x_1 \in \{0,\ 1\}\ et\ x_2 \in \{0,\ 1\}. \end{cases} \tag{III.13}$$

Bien entendu, ces problèmes possèdent des solutions triviales qui sont :
– $x_1 = 1$ et $x_2 = 0$ avec $v(Ex_1) = 3$ pour (Ex_1),
– $x_1 = 0$ et $x_2 = 0$ avec $v(Ex_2) = 0$ pour (Ex_2),
En effet ces problèmes ne satisfont pas l'hypothèses (h1) (voir page 16), cependant ils permettent d'illustrer les différentes contraintes surrogate calculées.

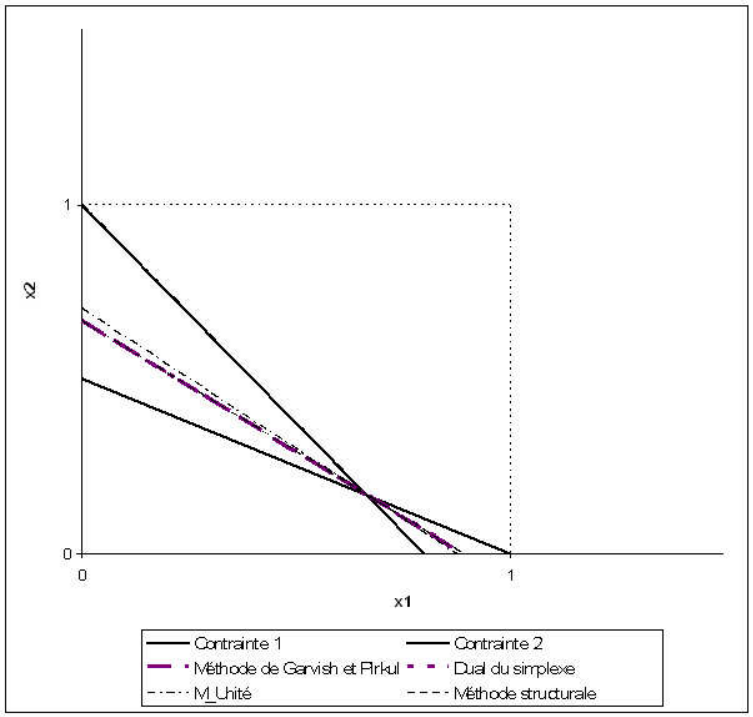


FIG. III.1 – Représentation graphique des différentes contraintes surrogate pour (Ex_1)

Sur les figures III.1 et III.2, nous avons représenté graphiquement les contraintes surrogate générées par les différentes méthodes ainsi que celles du problème original (Contrainte 1 et

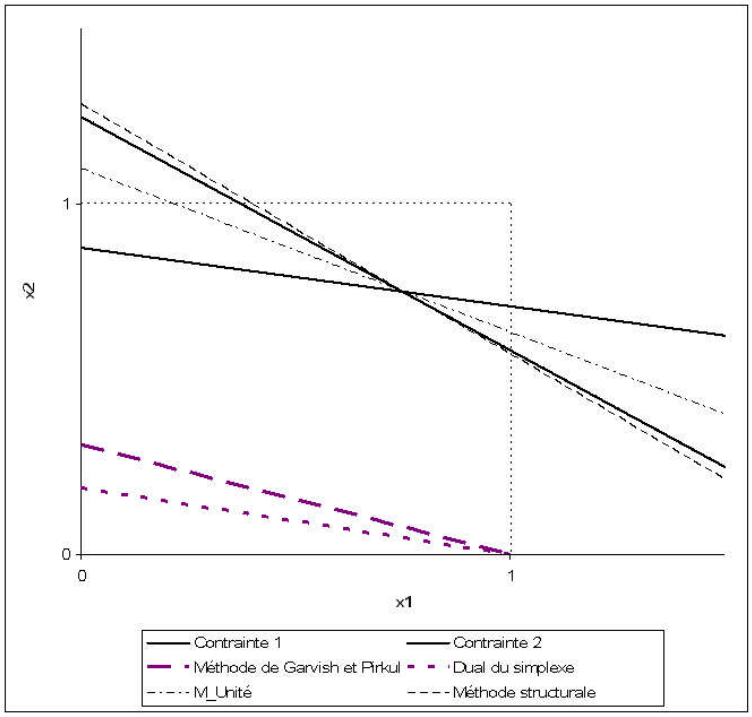


FIG. III.2 – Représentation graphique des différentes contraintes surrogate pour (Ex_2)

Contrainte 2). Nous n'avons pas fait figurer les méthodes M_Profit et M_Glouton car elles se confondent avec une des contraintes du problème original.

La figure III.1, nous montre qu'avec le problème (Ex_1) les contraintes fournies sont toutes plus ou moins équivalentes. Par contre avec la figure III.2, on peut voir que les contraintes fournies pour (Ex_2) par la méthode de Garvish et Pirkul et celle fournie par le dual du problème continue réduisent de façon importante le domaine.

Des tests numériques plus poussés sont présentés dans la suite.

III.2.2 Résultats numériques

Pour éprouver ces différents multiplicateurs, nous avons résolu de façon exacte avec CPLEX 9.0 la relaxation surrogate des instances de la littérature et nous avons comparé les bornes obtenues avec les valeurs optimales. Nous nous sommes aussi intéressés aux nombres moyens de contraintes non-respectées du problème (MKP) par la solution optimale de la relaxation surrogate. On notera les différentes méthodes présentées de la manière suivante :
- M_GP : la méthode de Garvish et Pirkul,
- M_Simp : la méthode du simplexe,
- M_Struc : la méthode structurale,
- M_Unite,
- M_Profit,
- M_Glouton.

Exemples détaillés sur un problème de Petersen

Nous présentons ici de façon détaillé les résultats obtenus sur le premier problème de l'instance de Petersen que nous noterons $(P1)$ et dont le profit optimal vaut 3800. Après une réduction de contraintes via les principes énoncés dans la partie II.3.2, page 32, nous obtenons :

$$(P1)\begin{cases} \max \begin{pmatrix} 1200 & 2000 & 100 & 600 & 500 & 2400 \end{pmatrix}.x, \\ \text{s.c.} \begin{pmatrix} 13 & 41 & 8 & 12 & 22 & 64 \\ 13 & 41 & 8 & 12 & 22 & 75 \\ 4 & 4 & 3 & 6 & 6 & 18 \\ 8 & 12 & 5 & 10 & 6 & 32 \\ 8 & 20 & 5 & 13 & 6 & 42 \\ 8 & 20 & 5 & 13 & 6 & 48 \\ 4 & 4 & 3 & 2 & 5 & 5 \end{pmatrix}.x \leq \begin{pmatrix} 80 \\ 96 \\ 20 \\ 36 \\ 44 \\ 48 \\ 18 \end{pmatrix} \\ x \in \{0,\ 1\}^6. \end{cases} \tag{III.14}$$

On note $v(S\_P1)$ la borne optimal de la relaxation surrogate de $(P1)$. Les différentes méthodes présentées précédemment nous donnent :

– M_GP : $v(S\_P1) = 3800$

$$8,15.x_1 + 20,36.x_2 + 5,09.x_3 + 13,16.x_4 + 6,20.x_5 + 42,71.x_6 \leq 44,87$$

– M_Simp : $v(S\_P1) = 3800$

$$467,04.x_1 + 1271,486.x_2 + 290,37.x_3 + 647,41.x_4 + 500.x_5 + 2400.x_6 \leq 2672,59$$

– M_Struc : $v(S\_P1) = 3900$

$$27,59.x_1 + 68,52.x_2 + 17,49.x_3 + 33,51.x_4 + 34,16.x_5 + 138,72.x_6 \leq 162,87$$

– M_Unité : $v(S\_P1) = 4300$

$$58.x_1 + 142.x_2 + 37.x_3 + 68.x_4 + 73.x_5 + 284.x_6 \leq 342$$

– M_Profit : $v(S\_P1) = 3800$

$$8.x_1 + 20.x_2 + 5.x_3 + 13.x_4 + 6.x_5 + 42.x_6 \leq 44$$

– M_Glouton : $v(S\_P1) = 3800$

$$8.x_1 + 20.x_2 + 5.x_3 + 13.x_4 + 6.x_5 + 42.x_6 \leq 44$$

On constate ici que la contrainte de M_GP se rapproche de la cinquième contrainte de $(P1)$, qui est par ailleurs la contrainte retenue par M_Profit et M_Glouton.

Parmi ces différentes approches on constate ici que seul M_Struc et M_Unité ne donnent pas le profit optimal de $(P1)$. Cependant on constate en pratique que bien que le profit optimal de la relaxation soit le même que celui du problème original, la solution x associée ne vérifie pas forcément l'ensemble de ses contraintes.

Nom de l'instance	Ecart moyen à l'optimum (%)					
	M_GP	M_Simp	M_Struc	M_Unite	M_Profit	M_Glouton
Petersen	2.68	1.25	6.13	7.94	2.22	6.47
Fréville & Plateau	18.20	3.48	19.59	21.81	25.31	34.23
Senyu et Toyoda	5.34	0.72	5.84	6.12	5.79	9.84
Weingartner & Ness	0.57	0.58	6.54	7.80	2.11	4.25
Shi	0.82	0.35	11.53	15.51	0.74	17.89

TAB. III.1 – Relaxation surrogate : Ecart moyen à l'optimum.

Nom de l'instance	Nombre moyen de contraintes non-respectées (%)					
	M_GP	M_Simp	M_Struc	M_Unite	M_Profit	M_Glouton
Petersen	30.00	20.00	32.86	34.29	25.71	44.29
Fréville & Plateau	67.92	41.25	48.33	51.46	76.25	75.83
Senyu et Toyoda	85.00	15.00	38.33	43.33	91.67	93.33
Weingartner & Ness	31.25	37.50	50.00	50.00	37.50	37.50
Shi	30.67	16.67	40.00	51.33	23.33	59.33

TAB. III.2 – Relaxation surrogate : Nombre moyen de contraintes non-respectées.

Résultats numériques sur des instances de la littérature

Afin d'apprécier les temps de calcul, présentés dans le tableau III.3, de chacunes de ces méthodes, les tests ont été menés sur une Sun Blade 100 sous système Solaris (500 MHz).

Nom de l'instance	Temps de calcul du multiplicateur (ms)					
	M_GP	M_Simp	M_Struc	M_Unite	M_Profit	M_Glouton
Petersen	7.14	2.86	0.10	1.42	4.28	1.43
Fréville & Plateau	12.50	2.50	6.25	7.50	7.50	5.00
Senyu et Toyoda	55.00	10.00	10.00	20.00	25.00	15.00
Weingartner & Ness	3.75	2.5	3.75	1.25	1.25	1.25
Shi	9.66	6.00	4.66	2.66	7.00	1.33

TAB. III.3 – Relaxation surrogate : Temps de calcul du multiplicateur.

A travers les tableaux III.1 et III.2 on se rend compte que la méthode M_Simp permet d'avoir des bornes proches de l'optimum et d'obtenir des solutions qui respectent le plus grand nombre de contraintes du problème (MKP). Les temps de calcul mis en jeux ici sont très courts, ils figurent dans le tableau III.3. On peut voir que les temps de calcul sont plus importants pour la méthode M_GP suivi de M_Profit, puis de M_Simp.

Dans la mesure où nous avons obtenu les meilleurs résultats avec la méthode M_Simp avec des temps de calcul relativement peu élevés, c'est cette méthode que nous avons retenue pour le calcul des multiplicateurs. Ce multiplicateur nous servira dans les algorithmes présentés dans la suite.

Amélioration des multiplicateurs

Nous avons essayer d'améliorer les multiplicateurs obtenus précédemment via deux méthodes :
– la rotation de contrainte de Elkihel [ELK 84a] et
– l'algorithme G de Fréville et Plateau [FRE 94]
L'algorithme G, proposé par Fréville et Plateau, qui est un algorithme sous-gradient, essaie d'améliorer de manière itérative les multiplicateurs surrogate.

Algorithme III.3 - *Algorithme G :*
Soit μ un multiplicateur quelconque tel que $\mu \neq 0$.
Faire

Résoudre $(S(\mu))$ pour obtenir une solution optimale x^*.
Si $W.x^* - c \leq 0$, alors x^* est optimal pour (MKP), STOP.
Calculer $g_\mu = W.x^* - c$.
Calculer les nouveaux multiplicateurs, avec un pas t, tel que :

$$\mu = \mu + t.\frac{g_\mu}{\|g_\mu\|^2}.$$

Projeter μ dans $\{\mu \mid \mu \in \mathbb{R}_+^m\}$:

$$pour\ i \in \{1, ..., m\},\ \mu_i = max\{0, \mu_i\}.$$

Jusqu'au critère d'arrêt.

Soit $\underline{\mu}$, μ et $\overline{\mu}$, trois multiplicateurs consécutifs générés par l'algorithme G. Les résultats numériques de [FRE 91] suggèrent de prendre le pas de calcul tel que :

$$t = |\underline{\mu}.g_\mu|,\ ainsi\ \overline{\mu} = \mu + t.\frac{g_\mu}{\|g_\mu\|^2}.$$

Le critère d'arrêt retenu est celui du nombre d'itérations sans diminution de la fonction objective.

Avec la rotation de contrainte, l'objectif est de modifier directement la contrainte surrogate afin de diminuer le domaine des solutions admissibles. La fonction objective ne sera donc pas modifiée.

Quelques tests numériques ont été fait afin de voir concrètement l'amélioration apportée à ces contraintes. Pour l'algorithme G, on s'est essentiellement intéressé à la diminution de la fonction objective, ainsi qu'au temps de calcul mis en jeu. En ce qui concerne la rotation de contrainte, sachant que la fonction objective ne changera pas, on s'est focalisé dans un premier temps sur le fait qu'il y a effectivement rotation ou pas. Ces tests ont été aussi menés sur une station Sun Blade 100 sous système Solaris.

Le tableau III.4 nous montre que l'algorithme G engendre des temps de calculs relativement importants pour une amélioration minime de la borne. Le temps de calcul étant un facteur important dans la construction d'une heuristique, nous n'avons pas retenu cette méthode.

Nom de l'instance	Temps de calcul (ms)		Ecart moyen à l'optimum (%)	
	sans G	avec G	sans G	avec G
Petersen	2,86	25,71	1,25	1,25
Fréville & Plateau	2,50	45,00	3,48	3,38
Senyu et Toyoda	10,00	45,00	0,72	0,72
Weingartner & Ness	2,50	13,75	0,58	0,50
Shi	6,00	14,66	0,35	0,34

Tab. III.4 – Relaxation surrogate : Amélioration par l'algorithme G.

Nom de l'instance	Rotation de la contrainte ?
Petersen	NON
Fréville & Plateau	NON
Senju & Toyoda	NON
Weingartner & Ness	NON
Shi	NON

Tab. III.5 – Relaxation surrogate : Amélioration par rotation de contraintes.

De même, dans le tableau III.5, parmi les tests menés, aucune rotation de la contrainte surrogate n'a été observée. Une rotation de contrainte avec ajout d'une contrainte additionnelle (ajout d'une contrainte du problème non-relaxé) engendrerait des temps de calcul trop grands. Nous n'avons donc pas retenu cette méthode aussi.

Le calcul du multiplicateur surrogate se fera donc simplement via la méthode M_Simp qui permet d'avoir le meilleur compromis entre temps de calcul et qualité de la borne obtenue.

III.3 L'heuristique HDP

Cette première étape de notre algorithme de recherche d'une solution exacte essaie d'utiliser le principe de dominance de la programmation dynamique. Ce principe peut-être appliqué directement sur les problèmes multi-contraints mais cela ne permettrait pas d'éliminer suffisamment de noeuds. C'est pourquoi, nous utilisons une technique de relaxation, la relaxation surrogate, qui permet de transformer le (MKP) en un problème uni contraint.

Du fait de la relaxation, la valeur optimale du problème relaxé majore celle du problème original. Il est donc nécessaire d'adapter la résolution par la programmation dynamique afin d'obtenir une borne inférieure pour (MKP). Pour cela, il suffit de vérifier à chaque étape de recherche que l'on vérifie toutes les contraintes du problème original.

De plus, on pourra assimiler cette première phase de programmation dynamique à une heuristique.

Lors de cette phase, nous allons écarter des noeuds qui ne vont pas conduire à une solution optimale au sens du problème relaxé, mais qui le peuvent au sens du (MKP) original. Ces noeuds seront donc conservés dans une liste secondaire, qui sera créée tout au long de cette première étape.

Nous allons tout d'abord présenter un algorithme de programmation dynamique pour les problè-

mes uni-contraints qui s'inspire des travaux de Plateau et Elkihel [ELK 84a]. Nous présenterons ensuite son utilisation dans le cadre de l'algorithme hybride suivi d'une méthode alternative. Enfin, quelques tests numériques seront présentés afin de voir les performances de ces approches.

III.3.1 Un algorithme de programmation dynamique amélioré pour les problèmes uni-contraints

A la section II.3.1, page 30, a été présenté une méthode de réduction. Reprenons les mêmes notations, appliquées à un problème uni-contraint (UKP) :

A chaque variable x_j $(j \in \{1,\ldots,n\})$, on associe trois bornes, à savoir :
– $\underline{v}$ une borne inférieure du problème (UKP), et $\underline{x}^t = (\underline{x}_1,\ldots,\underline{x}_n)$, la solution associée,
– $\overline{v}(x_j = 1)$, une borne supérieure de (UKP) lorsque x_j est fixée à 1,
– $\overline{v}(x_j = 0)$, une borne supérieure de (UKP) lorsque x_j est fixée à 0.
Notons :

$$Pour\ j \in \{1,\ldots,n\},\ \overline{v}_j = \min\{\overline{v}(x_j = 1),\ \overline{v}(x_j = 0)\}. \tag{III.15}$$

Le principe de réduction II.3.1 peut alors s'énoncer simplement :

$$Pour\ j \in \{1,\ldots,n\},\ si\ \overline{v}_j = \overline{v}(x_j = \delta) \leq \underline{v}\ et\ \delta \in \{0,1\},\ alors\ la\ variable\ x_j\ peut-\hat{e}tre\ fix\acute{e}e\ d\acute{e}finitivement\ \grave{a}\ 1-\delta.$$

La vitesse de convergence et le temps de résolution de (UKP) par la programmation dynamique est notamment lié au classement de variable adopté. D'après la proposition II.3.1, les variables susceptibles d'être fixées sont celles présentant une borne $\overline{v}_j$, $j \in \{1,\ldots,n\}$, la plus petite. Il n'est donc pas très intéressant de traiter ces variables en premier par la programmation dynamique car avec l'amélioration de la borne, elle ont une forte probabilité d'être fixé par le processus de réduction.

Par conséquent, en classant les variables selon les $\overline{v}_j$, $j \in \{1,\ldots,n\}$, décroissants, nous avons le schéma de résolution suivant :

$$\underset{\text{Programmation dynamique}}{\overset{\overline{v}_1 \geq \overline{v}_2 \geq \ldots}{\longrightarrow}} \Big\| \underset{\text{Réduction de variables}}{\overset{\ldots \geq \overline{v}_n}{\longleftarrow}}$$

On va alors s'arrêter lorsque l'ensemble des variables non traitées est vide. Ceci permet de s'arrêter plus tôt dans la recherche d'une solution optimale par la programmation dynamique.

III.3.2 Résolution du problème surrogate par la programmation dynamique hybride

On se base sur l'algorithme de programmation dynamique proposé chapitre II.5.2, intégrant le principe de dominance, pour la résolution de la relaxation surrogate de (MKP). Cependant

parmi les états explorés nous ne conserverons que ceux qui sont réalisables pour le problème (MKP).

Par conséquent, lors de l'exploration, un état (w, p) peut-être éliminé si il se trouve dans un des cas suivant :
– il n'est pas réalisable pour le (MKP),
– il est dominé,
– sa borne supérieure, que nous noterons $\overline{v}_{(w,p)}$, est plus petite que la meilleure borne inférieure connue.

Concernant le dernier cas, deux possibilités s'offrent à nous car nous pouvons considérer la meilleure borne inférieure connue :
– du problème (MKP), dans ce cas les états éliminés par leur borne supérieure ne peuvent mener à une meilleure solution,
– du problème (UKP), et dans ce cas nous n'avons aucune garantie sur le caractère optimal des états éliminés par leur borne supérieure.

A ceci s'ajoutent les différentes possibilités de classement de variables qui viennent s'ajouter à celui présenté précédemment :
– selon les profits,
– selon les profits réduits,
– selon les bornes $(\overline{v}_j)_{j\in\{1,\dots,n\}}$,
– selon le rapport des profits sur les poids,
– ... .

Algorithme générique

Cet algorithme est basé sur la résolution d'une relaxation surrogate $S(\mu)$ du (MKP). Par soucis de simplification, on adoptera les notations suivantes :
– Pour $j \in \{1, ...n\}$, $w_j = \sum_{i=1}^{m} \mu_j.w_{i,j}$ et
– $c = \sum_{m}^{i=1} \mu_j.c_j$

Ce qui donne :

$$(S(\mu)) \begin{cases} \max p^t.x, \\ \text{s.c.} \sum_{j=1}^{n} w_j.x_j \leq c, \\ x \in \{0,\ 1\}^n. \end{cases} \tag{III.16}$$

Soit l'étape k de la programmation dynamique, $k \in \{1, ..., n\}$.

La liste $\mathcal{L}_{k+1}$ est construite de façon récursive par l'algorithme de programmation dynamique (voir [ELB 05a] pour plus de détails et des exemples). A l'étape $k+1$ la liste $\mathcal{L}_{k+1}$ est générée comme suit :

Algorithme III.4 - *Génération des listes*

L'ensemble des états à l'étape $k+1$ *est donné par :*

$$\begin{aligned} \mathcal{L}'_{k+1} &= \mathcal{L}_k \oplus (w_{k+1}, p_{k+1}), \\ &= \{(w + w_{k+1}, p + p_{k+1}) \mid (w, p) \in \mathcal{L}_k \text{ and } w + w_{k+1} \leq c\}. \end{aligned}$$

Et nous avons :

$$\mathcal{L}_{k+1} = \mathcal{L}_k \cup \mathcal{L}'_{k+1} - \mathcal{D}_{k+1},$$

où $\mathcal{D}_{k+1}$ représente l'ensemble des états dominés à l'étape $k+1$:

$$\mathcal{D}_{k+1} = \{(w,p) \mid (w,p) \in \mathcal{L}_k \cup \mathcal{L}'_{k+1} \text{ et } \exists (w',p') \in \mathcal{L}_k \cup \mathcal{L}'_{k+1} \text{ avec } w' \leq w,\ p \leq p' \text{ et } (w',p') \neq (w,p)\}.$$

Initiallement, nous avons $\mathcal{L}_0 = \{(0,0)\}$. De plus à chaque états (w,p), nous associons deux bornes :
– une borne supérieure, $\overline{v}_{(w,p)}$, et
– une borne inférieure, $\underline{v}_{(w,p)}$.
Nous reviendrons plus tard sur le calcul de ces bornes.

La programmation dynamique hybride se présente alors comme suit :

Algorithme III.5 - *La programmation dynamique hybride (HDP)*

Initialisation :
$\mathcal{L}_0 = \{(0,0)\}$, $\mathcal{L}_{sec} = \emptyset$.
Initialiser $\underline{v}(S(\mu)) = \underline{v}(MKP)$ (où $\underline{v}(MKP)$ est une borne inférieure de (MKP)).

Construction des listes :
Pour $j = 1$ à n
$\mathcal{L}'_j = \mathcal{L}_{j-1} \oplus (w_j, p_j)$,
Retirer les états $(w,p) \in \mathcal{L}'_j$ non-réalisables pour (MKP),
$\mathcal{L}_j = \mathcal{L}_{j-1} \cup \mathcal{L}'_j$,
Pour tous les états $(w,p) \in \mathcal{L}_j$, calculer $\overline{v}_{(w,p)}$ et $\underline{v}_{(w,p)}$,
Mise à jour des bornes :
$p_{max} = max\ \{p \mid (w,p) \in \mathcal{L}_j\}$ *et* $v_{max} = max\ \{\underline{v}_{(w,p)} \mid (w,p) \in \mathcal{L}_j\}$,
Si v_{max} est une borne admissible pour (MKP), $\underline{v}(MKP) = max\ \{\underline{v}(MKP), v_{max}\}$,
Sinon $\underline{v}(MKP) = max\ \{\underline{v}(MKP),\ p_{max}\}$,
$\underline{v}(S(\mu)) = max\ \{\underline{v}(S(\mu)),\ v_{max}\}$,
Mise à jour de $\mathcal{L}_{sec}$:
$\mathcal{D}_j = \{(w,p) \mid (w,p)$ *est dominé ou* $\overline{v}_{(w,p)} \leq \underline{v}(S(\mu))\}$,
$\mathcal{D}_j = \mathcal{D}_j - \{(w,p) \mid \overline{v}_{(w,p)} \leq \underline{v}(MKP)\}$,
$\mathcal{L}_{sec} = \mathcal{L}_{sec} \cup \mathcal{D}_j$ *et* $\mathcal{L}_j = \mathcal{L}_j - \mathcal{D}_j$,
Fin pour.

A la fin de cet algorithme, nous récupérons donc :
– $\underline{v}(MKP)$, une borne inférieure de (MKP), et
– $\mathcal{L}_{sec}$, une liste d'états qui nous servira à améliorer la précédente borne.

Calcul des bornes supérieures associées à un état

A une étape $k \in \{1, ..., n-1\}$ de la programmation dynamique, nous construisons la liste $\mathcal{L}_{k+1}$ à partir de la liste $\mathcal{L}_k$. Soit $(w,p) \in \mathcal{L}_k$, et notons $x^{(w,p)} = (x_1^{(w,p)}, ..., x_k^{(w,p)})$ les variables associées.Nous avons donc :
– $w = \sum_{j=1}^{k} w_j . x_j^{(w,p)}$ et

– $p = \sum_{j=1}^{k} p_j.x_j^{(w,p)}$.

On considère alors les sous problèmes suivants :

$$(MKP(w,p)) \begin{cases} \max \sum_{j=k+1}^{n} p_j.x_j + p, \\ \text{s.c.} \sum_{j=k+1}^{n} w_{i,j}.x_j \leq c - \sum_{j=1}^{k} w_{i,j}.x_j^{(w,p)} \\ x_j \in \{0,1\},\ j \in \{k+1,...,n\}, \end{cases} \tag{III.17}$$

et

$$(S(w,p)) \begin{cases} \max \sum_{j=k+1}^{n} p_j.x_j + p, \\ \text{s.c.} \sum_{j=k+1}^{n} w_j.x_j \leq c_i - w,\ i \in \{1,...,m\}, \\ x_j \in \{0,1\},\ j \in \{k+1,...,n\}. \end{cases} \tag{III.18}$$

Pour le calcul de $\overline{v}_{(w,p)}$, deux options s'offrent à nous, à savoir le calcul à partir de la relaxation continue de :
– $(MKP(w,p))$, ce qui implique des temps de calcul plus long mais une meilleure élimination des éléments par bornes supérieures, ou
– $(S(w,p))$ plus simple à résoudre avec la méthode de Dantzig décrite page 28.

Bien entendu, pour $k = 0$, nous avons égalité des bornes avec le choix du multiplicateur fait précédemment (voir page 66). Mais l'écart entre ces bornes se creuse progressivement avec k. On pourrai envisager de recalculer pour chaque sous-problème $(S(w,p))$ un vecteur multiplicateur, ce qui reviendrait à résoudre le dual de la relaxation continue de $(MKP(w,p))$, ce qui est équivalent à résoudre la relaxation continue de $(MKP(w,p))$.

Calcul des bornes inférieures associées à un état

Nous reprenons les mêmes notations ainsi que les sous-problèmes définis précédemment, et là aussi nous avons les deux mêmes options, à savoir la résolution approchée de :

– $(MKP(w,p))$: ce qui revient à faire de la programmation dynamique sur (MKP) en éliminant les états dominés du point de vue de la relaxation surrogate (dans ce cas on prendra $\underline{v}(S(\mu)) = \underline{v}(MKP)$),
– $(S(w,p))$: ce qui revient à faire de la programmation dynamique sur $(S(w,p))$ en conservant les états réalisables pour (MKP).

On peut déjà présager que la première solution donnera des bornes de meilleures qualités mais avec des temps de calcul moins bons qu'avec la deuxième option. De plus cette dernière nous permettrait de construire une heuristique dont les temps de calculs sont quasiment indépendants du nombre de contraintes du problème original (si on adopte le calcul des bornes supérieures via $(S(w,p))$).

On appellera :
– *HDP_S*, l'algorithme HDP intégrant le calcul des bornes inférieure d'un état (w,p) à partir de $(S(w,p))$ et
– *HDP_MKP*, celui utilisant $(MKP(w,p))$.

Remarque: *Dans l'algorithme III.5, il est important de comprendre que :*
- *pour HDP_S, deux fonctions objectives sont gérées, à savoir celle de (MKP), avec* $\underline{v}(MKP)$*, et celle de* $(S(\mu))$*, avec* $\underline{v}(S(\mu))$ *;*
- *pour HDP_MKP, nous prenons* $\underline{v}(S(\mu)) = \underline{v}(MKP)$*, et de ce fait, dans* $\mathcal{L}_{sec}$ *sont conservés que les états dominés du point de vue de la relaxation surrogate ;*

La méthode approchée retenue pour le calcul des bornes inférieure est la méthode du Glouton (voir page 48). En s'inspirant des travaux de Fréville et Plateau pour leur heuristique AGNES, nous avons adopter ici le classement des indices des variables suivant nous permettant d'exploiter la relaxation surrogate :

$$\frac{p_1}{w_1} \geq \ldots \geq \frac{p_j}{w_j} \geq \ldots \geq \frac{p_n}{w_n}. \tag{III.19}$$

Nous appellerons cette heuristique *Glouton_Surr*. La borne obtenue avec *Glouton_Surr* est d'une façon générale meilleure que celle du *Glouton* où $u = (1, ..., 1)^t$ pour des temps de calculs identiques. Ceci est illustré dans le tableau III.6 sur certaines instances de la littérature.

Nom de l'instance	Ecart à l'optimalité (%)	
	Glouton	Glouton_Surr
Petersen	3,13	4,70
Fréville & Plateau	12,74	6,54
Senju & Toyoda	1,14	0,28
Weingartner & Ness	3,23	1,77
Shi	4,63	0,51

Tab. III.6 – Comparaison de deux algorithmes gloutons.

Afin d'améliorer cette borne, nous nous sommes intéressés à la solution de $\overline{v}_{(w,p)}$. Dans certains cas, cette solution est admissible pour le problème considéré, et dans le cas où elle est meilleure que celle donné par l'heuristique, nous prendrons cette borne comme borne inférieure. Cette procédure sera nommé *Check_BS*, elle reçoit comme argument la solution $\overline{x}$ associée à une borne supérieure et permet donc de vérifier la réalisabilité de cette dernière.

Algorithme III.6 - *Check_BS(*$\overline{x}$*) :*

REALISABLE=true ;

Pour $j \in 1, ...n$
  Si $\overline{x}_j \notin \{0, 1\}$,
    REALISABLE=false ;
    Retourner REALISABLE ;
    STOP ;
  Fin Si
Fin Pour

Si $\overline{x}$ *est non-réalisable pour* (MKP)*, REALISABLE=false fin Si ;*

Retourner REALISABLE ;

Remarque: *Dans le cas où Check_BS retourne true, il est inutile de continuer à explorer dans la direction de l'état considéré car une solution optimale du sous-problème a été trouvé.*

Réduction de variables

Nous avons combiné l'algorithme III.5 avec la méthode de réduction présentée à la section II.3.1, page 30, et repris section III.3.1. Il est important de comprendre que les variables sont fixées en accord avec le problème (MKP), les variables ne sont pas réduites en fonction du problème surrogate. Ceci nous permet de conserver la réalisabilité des états explorés.

Le calcul des bornes nécessaires se fait au début de l'algorithme et sont recalculées à chaque fois que des variables sont fixées. Ce test de réduction est utilisé à chaque fois que la borne inférieure de (MKP) est améliorée.

III.3.3 Comparaison des heuristiques

On se rend compte que du point de vue algorithmique, plusieurs choix s'offrent à nous sur le plan :
– classement de variables,
– calcul des bornes supérieures,
– calcul des bornes inférieures.
En ce qui concerne le choix des calculs des bornes on se limitera aux deux cas extrêmes où dans un première cas, le temps de calcul est privilégié, combinaison de l'algorithme *HDP_S* avec calcul des bornes supérieures à partir de la relaxation, et dans un deuxième cas, on privilégie la qualité de la borne, combinaison de *HDP_MKP* avec calcul des bornes sans relaxation. On adoptera la même notation, *HDP_S* et *HDP_MKP*, pour ses deux heuristiques. Les premiers résultats obtenues avec *HDP_MKP* furent présentés dans [BOY 05].

Pour le classement des indices des variables, nous présenterons là aussi deux cas, à savoir les cas qui ont donné les meilleurs résultats :
– *C_BS*, classement selon les bornes supérieures croissantes (page 75) et
– *C_PR*, classement selon les profits réduits décroissants,
– *C_PW*, classement selon les rapports profits sur poids de la relaxation surrogate décroissants. (équation III.19).

Remarque: *Il serait plus logique d'adopter le classement selon les profits réduits croissants en accord avec le schéma présenté page 75. Mais en pratique les tests numériques nous ont montré que peu de variables sont réduites. Le classement selon les profits réduits décroissants permet "d'aiguiller" au plus tôt les états vers une solution potentiellement optimale.*

Les tests numériques présentés dans les tableaux III.7, III.8 et III.9 ont été menés sur les petites instances de la littérature. Nous avons voulu comparer les temps de calcul, ainsi que les écarts relatifs à la valeur optimale.

Ces tests nous montrent que *HDP_MKP* permet, d'une façon générale, d'avoir la meilleure approximation de la valeur optimale, avec l'obtention d'une égalité dans prêt de 80% des cas. Cependant cette approche entraîne des temps de calculs presque 100 fois supérieures à ceux de *HDP_S*, ce qui la rend inadapté pour des problèmes de grandes tailles. L'algorithme qui donne

le meilleure compromis entre temps de calcul et qualité de la borne semble être *HDP_S* avec le classement de variable *C_PW* (rapport profits sur poids de la relaxation surrogate décroissant).

Nom de l'instance	HDP_S			HDP_MKP		
	C_PR	C_BS	C_PW	C_PR	C_BS	C_PW
Petersen	4,69%	2,89%	1,30%	0,98%	1,00%	1,01%
Fréville & Plateau	5,89%	5,37%	3,46%	0,00%	0,05%	0,15%
Senju & Toyoda	0,28%	0,43%	0,72%	0,00%	0,00%	0,00%
Weingartner & Ness	1,77%	1,77%	0,58%	0,00%	0,01%	0,04%
Shi	0,51%	0,42%	0,35%	0,00%	0,00%	0,09%

TAB. III.7 – Ecart à l'optimalité des algorithmes HDP.

Nom de l'instance	HDP_S			HDP_MKP		
	C_PR	C_BS	C_PW	C_PR	C_BS	C_PW
Petersen	0/7	0/7	1/7	5/7	4/7	3/7
Fréville & Plateau	1/8	1/8	0/8	8/8	7/8	5/8
Senju & Toyoda	1/2	0/2	0/2	2/2	2/2	2/2
Weingartner & Ness	1/8	1/8	2/8	7/8	7/8	7/8
Shi	6/30	8/30	9/30	30/30	30/30	27/30

TAB. III.8 – Optimalité avec les algorithmes HDP.

Nom de l'instance	HDP_S			HDP_MKP		
	C_PR	C_BS	C_PW	C_PR	C_BS	C_PW
Petersen	1,43ms	8,57ms	2,86ms	112,86ms	237,14ms	185,71ms
Fréville & Plateau	5,00ms	8,75ms	5,00ms	321,25ms	353,75ms	320,00ms
Senju & Toyoda	10,00ms	25,00ms	10,00ms	955,00ms	2000,00ms	965,00ms
Weingartner & Ness	3,75ms	5,00ms	2,50ms	36,25ms	35,00ms	41,25ms
Shi	4,33ms	6,66ms	4,33ms	36,00ms	80,00ms	38,33ms

TAB. III.9 – Temps de calcul des algorithmes HDP.

III.3.4 HDP augmenté

Les états, qui ont été conservés dans la liste secondaire $\mathcal{L}_{sec}$, sont des états pour lesquels nous n'avons aucune garantie qu'ils ne peuvent mener à une meilleure solution au sens du problème multi-contraint. Ces états présentent donc un grand intérêt si l'on veut essayer d'améliorer la borne obtenue précédemment. Cette recherche pourrait s'apparenter à une recherche au voisinage de ces états.

Dans ce mémoire deux méthodes de recherche complémentaire sont présentées. La première, celle présentée ici, est la plus simple et la moins gourmande en temps de calcul. La deuxième,

qui complète la première, est présentée dans le prochain chapitre et est plus évoluée mais permet d'améliorer sensiblement la borne.

HDP_augmenté consiste à estimer une borne inférieure des états $(w,p) \in \mathcal{L}_{sec}$ via une heuristique gloutonne, à savoir la même présentée précédemment pour HDP (voir page 79). Une partie des variables des états conservés dans $\mathcal{L}_{sec}$ ont été soit traitées par la programmation dynamique, soit fixées par le processus de fixation de variables. La recherche d'une solution approchée est donc améliorée par ce "pré-traitement".

Algorithme III.7 - *HDP_augmenté :*

Tant que $\mathcal{L}_{sec} \neq \emptyset$
Prendre un couple $(w,p) \in \mathcal{L}_{sec}$,
$\mathcal{L}_{sec} = \mathcal{L}_{sec} - (w,p)$,
Calculer $\underline{v}_{(w,p)}$,
Si $\underline{v}(MKP) \leq \underline{v}_{(w,p)}$, $\underline{v}(MKP) = \underline{v}_{(w,p)}$.
Fin tant que.

Nous présentons quelques tests numériques traduisant l'amélioration apportée à la borne obtenue précédemment et les temps de calcul supplémentaire mis en jeu. Les résultats sont présentés dans les tableaux III.10, III.11 et III.12.

On peut voir à travers ces tests que *HDP_augmenté* permet d'améliorer de façon significative les bornes obtenues précédemment, notamment lorsque l'on compare les tableaux III.11 et III.8 présentant le nombre de fois où l'optimalité est atteinte. Cette amélioration est toutefois beaucoup plus marquée pour *HDP_S* que pour *HDP_MKP*, qui avait déjà des bornes très proches de l'optimum.

Du point de vue des temps de calcul, on constate que, proportionnellement, *HDP_augmenté* entraîne une augmentation du temps de calcul qui est peu importante avec *HDP_MKP*, mais qui peut devenir important pour *HDP_S*. Cependant les temps de calcul global pour *HDP_S* reste très en dessous de ceux de *HDP_MKP*. On notera la multiplication par presque 7 du temps de calcul pour les problèmes de Petersen avec *HDP_S+C_PR*. En fait, on peut constater que plus la borne fournie par les différents algorithmes HDP est proche de l'optimum, plus le temps de calcul engendré par *HDP_augmenté* est faible.

HDP_S+C_PR semble être l'algorithme proposant le meilleur compromis entre temps de calcul et qualité de la borne. Cette version de l'algorithme fut présentée dans [BOY 06]. Cependant on constatera les très bons résultats, en terme d'approximation, de l'algorithme *HDP_MKP+C_PR* où la valeur optimale n'a pas été atteinte que dans un seul cas.

Nom de l'instance	HDP_S			HDP_MKP		
	C_PR	C_BS	C_PW	C_PR	C_BS	C_PW
Petersen	0,03%	0,09%	0,04%	0,00%	0,01%	0,03%
Fréville & Plateau	0,22%	0,27%	0,49%	0,00%	0,05%	0,15%
Senju & Toyoda	0,00%	0,07%	0,00%	0,00%	0,00%	0,00%
Weingartner & Ness	0,00%	0,01%	0,05%	0,00%	0,00%	0,04%
Shi	0,00%	0,01%	0,02%	0,00%	0,00%	0,09%

TAB. III.10 – Ecart à l'optimalité des algorithmes *HDP+HDP_augmenté*.

Nom de l'instance	HDP_S			HDP_MKP		
	C_PR	C_BS	C_PW	C_PR	C_BS	C_PW
Petersen	6/7	5/7	6/7	7/7	6/7	5/7
Fréville & Plateau	6/8	6/8	4/8	8/8	7/8	5/8
Senju & Toyoda	2/2	1/2	2/2	2/2	2/2	2/2
Weingartner & Ness	8/8	7/8	5/8	7/8	8/8	7/8
Shi	30/30	28/30	27/30	30/30	30/30	27/30

TAB. III.11 – Optimalité avec les algorithmes *HDP+HDP_augmenté*.

Nom de l'instance	HDP_S			HDP_MKP		
	C_PR	C_BS	C_PW	C_PR	C_BS	C_PW
Petersen	7,14ms	15,71ms	2,95ms	121,43ms	250,00ms	218,57ms
Fréville & Plateau	18,87ms	17,50ms	27,50ms	361,25ms	408,75ms	372,50ms
Senju & Toyoda	10,00ms	40,00ms	35,00ms	1070,00ms	2080,00ms	1050,00ms
Weingartner & Ness	6,25ms	5,00ms	3,75ms	37,50ms	36,25ms	43,75ms
Shi	6,00ms	9,66ms	4,66ms	36,33ms	85,67ms	39,33ms

TAB. III.12 – Temps de calcul des algorithmes *HDP+HDP_augmenté*.

III.4 Une méthode coopérative de résolution exacte

Au cours du déroulement de l'algorithme de *HDP*, nous avons conservé dans $\mathcal{L}_{sec}$ les états éliminés par la programmation dynamique pour lesquels nous avons aucune garantie qu'ils ne vont pas mener à une solution meilleure pour (MKP). Il est alors aisé de construire à partir de *HDP* une méthode de résolution exacte pour (MKP). En effet, il suffit de continuer l'exploration de ces états là où la programmation dynamique s'est arrêtée.

Les états de $\mathcal{L}_{sec}$ ayant des profondeurs d'exploration différentes, nous avons opté pour un algorithme de branch-and-bound. On construit alors une méthode coopérative combinant programmation dynamique et branch-and-bound. Cette approche fut présentée dans [Boy].

Un état $(w, p) \in \mathcal{L}_{sec}$ devient alors un noeud, que l'on notera $\mathcal{N}$, pour le branch-and-bound. A ce noeud $\mathcal{N}$, on associe les paramètres suivants :
– $I(\mathcal{N})$, l'ensemble des indices des variables libres (non-traitées par un processus d'exploration et non-fixées),
– $J_0(\mathcal{N})$, l'ensemble des indices des variables fixées à 0,
– $J_1(\mathcal{N})$, l'ensemble des indices des variables fixées à 1 et
– $x(\mathcal{N})$, sa solution associée.
Nous avons donc ;

$$I(\mathcal{N}) \cup J_0(\mathcal{N}) \cup J_1(\mathcal{N}) = \{1, ..., n\} \; et$$
$$pour \; j \in J_\epsilon(\mathcal{N}), \; x_j(\mathcal{N}) = \epsilon \; avec \; \epsilon \in \{0, 1\}.$$

Ainsi nous pouvons redéfinir le sous-problème $(MKP(\mathcal{N}))$ associé au noeud $\mathcal{N}$ de la façon suivante :

$$(MKP(\mathcal{N}))\begin{cases} \max \sum_{j\in I(\mathcal{N})} p_j.x_j + \sum_{j\in J_1(\mathcal{N})} p_j, \\ \text{s.c.} \sum_{j\in I(\mathcal{N})} w_{i,j}.x_j \leq c_i - \sum_{j\in J_1(\mathcal{N})} w_{i,j},\ i \in \{1,...,m\}, \\ x_j \in \{0,1\},\ j \in I(\mathcal{N}). \end{cases} \quad \text{(III.20)}$$

Sa relaxation continue sera notée $\overline{(MKP(\mathcal{N}))}$.

La mise en place d'un algorithme de branch-and-bound nécessite un certain nombre de choix (voir section II.5.1 page 44) :
– le principe d'évaluation : le calcul des bornes,
– le principe de séparation : le choix de la variable sur lequel est fait le branchement,
– la stratégie de parcours : le choix du noeud à explorer,
Les différentes stratégies retenues sont décrites dans la suite.

III.4.1 Calcul des bornes

Le calcul des bornes supérieures est souvent le processus consommant le plus de temps de calcul. Pour cette raison, pour un noeud $\mathcal{N}$, il est souvent commode de résoudre la relaxation continue d'une relaxation surrogate, voir même de résoudre une relaxation lagrangienne, du problème $(MKP(\mathcal{N}))$. Cependant, nous avons opté pour la résolution par le simplexe de $\overline{(MKP(\mathcal{N}))}$ pour deux raisons :
– une meilleure qualité de la borne supérieure et
– obtention des profits réduits pour la réduction de variable (voir page 31).
Ainsi, à chaque fois qu'un noeud est choisi, une borne supérieure est évaluée. Si le noeud est conservé, on essaie de fixer des variables à partir des profits réduits. Ceci nous permet de rentabiliser le temps de calcul passé dans le simplexe.

En ce qui concerne le calcul des bornes inférieures le premier choix retenu fut le même algorithme glouton que précédemment (voir page 79). Nous avons alors testé une heuristique gloutonne, *Glouton_Continue*, basée sur la solution associée à la borne supérieure (voir algorithme III.8). Ainsi, nous construisons une solution admissible à partir de la solution non-entière de la relaxation continue. Cette heuristique permet d'une façon générale d'améliorer la convergence vers la solution optimale.

Algorithme III.8 - *Glouton_Continue :*

Soit $\overline{x}$ *une solution optimale de la relaxation continue de* (MKP) *;*
Notons $J = \{j \mid \overline{x}_j \in \{0,1\}\}$ *;*
On suppose alors les indices $j \notin J$ *classées selon les* $\overline{x}_j$ *décroissants ;*

Initialisation :
$\underline{x} = (\underline{x}_1, ..., \underline{x}_n)^t$, *tel que si* $j \in J$ $\underline{x}_j = \overline{x}_j$, *sinon* $\underline{x}_j = 0$.
$\underline{v} = \sum_{j\in J} p_j.\underline{x}_j$ *;*
$Q = c - \sum_{j\in J} W^j.\underline{x}_j$, *vecteur représentant la place restante dans le sac à dos ;*

Pour $j \notin \{1, ..., n\}$
  Si $Q - W^j \geq 0$

$\underline{x}_j = 1$;
$\underline{v} = \underline{v} + p_j$;
$Q = Q - W^j$;
Fin Si ;
Fin Pour ;

Retourner $\underline{v}$ *et* $\underline{x}$.

III.4.2 La stratégie de branchement

Soit $\mathcal{N}$ un noeud pendant du branch-and-bound.

La stratégie de branchement choisie vise à minimiser la non-réalisabilité de la solution obtenue par la résolution de $\overline{(MKP(\mathcal{N}))}$. Si l'on note $\overline{x}(\mathcal{N})$ cette solution, l'indice $k \in I(\mathcal{N})$ de la variable sur laquelle sera fait le branchement est tel que :

$$k = argmin\,\{|\overline{x}_j(\mathcal{N}) - 0,5| \mid j \in I(\mathcal{N})\}\,. \tag{III.21}$$

On effectue donc le branchement sur la variable libre dont la valeur dans $\overline{x}(\mathcal{N})$ est la plus proche de $0,5$. Ainsi à chaque étape de l'algorithme nous ne faisons pas de calcul de bornes supérieures redondantes. En effet, si on branche sur la variable $x_k(\mathcal{N})$ tel que $\overline{x}_k(\mathcal{N}) \in \{0,1\}$, le calcul de la borne supérieure d'un des deux fils résultants est inutile car elle sera la même. De plus, on concentre l'exploration sur les variables fractionnaires.

III.4.3 La stratégie de parcours

Nous avons retenu la stratégie de parcours "Meilleur d'abord". Elle est connue pour étant celle donnant les meilleurs temps de calcul comparée aux autres.

Durant le déroulement du branch-and-bound, l'ensemble des noeuds $\mathcal{N}$ est donc classé dans une liste, $\mathcal{L}$, selon leur borne supérieure décroissante. A chaque étape du branch-and-bound, le premier noeud de cette liste est retiré pour être traité par l'algorithme. Les fils résultants sont insérés dans cette liste en conservant l'ordre établi.

III.4.4 Algorithme

La méthode de branch-and-bound utilisée, que l'on notera *B&B*, est détaillée dans l'algorithme suivant :

Algorithme III.9 - *B&B :*

Initialisation :
$\mathcal{L} = \{\mathcal{N}_{\mathcal{VIDE}}\}$, *avec* $\mathcal{N}_{\mathcal{VIDE}}$ *le noeud correspondant au sac à dos vide ;*
Initialiser $v(MKP)$ *par une borne inférieure de (MKP) avec Glouton_Continue ;*

Tant que $\mathcal{L} \neq \emptyset$
Soit $x(\mathcal{N})$ *le premier élément de* $\mathcal{L}$ *(celui ayant la meilleure borne supérieure) ;*
$\mathcal{L} = \mathcal{L} - \{\mathcal{N}\}$;

Résoudre $\overline{(MKP(\mathcal{N}))}$ *et récupérer sa valeur optimale* $\overline{v}(\mathcal{N})$ *et* $\overline{x}(\mathcal{N})$ *sa solution associée, ainsi que les profits réduits* $p_{red}(\mathcal{N})$ *;*
Si $\lfloor \overline{v}(\mathcal{N}) \rfloor > v$*, alors*
Réduire les variables à partir des profits réduits $p_{red}(\mathcal{N})$ *et mettre à jour* $I(\mathcal{N})$*,* $J_1(\mathcal{N})$ *et* $J_0(\mathcal{N})$ *;*
Calculer $\underline{v}(\mathcal{N})$ *une borne inférieure de* $(MKP(\mathcal{N}))$ *avec Glouton_Continue ;*
Si $v(MKP) < \underline{v}(\mathcal{N})$*, alors* $v(MKP) = \underline{v}(\mathcal{N})$ *fin Si ;*
Choisir la variable de branchement k *tel que* $k = argmin\{|\overline{x}_j - 0,5| \mid j \in I(\mathcal{N})\}$ *;*
Création des 2 fils $\mathcal{N}_1$ *et* $\mathcal{N}_0$ *de* $\mathcal{N}$ *:*
$\rightarrow \mathcal{N}_1$ *:* $I(\mathcal{N}_1) = I(\mathcal{N}) - \{k\}$*,* $J_1(\mathcal{N}_1) = J_1(\mathcal{N}) \cup \{k\}$ *et* $J_0(\mathcal{N}_1) = J_0(\mathcal{N})$ *et*
$\rightarrow \mathcal{N}_0$ *:* $I(\mathcal{N}_0) = I(\mathcal{N}) - \{k\}$*,* $J_0(\mathcal{N}_0) = J_0(\mathcal{N}) \cup \{k\}$ *et* $J_1(\mathcal{N}_0) = J_1(\mathcal{N})$ *;*
Si $I(\mathcal{N}_1) \neq \emptyset$ $(\Leftrightarrow\ I(\mathcal{N}_0) \neq \emptyset)$*, alors*
Fixer $\overline{v}(\mathcal{N}_1) = \overline{v}(\mathcal{N}_0) = \overline{v}(\mathcal{N})$ *;*
Insérer $\mathcal{N}_0$ *dans* $\mathcal{L}$ *en conservant le classement des bornes supérieures décroissantes ;*
De même pour $\mathcal{N}_1$ *si ce noeud est réalisable ;*
Fin Si ;
Fin Si ;
Fin Tant que ;

Dans le cadre de la méthode coopérative, le branch-and-bound est initialisé comme suit :
– la liste $\mathcal{L}$ est initialisée avec les éléments de la liste secondaire $\mathcal{L}_{sec}$ et
– la borne $v(MKP)$ est initialisée avec la borne obtenue avec *HDP+HDP_augmenté*.
Ainsi on reprend directement l'exploration là où elle s'est arrêtée.

Les résultats obtenus sur les petites instances de la littérature avec les différentes variantes de *HDP* sont présentés dans les tableaux III.13 et III.14.

Les tableau III.13 montre le temps de calcul total moyen nécessaire pour la résolution exacte de ces instances avec les différents algorithmes *HDP*. On peut constater les très bonnes performances de *HDP_S* avec le classement de variable *C_PR* (profits réduits décroissants) qui dans tous les cas est le plus rapide. Bien que l'heuristique *HDP_MKP+HDP_augmenté* fournit les meilleurs bornes, le temps nécessaire à leur obtention n'est pas compensé par *B&B*.

Une autre analyse intéressante est le relevé du pourcentage de temps passé dans *B&B* (tableau III.14). On constate, d'une façon générale, qu'avec *HDP_S*, ce pourcentage est important alors les temps de calcul de l'heuristique était relativement faible (voir tableau III.12). Avec *HDP_MKP* nous sommes dans la configuration inverse. Il semble donc que la difficulté de résolution des instances soit essentiellement concentrée dans la programmation dynamique avec *HDP_MKP*, et que cette difficulté se déplace dans le branch-and-bound avec *HDP_S*.

Nom de l'instance	HDP_S			HDP_MKP		
	C_PR	C_BS	C_PW	C_PR	C_BS	C_PW
Petersen	24,28ms	58,57ms	44,28ms	145,71ms	291,43ms	270,00ms
Fréville & Plateau	150,00ms	158,75ms	166,25ms	465,00ms	498,75ms	506,25ms
Senju & Toyoda	475,00ms	595,00ms	455,00ms	1420,00ms	2490,00ms	1385,00ms
Weingartner & Ness	6,25ms	6,25ms	6,25ms	37,50ms	37,50ms	43,75ms
Shi	6,00ms	11,33ms	7,33ms	36,33ms	86,33ms	39,00ms

TAB. III.13 – Temps de calcul des algorithmes *HDP+HDP_augmenté+B&B*.

Nom de l'instance	HDP_S			HDP_MKP		
	C_PR	C_BS	C_PW	C_PR	C_BS	C_PW
Petersen	70,59%	73,17%	93,55%	16,67%	14,22%	19,05%
Fréville & Plateau	87,50%	88,98%	83,46%	22,31%	18,05%	26,42%
Senju & Toyoda	97,89%	93,28%	92,31%	24,65%	16,47%	24,19%
Weingartner & Ness	0,00%	20,00%	40,00%	0,00%	3,33%	0,00%
Shi	0,00%	14,71%	36,36%	0,00%	0,87%	4,27%

TAB. III.14 – Pourcentage du temps passé dans *B&B* avec *HDP+HDP_augmenté+B&B*.

III.5 La phase de branch and bound limité

Nous présentons ici une deuxième recherche au voisinage qui vient compléter les algorithmes *HDP+HDP_augmenté* présenté précédemment. L'objectif est d'essayer d'améliorer encore la solution approchée obtenue précédemment toujours en s'aidant de la liste secondaire $\mathcal{L}_{sec}$. Elle se base sur l'algorithme de branch-and-bound présenté précédemment.

Pour chaque état conservé dans $\mathcal{L}_{sec}$, une partie des variables ont été traitées par la programmation dynamique. L'idée est d'aller un peu plus loin dans cette exploration en se basant sur un algorithme de branch-and-bound. Quand on utilise un algorithme de branch-and-bound classique, les temps de calcul restent importants. Nous avons donc modifié cette algorithme afin d'avoir une exploration que l'on pourrait qualifier d'"approchée" et ainsi avoir des temps de calcul plus faible. Les meilleures résultats obtenus par cette approche ont été rapportés dans [BOY 07].

On reprend l'ensemble des notations vues précédemment.

Tout d'abord, nous avons forcé la réduction de variables afin de concentrer l'exploration sur le coeur du sous-problème résultant des variables restées libres pour un état donné de $\mathcal{L}_{sec}$. Ce coeur est défini par les variables fractionnaires de la solution optimale de la relaxation continue de ce sous-problème. Le reste des variables est alors fixé à la valeur obtenue par cette dernière solution. La méthode de branch-and-bound est alors appliquée uniquement aux variables constituant le coeur. Cette réduction de variable est présenté dans l'algorithme III.10. *Red_Var* reçoit comme argument un noeud ($\mathcal{N}$).

Algorithme III.10 - *Red_Var($\mathcal{N}$)*

Résoudre la relaxation continue de $(MKP(\mathcal{N}))$ *;*
Récupérer sa solution optimale $\overline{x}(\mathcal{N})$ *;*

Pour $j \in \{1, ..., n\}$, *si* $\overline{x}_j(\mathcal{N}) \in \{0, 1\}$,
    Fixer $x_j(\mathcal{N})$ *à* $\overline{x}_j(\mathcal{N})$ *;*
    $I(\mathcal{N}) = I(\mathcal{N}) - \{j\}$ *;*
    $J_k(\mathcal{N}) = J_k(\mathcal{N}) \cup \{j\}$, *avec* $k = x_j(\mathcal{N})$ *;*
Fin Si

Les temps de calcul sont ici liés, en grande partie, à la taille de la liste secondaire $\mathcal{L}_{sec}$. Deux options peuvent alors être envisagées :
– *BB_lim_N* : limiter le nombre de noeuds considérés dans $\mathcal{L}_{sec}$,
– *BB_lim_T* : limiter le temps passé dans la phase de branch-and-bound limité.
Bien entendu, plusieurs autres possibilités existent, cependant nous nous sommes arrêtés sur ces deux dernières pour le moment. Dans la suite est présenté en détail ces différentes limitations.

III.5.1 *BB_lim_N* : limitation du nombre de noeuds dans $\mathcal{L}_{sec}$

Si on veut limiter le nombre de noeuds considérés dans $\mathcal{L}_{sec}$, il n'est pas facile de sélectionner ceux qui seront conservés et ceux qui seront rejetés. Le but est de retenir ceux qui sont les plus susceptibles de donner une solution optimale. Le critère de sélection d'un noeud ($\mathcal{N}$), retenu ici, est sa borne supérieure $\overline{v}(\mathcal{N})$, obtenue par la résolution de la relaxation continue de ($MKP(\mathcal{N})$).

On suppose les noeuds dans $\mathcal{L}_{sec}$ classés selon leurs bornes supérieures décroissantes. On note $K = |\mathcal{L}_{sec}|$, le cardinal de $\mathcal{L}_{sec}$. En considérant notre critère de sélection, il suffit de considérer les $\lfloor \gamma_n.K \rfloor$, $\gamma_n \in]0,1[$ et $\lfloor \gamma_n.K \rfloor \geq 1$, premiers noeuds de $\mathcal{L}_{sec}$. La nouvelle liste ainsi construite sera noté $\mathcal{L}_{sec}(\gamma_n)$.

Remarque: *Lorsque $\gamma_n = 1$, on retombe sur la méthode coopérative de résolution exacte vue dans la partie III.4.*

Pour les tests numérique des tableaux III.18, III.19 et III.20, γ_n a été fixé à $0,5$. Cette valeur de γ_n semble être le meilleur compromis entre temps de calcul et amélioration de la borne.

Nom de l'instance	HDP_S			HDP_MKP		
	C_PR	C_BS	C_PW	C_PR	C_BS	C_PW
Petersen	0,03%	0,09%	0,03%	0,00%	0,01%	0,03%
Fréville & Plateau	0,22%	0,27%	0,21%	0,00%	0,05%	0,15%
Senju & Toyoda	0,00%	0,07%	0,00%	0,00%	0,00%	0,00%
Weingartner & Ness	0,00%	0,01%	0,04%	0,00%	0,00%	0,04%
Shi	0,00%	0,01%	0,01%	0,00%	0,00%	0,09%

Tab. III.15 – Ecart à l'optimalité des algorithmes *HDP+HDP_augmenté+BB_lim_N*.

Nom de l'instance	HDP_S			HDP_MKP		
	C_PR	C_BS	C_PW	C_PR	C_BS	C_PW
Petersen	6/7	5/7	6/7	7/7	6/7	5/7
Fréville & Plateau	6/8	6/8	4/8	8/8	7/8	5/8
Senju & Toyoda	2/2	2/2	2/2	2/2	2/2	2/2
Weingartner & Ness	8/8	7/8	6/8	7/8	8/8	7/8
Shi	30/30	30/30	28/30	30/30	30/30	27/30

Tab. III.16 – Optimalité avec les algorithmes *HDP+HDP_augmenté+BB_lim_N*.

Nom de l'instance	HDP_S			HDP_MKP		
	C_PR	C_BS	C_PW	C_PR	C_BS	C_PW
Petersen	8,28ms	15,71ms	5,71ms	122,86ms	258,57ms	219,71ms
Fréville & Plateau	19,25ms	21,25ms	28,75ms	366,25ms	409,25ms	378,75ms
Senju & Toyoda	10,00ms	40,00ms	35,00ms	1075,00ms	2085,00ms	1055,00ms
Weingartner & Ness	6,25ms	5,00ms	3,75ms	38,75ms	37,00ms	43,75ms
Shi	6,00ms	11,33ms	4,66ms	37,33ms	86,33ms	39,66ms

TAB. III.17 – Temps de calcul des algorithmes *HDP+HDP_augmenté+BB_lim_N*.

III.5.2 *BB_lim_T* : limitation du temps de calcul

Le problème de l'option précédente est que même si l'on prend un γ_n très petit, rien ne garantie des temps de résolution raisonnables. Il est alors intéressant de pouvoir limiter directement ce dernier.

L'objectif d'une limitation du temps de calcul est de laisser suffisamment de temps pour les problèmes plus difficiles à résoudre, et d'arrêter précocement l'exploration pour les problèmes plus faciles. Cependant, on ne peut savoir à priori si un problème sera difficile ou pas à résoudre.

Ici, une partie de l'exploration a été faite via un algorithme de type programmation dynamique. Cet algorithme pourrait s'apparenter à celui d'un branch-and-bound avec comme stratégie de parcours la largeur d'abord. Le temps de calcul nécessaire à cette première phase peut dons être vu comme un critère reflétant la difficulté de résolution du problème considéré. En notant T ce temps, on peut limiter celui passé dans le branch-and-bound à $\gamma_t.T$, avec $\gamma_t > 0$.

Remarque: *De même, lorsque γ_t est suffisamment grand, on retombe sur la méthode coopérative de résolution exacte vue dans la partie III.4.*

La valeur adoptée pour γ_t pour les tests numérique des tableaux III.18, III.19 et III.20 est 1. Ainsi, dans le pire cas, on multiplie par deux le temps de calcul total par rapport à *HDP+HDP_augmenté*.

Nom de l'instance	HDP_S			HDP_MKP		
	C_PR	C_BS	C_PW	C_PR	C_BS	C_PW
Petersen	0,01%	0,03%	0,01%	0,00%	0,00%	0,00%
Fréville & Plateau	0,17%	0,20%	0,10%	0,00%	0,00%	0,00%
Senju & Toyoda	0,00%	0,07%	0,00%	0,00%	0,00%	0,00%
Weingartner & Ness	0,00%	0,01%	0,00%	0,00%	0,00%	0,00%
Shi	0,00%	0,00%	0,00%	0,00%	0,00%	0,08%

TAB. III.18 – Ecart à l'optimalité des algorithmes *HDP+HDP_augmenté+BB_lim_T*.

Nom de l'instance	HDP_S			HDP_MKP		
	C_PR	C_BS	C_PW	C_PR	C_BS	C_PW
Petersen	6/7	6/7	6/7	7/7	7/7	7/7
Fréville & Plateau	6/8	6/8	6/8	8/8	8/8	8/8
Senju & Toyoda	2/2	2/2	2/2	2/2	2/2	2/2
Weingartner & Ness	8/8	7/8	6/8	7/8	8/8	8/8
Shi	30/30	30/30	30/30	30/30	30/30	29/30

TAB. III.19 – Optimalité avec les algorithmes *HDP+HDP_augmenté+BB_lim_T*.

Nom de l'instance	HDP_S			HDP_MKP		
	C_PR	C_BS	C_PW	C_PR	C_BS	C_PW
Petersen	11,42ms	31,43ms	15,71ms	144,29ms	297,14ms	270,00ms
Fréville & Plateau	33,75ms	47,50ms	31,25ms	467,50ms	502,50ms	510,00ms
Senju & Toyoda	30,00ms	80,00ms	30,00ms	1415,00ms	2500,00ms	1395,00ms
Weingartner & Ness	6,24ms	10,00ms	7,50ms	37,50ms	37,00ms	43,75ms
Shi	6,00ms	11,67ms	8,33ms	35,66ms	85,33ms	39,33ms

TAB. III.20 – Temps de calcul des algorithmes *HDP+HDP_augmenté+BB_lim_T*.

III.5.3 Comparaison des deux approches

Les premiers tests numériques menés (voir les tableaux III.18, III.19 et III.20 et les tableaux III.18, III.19 et III.20) nous montrent que *BB_lim_T* permet d'obtenir une meilleure amélioration de la borne obtenue avec *HDP+HDP_augmenté* que *BB_lim_N*. Cette amélioration se paye au niveau des temps de calcul.

Limiter la taille de la liste secondaire ne semble donc pas être la meilleure approche si l'on veut essayer d'améliorer la borne obtenue par les heuristiques *HDP+HDP_augmenté*. Les temps de calcul sont relativement importants compte tenu de l'amélioration apportée. La limitation en temps de calcul permet au branch-and-bound d'explorer en partie l'ensemble des noeuds conservés et d'obtenir une meilleure amélioration .

Enfin, on peut remarquer aussi que les heuristiques basées sur *HDP_MKP* ont des temps de calcul peu sensibles à la limitation choisie. Ceci vient du fait que l'essentiel de la résolution a été faite dans la programmation dynamique.

III.6 Conclusion

Dans ce chapitre nous avons présenté les différentes méthodes que nous proposons pour la résolution du problème du sac à dos multidimensionnel. L'ensemble de ces méthodes est construit à partir de l'heuristique *HDP* qui est basée sur la résolution de la relaxation surrogate du MKP via un algorithme de type programmation dynamique. Cette heuristique peut-être facilement combinée à d'autres méthodes de résolution afin d'obtenir des heuristiques ou des méthodes coopératives de résolution exacte. Nous avons combiné *HDP* avec :

– *HDP_augmenté*, qui effectue des évaluations de bornes inférieures à partir d'une heuristique gloutonne,
– *BB_lim*, qui complète *HDP+HDP_augmenté* par un algorithme de type branch-and-bound bridé afin de garantir des temps de calcul raisonnables et

– un algorithme de branch-and-bound, afin de construire une méthode coopérative de résolution exacte.

Les choix algorithmiques ont été illustrés par des tests numériques sur des instances de la littérature de petite taille. Ces tests nous permettent d'avoir une première appréciation des performances des différentes algorithmes étudiés. Deux variantes de *HDP* ont été mises en parallèle :

– *HDP_S*, dont la résolution par programmation dynamique privilégie la relaxation surrogate du MKP et
– *HDP_MKP*, dont la résolution par programmation dynamique privilégie le MKP original.

Ces tests préliminaires nous montrent que les heuristiques *HDP_S* semblent donner les meilleurs temps de calcul par rapport aux heuristiques *HDP_MKP*. Ces dernières permettent néanmoins d'obtenir les meilleures approximations de la valeur optimale. Dans le chapitre suivant, nous présentons des tests numériques faits sur des instances de taille plus importante.

Chapitre IV

Les résultats numériques

IV.1 Introduction

Dans ce chapitre, nous présentons l'ensemble des tests numériques que nous avons menés avec les méthodes de résolution que nous proposons. Ces tests ont été menés sur des instances de taille importante issues de la littérature ou engendrées de manière aléatoire.

En particulier, nos heuristiques ont été comparées avec les heuristiques AGNES de Fréville et Plateau [FRE 94], ADP-based heuristics approach de Bertsimas et Demir [BER 02] et SMA de Hanafi, Fréville et El Abdellaoui [HAN 96]. Nos méthodes coopératives ont été comparées à une méthode classique de résolution exacte : le branch-and-bound.

Par ailleurs, nous présentons les tests effectués pour les deux techniques que nous proposons afin d'engendrer aléatoirement des instances difficiles. Ces instances ont été résolues de manière exacte avec CPLEX 9.0 et les temps de calcul ont été comparés avec les instances engendrées par des techniques classiques.

IV.2 Présentation des instances considérées

Les tests numériques présentés ici ont été réalisés sur les grandes instances de la littérature, ainsi que sur des instances engendrées de façon aléatoire. Concernant ces-dernières, nous avons considéré deux classes d'instances, à savoir :
– I_NC : une instances à problèmes non-corrélés et
– I_FC : une instances à problèmes fortement corrélés.
En notant n le nombre de variable d'un problème d'une de ces instances, nous avons engendré, pour chaque valeur de $n \in \{50, 100, 150, 200, 250, 300, 400, 500\}$, 25 problèmes différents. Les différents paramètres retenus sont présentés dans le tableau IV.1, sachant que tous les coefficients ont été tirés aléatoirement dans $I = [1, U_p] = [1, U_w] = [1, 1000]$.

Instance	m	ϵ	sc
I_NC	$n/2$	0.9	
I_NC	$n/2$	0.5	
I_FC	$n/10$	0.9	100
I_FC	$n/10$	0.5	100

Tab. IV.1 – Paramètres des instances aléatoires.

IV.3 Tests numériques sur les grandes instances de la littérature

L'ensemble des tests préliminaires effectué dans le chapitre précédent nous suggère que le classement de variable *C_PR* (profits réduits décroissants) permet d'avoir le meilleur compromis entre temps de calcul et qualité de la borne. Nous nous intéresserons donc à ce classement de variable dans cette partie. Les tests présentés ici ont été faits sur les instances de Chu et Beasley. Les valeurs optimales pour ces instances n'étant pas connues, les écarts relatifs ont été calculés par rapport à la valeur optimale de la relaxation continue de ces instances.

Afin de limiter les temps de résolution, nous arrêtons la résolution si elle dépasse 10 minutes (600 secondes). La borne retenue est alors la meilleure qui a été obtenue avant l'arrêt de la résolution.

IV.3.1 *HDP* et *HDP+HDP_augmenté*

Nous allons comparer les heuristiques *HDP_S* et *HDP_MKP*, ainsi que leur version complétée de *HDP_augmenté* avec des heuristiques de la littérature (voir partie II.4 page 37 pour plus de détails) :
– AGNES de Fréville et Plateau [FRE 94] ;
– ADP-based heuristics approach de Bertsimas et Demir [BER 02] ;
– Simple Multistage Algorithm (SMA) de Hanafi, Fréville et El Abdellaoui [HAN 96].
Les résultats obtenus sont présentés dans les tableaux IV.2 et IV.3. En comparant les performances des deux versions de l'heuristique proposée, *HDP_S* et *HDP_MKP*, on constate, d'une façon générale, les bons résultats de *HDP_S* aussi bien en temps de calcul qu'en qualité de la borne. Dans la suite, nous présenterons que les tests numériques de *HDP_S*, afin de nous concentrer sur la version de l'algorithme la plus prometteuse.

Si on compare *HDP_S* et *HDP_S+HDP_augmenté* aux autres heuristiques de la littérature (SMA, ADP et AGNES), on peut voir que *HDP_S+HDP_augmenté* permet d'obtenir une meilleure approximation de la valeur optimale pour un temps de calcul qui reste compétitif. On remarque que l'adjonction de *HDP_augmenté* permet d'améliorer de façon significative la borne obtenue avec *HDP_S*.

Nom de l'instance	Taille nxm	HDP				heuristiques		
		S	S+aug	MKP	MKP+aug	SMA	ADP	AGNES
Chu & Beasley 1	100x5	1,96	0,69	0,77	0,62	2,68	1,72	0,88
Chu & Beasley 2	250x5	0,58	0,21	0,30	0,16	1,17	0,58	0,29
Chu & Beasley 3	500x5	0,27	0,07	0,16	0,16	0,59	0,26	0,12
Chu & Beasley 4	100x10	2,87	1,25	1,36	1,08	3,60	1,97	1,54
Chu & Beasley 5	250x10	1,03	0,47	0,63	0,63	1,60	0,76	0,57
Chu & Beasley 6	500x10	0,54	0,21	0,82	0,82	0,80	0,38	0,26
Chu & Beasley 7	100x30	4,23	2,05	3,91	2,45	5,13	2,70	3,22
Chu & Beasley 8	250x30	1,7	0,90	1,95	1,95	2,60	1,18	1,41
Chu & Beasley 9	500x30	1,39	0,49	2,28	2,28	1,45	0,58	0,72

S : *HDP_S*
MKP : *HDP_MKP*
S+aug : *HDP_S+HDP_augmenté*
MKP+aug : *HDP_MKP+HDP_augmenté*

TAB. IV.2 – HDP : les grandes instances de la littérature (Écart relatif (%)).

Nom de l'instance	Taille nxm	HDP				heuristiques		
		S	S+aug	MKP	MKP+aug	SMA	ADP	AGNES
Chu & Beasley 1	100x5	0,03	0,07	9,88	10,25	0,15	0,12	0,10
Chu & Beasley 2	250x5	0,27	0,52	254,86	285,79	1,94	0,24	0,10
Chu & Beasley 3	500x5	1,50	2,07	600,00	600,00	15,63	1,03	0,34
Chu & Beasley 4	100x10	0,05	0,12	50,53	54,99	0,17	0,15	0,10
Chu & Beasley 5	250x10	0,45	0,94	600,00	600,00	2,39	0,34	0,10
Chu & Beasley 6	500x10	2,36	3,81	600,00	600,00	19,49	1,47	0,44
Chu & Beasley 7	100x30	2,49	5,36	522,17	542,41	0,39	0,24	0,10
Chu & Beasley 8	250x30	22,26	36,66	600,00	600,00	4,79	1,27	0,34
Chu & Beasley 9	500x30	81,31	88,07	600,00	600,00	40,80	4,10	1,03

S : *HDP_S*
MKP : *HDP_MKP*
S+aug : *HDP_S+HDP_augmenté*
MKP+aug : *HDP_MKP+HDP_augmenté*

TAB. IV.3 – HDP : les grandes instances de la littérature (Temps de calcul (s)).

IV.3.2 *HDP+HDP_augmenté+BB_lim_T*

Dans cette partie, nous allons mettre en avant l'amélioration apportée par *BB_lim_T* par rapport aux bornes obtenues précédemment. Comme annoncé précédemment, nous présenterons les résultats que pour *HDP_S*.

Le tableau IV.4 nous montre que la procédure *BB_lim_T*permet d'améliorer légèrement l'approximation de la valeur optimale. Ceci se paye bien entendu sur le temps de calcul, cependant ces temps restent bien en dessous de ceux obtenus avec *HDP_MKP*.

Nom de l'instance	Taille nxm	Ecart relatif (%)		Am. (%)	Temps de calcul de S+aug+BB_lim (s)
		S+aug	S+aug+BB_lim		
Chu & Beasley 1	100x5	0,69	0,64	2,75	1,89
Chu & Beasley 2	250x5	0,21	0,20	0,67	3,28
Chu & Beasley 3	500x5	0,07	0,06	0,40	5,60
Chu & Beasley 4	100x10	1,25	1,20	2,86	3,15
Chu & Beasley 5	250x10	0,47	0,45	1,07	2,75
Chu & Beasley 6	500x10	0,21	0,20	0,64	9,31
Chu & Beasley 7	100x30	2,05	2,04	3,63	9,45
Chu & Beasley 8	250x30	0,90	0,89	1,12	55,14
Chu & Beasley 9	500x30	0,49	0,48	0,59	188,34

S+aug : *HDP_S+HDP_augmenté*
S+aug+BB : *HDP_S+HDP_augmenté+BB_lim_T*
Am : Amélioration de la borne par rapport à *HDP_S+HDP_augmenté*

TAB. IV.4 – BB_lim_T : les grandes instances de la littérature.

IV.3.3 Les méthodes coopératives

Nous avons voulu comparer les performances des notre méthode hybride avec ceux d'une méthode classique, à savoir un branch-and-bound. Le branch-and-bound utilisé reprend les mêmes stratégies retenues pour la construction de notre méthode (voir page 83).

Nous avons limité le temps de calcul global de résolution à 10 minutes et nous avons comparé la borne obtenue par le branch-and-bound avec celle de notre méthode hybride en calculant un écart de la façon suivante :

$$Ecart = \frac{v_{BB} - v_{MC}}{v_{BB}}, \tag{IV.1}$$

avec v_{BB} et v_{MC} les bornes obtenues respectivement par le branch-and-bound et la méthode coopérative.

Bien entendu, dans le cas où le temps de résolution n'excède pas ces 10 minutes, nous obtenons les valeurs optimales et dans ce cas il est intéressant de comparer les temps de calcul.

On peut voir, au travers du tableau IV.5, les résultats encourageant de la méthode coopérative sur ces instances. Sur les deux premières instances, nous obtenons les valeurs optimales plus rapidement que le branch-and-bound. Lorsque le temps de résolution dépasse les dix minutes, on peut voir, d'une façon générale, la convergence un peu plus rapide de la méthode coopérative (si l'écart est négatif, alors $v_{BB} < v_{MC}$).

Instance	nxm	Ecart (%)	t_BB (s)	t_MC(s)
Chu & Beasley 1	100x5	0,00	166,60	160,06
Chu & Beasley 2	250x5	0,0038	501,61	493,56
Chu & Beasley 3	500x5	-0,0146	600,00	600,00
Chu & Beasley 4	100x10	0,00	533.95	600,00
Chu & Beasley 5	250x10	-0,0157	600,00	600,00
Chu & Beasley 6	500x10	-0,0525	600,00	600,00
Chu & Beasley 7	100x30	-0,0602	600,00	600,00
Chu & Beasley 8	250x30	0,0158	600,00	600,00
Chu & Beasley 9	500x30	-0.0223	600,00	600,00

Ecart : Ecart entre la borne obtenu avec la méthode coopérative et celle du branch-and-bound
t_BB : temps de calcul du branch-and-bound
t_MC : temps de calcul de la méthode coopérative

Tab. IV.5 – Méthode coopérative : les grandes instances de la littérature.

IV.4 Tests numériques sur les instances aléatoires

Nous présentons ici les test numériques effectués sur des instances contenant des problèmes engendrés aléatoirement, et en particulier sur les problèmes non-corrélés et sur les problèmes corrélés.

Au vue des résultats obtenus précédemment, nous limiterons les tests sur les problèmes engendrés aléatoirement à la version de l'algorithme la plus performante, à savoir celle basée sur *HDP_S*

avec le classement de variable *C_PR* (voir page 80). De plus *HDP+HDP_augmenté* étant la version de l'algorithme ayant le meilleure compromis entre temps de calcul et qualité de la borne, nous présenterons que les tests effectués sur cette approche heuristique.

IV.4.1 *HDP+HDP_augmenté*

Là aussi, les valeurs optimales pour ces instances n'étant pas connues, les écarts relatifs ont été calculés par rapport à la valeur optimale de la relaxation continue de ces instances.

Les tests sur les instances non-corrélées sont présentées dans les tableaux IV.6 et IV.7, et ceux sur les instances corrélées dans les tableaux IV.8 et IV.9. On constate que *HDP+HDP_augmenté* donne systématiquement une meilleure approximation de la valeur optimale sur l'ensemble des instances. Cependant, du point de vue temps de calcul, on constate que notre heuristique reste assez compétitif sur les instances non-corrélées et en particulier pour $\epsilon = 0,9$, mais sur les instances corrélées, ces temps deviennent plus importants.

ϵ	n	S+aug	ADP	SMA	AGNES
0,5	50	1,81	3,31	5,13	4,46
0,5	100	1,19	1,53	3,29	2,86
0,5	150	0,72	1,05	2,15	1,90
0,5	200	0,56	0,78	1,77	1,50
0,5	250	0,52	0,71	1,64	1,53
0,5	300	0,50	0,55	1,48	1,34
0,5	400	0,45	0,48	1,20	0,94
0,5	500	0,36	0,44	1,08	0,87
0,9	50	0,45	0,52	0,94	0,51
0,9	100	0,15	0,44	0,31	0,37
0,9	150	0,12	0,19	0,26	0,25
0,9	200	0,09	0,12	0,33	0,28
0,9	250	0,08	0,10	0,21	0,18
0,9	300	0,06	0,08	0,19	0,13
0,9	400	0,04	0,06	0,13	0,08
0,9	500	0,03	0,05	0,10	0,10

S+aug : *HDP+HDP_augmenté*

Tab. IV.6 – HDP : Instances non-corrélées (Ecart relatif (%)).

IV.4.2 Les méthodes coopératives

Nous reprenons ici le même calcul de l'écart relatif de l'équation IV.1.

ϵ	n	S+aug	ADP	SMA	AGNES
0,5	50	0,03	0,10	0,07	0,02
0,5	100	0,22	0,53	0,78	0,10
0,5	150	0,50	1,09	3,90	0,30
0,5	200	3,06	3,51	11,39	0,65
0,5	250	9,77	7,35	25,45	1,33
0,5	300	84,46	14,85	57,27	2,44
0,5	400	227,96	41,93	171,74	5,91
0,5	500	519,10	80,52	1110,55	12,42
0,9	50	0,01	0,03	0,12	0,04
0,9	100	0,04	0,21	1,83	0,12
0,9	150	0,12	0,70	10,17	0,30
0,9	200	0,28	1,38	31,83	0,66
0,9	250	0,61	2,43	77,73	1,26
0,9	300	1,00	4,48	160,99	2,20
0,9	400	2,81	8,09	1161,58	4,87
0,9	500	8,23	23,50	1644,29	11,20

S+aug : *HDP+HDP_augmenté*

Tab. IV.7 – HDP : Instances non-corrélées (Temps de calcul (s.)).

Les tests numériques sont présentés dans les tableaux IV.10 et IV.11. On constate que, pour les problèmes non-corrélés, la méthode coopérative à des temps de calcul légèrement meilleure que le branch-and-bound. Pour les problèmes plus difficile (temps de calcul supérieure à 10 minutes), la méthode coopérative semble converger un peu plus vite vers la valeur optimale. Ces résultats viennent confirmer ce qui avaient été observé sur les instances de la littérature.

IV.5 Conception de nouvelles instances difficiles

Nous présentons dans cette partie les tests faits sur les instances aléatoires dérivées d'un problème en égalité, ainsi que celles basées sur la transformé en $\mathbb{Z}$ (voir partie I.4.3 page 20 et partie I.4.4 page 21). Les instances ont été résolues de manière exacte avec CPLEX 9.0.

Afin d'apprécier la difficulté de résolution de ces instances nous les avons comparées avec des instances non-corrélées, I_NC, et fortement corrélées, I_FC. Nous les avons engendrées de la manière suivante :
– le nombre de variable $n \in \{50, 100, 150, 200, 250, 300\}$,
– pour chaque n, 25 problèmes ont été générés,
– le nombre de contraintes $m = 5$,
– les coefficients sont tirés aléatoirement dans $I = [1, 10]$ et dans $I = [1, 1000]$,
– $\epsilon = 0, 5$ et
– $sc = 5$ (coefficient de corrélation).
Nous considérons des problèmes de taille raisonnable afin de conserver des temps de résolution exploitables qui permettent de mettre en évidence la pertinence des techniques que nous avons

ϵ	n	S+aug	ADP	SMA	AGNES
0,5	50	1,75	3,48	6,43	4,12
0,5	100	1,07	1,50	3,51	2,55
0,5	150	1,15	1,44	2,28	2,85
0,5	200	0,99	1,07	2,05	2,18
0,5	250	0,97	0,98	1,64	1,92
0,9	50	0,67	1,65	3,22	2,39
0,9	100	0,70	1,00	2,50	1,54
0,9	150	0,59	0,68	1,68	1,28
0,9	200	0,53	0,63	1,34	1,47
0,9	250	0,52	0,61	1,11	1,17

S+aug : *HDP+HDP_augmenté*

TAB. IV.8 – HDP : Instances corrélées (Ecart relatif (%)).

ϵ	n	S+aug	ADP	SMA	AGNES
0,5	50	0,64	0,06	0,04	0,03
0,5	100	23,26	0,41	0,36	0,18
0,5	150	30,05	1,41	1,41	0,62
0,5	200	42,48	3,53	3,81	1,46
0,5	250	80,90	7,15	8,43	2,81
0,9	50	0,15	0,03	0,04	0,02
0,9	100	16,15	0,23	0,45	0,10
0,9	150	17,07	0,73	2,09	0,33
0,9	200	25,42	1,83	6,18	0,77
0,9	250	66,33	3,66	16,64	1,51

S+aug : *HDP+HDP_augmenté*

TAB. IV.9 – HDP : Instances corrélées (Temps de calcul (s)).

développées.

IV.5.1 Instances engendrées à partir d'un problème en contrainte égalité

Afin de montrer l'intérêt de cette approche, nous avons engendré, selon le principe énoncé dans la partie I.4.3, des instances non-corrélées et fortement corrélées. Nous appellerons ces instances respectivement I_L_NC(λ) et I_L_FC(λ), avec λ paramétrant le multiplicateur de Lagrange $\Lambda = \lambda.(1...1)^t$. Lorsque le temps de calcul excède une heure, nous arrêtons la résolution.

Les tests sont présentés dans les tableaux IV.12 et IV.13. On constate que même pour des λ assez faibles (et non nuls), nos instances nécessitent des temps de résolution plus importants que pour le cas $\lambda = 0$. Plus ce coefficient sera pris grand, plus on se rapproche du problème en contrainte égalité et plus les instances seront, à priori, difficiles.

ϵ	n	Ecart (%)	t_BB (s)	t_MC (s)
0,5	50	0.00	0,35	0,33
0,5	100	0.00	0,81	0,99
0,5	150	0.00	328,73	329,60
0,5	200	-0.01	600,00	600,00
0,5	300	-0.01	600,00	600,00
0,5	400	0.00	600,00	600,00
0,5	500	-0.01	600,00	600,00
0,9	50	0.00	0,02	0,01
0,9	100	0.00	0,17	0,10
0,9	150	0.00	1,36	1,24
0,9	200	0.00	6,34	6,30
0,9	250	0.00	30,90	28,13
0,9	300	0.00	47,46	50,56
0,9	400	0.00	152,00	144,77
0,9	500	0.00	600,00	557,42

Ecart : Ecart entre la borne obtenu avec la méthode coopérative et celle du branch-and-bound
t_BB : temps de calcul du branch-and-bound
t_MC : temps de calcul de la méthode coopérative

TAB. IV.10 – Méthode coopérative : Instances non-corrélées.

ϵ	n	Ecart (%)	t_BB (s)	t_MC (s)
0,5	50	-0.02	600,00	600,00
0,5	100	-0.10	600,00	600,00
0,5	150	-0.05	600,00	600,00
0,9	50	0.00	600,00	600,00
0,9	100	0.00	600,00	600,00
0,9	150	0.00	600,00	600,00
0,9	200	0.00	600,00	600,00

Ecart : Ecart entre la borne obtenu avec la méthode coopérative et celle du branch-and-bound
t_BB : temps de calcul du branch-and-bound
t_MC : temps de calcul de la méthode coopérative

TAB. IV.11 – Méthode coopérative : Instances corrélées.

n	I_L_NC(0)	I_L_NC(5)	I_L_NC(10)	I_L_FC(0)	I_L_FC(5)	I_L_FC(10)
50	0,06	4,63	4,93	0,19	5,21	7,85
100	0,06	9,39	11,65	2,66	6,39	10,91
150	0,08	13,53	14,24	3,82	9,76	18,17
200	0,10	15,59	17,98	5,70	12,09	28,89
250	0,11	18,89	20,34	8,63	23,27	36,47
300	0,17	20,13	25,41	14,83	26,41	41,52

TAB. IV.12 – Temps de calcul (en s.) pour $I = [1, 10]$.

n	I_L_NC(0)	I_L_NC(1)	I_L_NC(5)	I_L_FC(0)	I_L_FC(1)
50	0,11	8,23	2641,65	101,65	>1h
100	0,87	288,21	>1h	2031,14	>1h
150	2,26	1069,99	>1h	>1h	>1h
200	4,58	3586,76	>1h	>1h	>1h
250	10,56	>1h	>1h	>1h	>1h
300	10,91	>1h	>1h	>1h	>1h

TAB. IV.13 – Temps de calcul (en s.) pour $I = [1, 1000]$.

IV.5.2 Instances engendrées à partir de l'analyse de la transformée en $\mathbb{Z}$

Dans cette partie, nous mettons en avant l'intérêt de cette approche. Nous nous sommes limités à 2 contraintes. Les coefficients δ_a, δ_b et p_0 ont été tirés aléatoirement dans l'intervalle $[1, 10]$. Ces problèmes demandant des temps de résolution plus importants, nous avons considéré des instances pour lesquelles $n \in \{50, 100, 150, 200\}$; nous nommerons ces instances I_Z.

Les temps de résolution ont été comparés à des instances de problèmes fortement corrélés avec $I = [1, 1000]$ et $m = 2$. Les résultats sont présentés dans le tableau IV.14. Nous arrêtons la résolution lorsque le temps de calcul dépasse une heure.

Les résultats sont présentés dans le tableau IV.14. On constate qu'à taille égale nous obtenons des temps de résolution beaucoup plus importants pour des problèmes engendrés par des techniques faisant appel à la transformée en $\mathbb{Z}$.

n	I_FC	I_Z
50	6,19	8,73
100	9,66	426,46
150	25,82	>1h
200	175,22	>1h

TAB. IV.14 – I_Z : Temps de calcul (en s.).

IV.6 Conclusion

Les approche heuristiques développées au chapitre précédent ont été testées pour différentes instances de la littérature ou des instances engendrées aléatoirement. De ces tests, il découle

que l'heuristique *HDP_S+HDP_augmenté* avec le classement de variables C_PR (profits réduits décroissants) permet d'avoir le meilleur compromis entre temps de calcul et qualité de la borne. L'heuristique *HDP_S* est basée sur la résolution de la relaxation surrogate du problème (MKP) par un algorithme de programmation dynamique modifié. *HDP_augmenté* est une heuristique construite à partir d'un algorithme glouton. Elle vient compléter *HDP_S* afin d'améliorer la borne obtenue par cette dernière. Dans tous les cas, *HDP_S+HDP_augmenté* permet d'avoir une meilleure approximation de la valeur optimale que les heuristiques proposées par Fréville et Plateau [FRE 94], Bertsimas et Demir [BER 02] et Hanafi, Fréville et El Abdellaoui [HAN 96].

La combinaison *HDP_S+HDP_augmenté* avec un algorithme de branch-and-bound nous permet de construire une méthode coopérative originale pour la résolution exacte du MKP. Les tests numériques effectués sur cette méthode sont assez prometteurs. Ils nous montrent que pour certaines instances nous avons des temps de résolution plus faibles que ceux obtenus avec une méthode classique de résolution telle que le branch-and-bound. De plus, dans les cas où nous avons été obligés d'arrêter la résolution du fait de temps de calcul trop important, la borne obtenue par notre méthode coopérative est meilleure que celle du branch-and-bound classique.

Nous avons aussi présenté les résultats préliminaires obtenus avec nos deux techniques pour engendrer aléatoirement des instances difficiles :
– instances dérivées de problèmes combinatoires en contrainte égalité et
– instances engendrées à partir de l'analyse de la transformée en $\mathbb{Z}$.
Les tests numériques montrent que ces techniques semblent conduire à des instances plus difficiles à résoudre que des instances non-corrélées ou fortement corrélées de même taille. Ces résultats nous permettent d'envisager d'approfondir l'étude sur ces approches.

Conclusions et Prospectives

Les travaux présentés dans ce mémoire portent sur la résolution du problème du sac à dos multidimensionnel. Nous avons proposé des méthodes heuristiques efficaces basées sur la programmation dynamique et la relaxation surrogate, ainsi qu'une méthode coopérative permettant de résoudre de manière exacte le problème MKP. Notre travail a porté aussi sur la nature de la difficulté en optimisation combinatoire et nous avons proposé deux techniques pour engendrer aléatoirement des instances difficiles.

L'approche heuristique proposée ici a donné lieu à un algorithme que nous avons testé pour différentes instances de la littérature ou engendrées aléatoirement. De ces tests, il découle que l'heuristique *HDP_S+HDP_augmenté* permet d'avoir le meilleur compromis entre temps de calcul et qualité de la borne. *HDP_S* est basée sur la résolution de la relaxation surrogate du problème (MKP) par un algorithme de programmation dynamique modifié afin de garantir la réalisabilité de la solution obtenue. Lors de ce processus de résolution certains états mis à l'écart sont conservés et sont utilisés par *HDP_augmenté*, qui, par une recherche au voisinage construite à partir d'un algorithme de type glouton, essaye d'améliorer la meilleure borne obtenue avec *HDP_S*. Dans tous les cas testés, cette heuristique permet d'avoir une meilleure approximation de la valeur optimale que les heuristiques proposées par Fréville et Plateau [FRE 94], Bertsimas et Demir [BER 02] et Hanafi, Fréville et El Abdellaoui [HAN 96]. La qualité de la borne obtenue se paye sur certaines instances de la littérature et engendrées aléatoirement par des temps de calcul légèrement supérieures.

Notre approche heuristique *HDP* laisse à l'utilisateur un large choix afin d'améliorer la borne obtenue. Sur ce point, nous avons proposé deux méthodes, à savoir, *HDP_augmenté*, qui engendre un temps de calcul supplémentaire relativement peu important, et *HDP_augmenté+BB_lim_T*, qui permet de se rapprocher un peu plus de la valeur optimale par une exploration limitée des états conservés via un algorithme de branch-and-bound limité.

Cette flexibilité de notre heuristique permet ainsi d'envisager une résolution exacte du problème traité. Pour cela on explore entièrement les éléments de la liste secondaire. L'emploi d'une méthode de branch-and-bound nous permet ainsi de construire une méthode coopérative originale (c'est-à-dire faisant coopérer la programmation dynamique et le branch-and-bound). Les tests numériques effectués sur cette méthode de résolution exacte sont assez prometteurs. Ils nous montrent que pour certaines instances nous avons des temps de résolution plus faibles que ceux obtenus avec une méthode classique de résolution telle que le branch-and-bound. De plus, dans les cas où nous avons été obligés d'arrêter la résolution du fait de temps de calcul trop important, la borne obtenue par notre méthode coopérative est meilleure que celle du branch-and-bound classique.

Notre travail a porté aussi sur la question de la difficulté des instances notamment dans le cas où celles-ci sont engendrées aléatoirement. Nous avons présenté deux nouvelles techniques pour

engendrer aléatoirement des instances difficiles :
– instances dérivées de problèmes combinatoires en contrainte égalité et
– instances engendrées à partir de l'analyse de la transformée en $\mathbb{Z}$.
Les tests numériques présentés montrent que ces techniques permettent d'obtenir des instances difficiles. Les problèmes dérivés d'un problème combinatoire en contrainte égalité rendent notamment plus ardue les instances à données non-corrélées et fortement corrélées.

Nous avons proposé différentes méthodes afin d'améliorer la borne obtenue par notre heuristique *HDP_S*, cependant nous comptons aussi étudier plusieurs autres approches. Par exemple, les méta-heuristiques se prêtent assez bien à la recherche au voisinage des éléments de notre liste secondaire (éléments mis à l'écart lors du processus de résolution de *HDP*). En effet, on peut, considérer *HDP* comme un processus de diversification et appliquer une recherche Tabou sur la liste secondaire, ou encore considérer les éléments de cette dernière comme une population de départ et appliquer un algorithme génétique.

Les résultats obtenus avec les deux techniques que nous proposons pour engendrer des instances difficiles sont assez prometteurs et montrent l'intérêt de ces techniques. Les interprétations faites sur les ratios profit sur poids proches, semblent engendrer des instances plus ardues à résoudre. On envisagera de généraliser l'approche de la transformée en $\mathbb{Z}$, appliquée aux problèmes à deux contraintes, aux problèmes à plus de deux contraintes. Les interprétations liées aux ratios ainsi que les techniques utilisées afin d'engendrer des problèmes difficiles seront approfondies.

Références bibliographiques

[**AHR 75**] J. H. Ahrens et G. Finke, "Merging and sorting applied to zero-one knapsack problem", *Operations Research*, vol. 23, 1975, pages 1099-1109.

[**AND 05**] K. Andersen, G. Cornuejols et Y. Li, "Reduce-and-Split Cuts : Improving the Performance of Mixed Integer Gomory Cuts", *Management Science*, vol. 51, 2005, pages 1720-1732.

[**BAL 80**] E. Balas et C. H. Martin, "Pivot and complement-a heuristics for 0-1 programming", *Management Science*, vol. 26, 1980, pages 86-96.

[**BAL 93**] E. Balas, S. Ceria et G. Cornuéjols, "A lift-and-project cutting plane algorithm for mixed 0-1 programs", *Mathematical Programming*, vol. 58, 1993.

[**BAL 08**] S. Balev, N. Yanev, A. Freville et R. Andonov, "A dynamic programming based reduction procedure for the multidimensional 0-1 knapsack problem", *European Journal of Operational Research*, vol. 186, 2008, pages 63-76.

[**BEI 03**] R. Beier et B. Vöcking, "Random knapsack in expected polynomial time", proceedings of *STOC '03 : Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, ACM, 2003, pages 232–241.

[**BEL 54**] R. Bellman, "Some applications of the theory of dynamic programming, a review", *Operations Research*, vol. 2, 1954, pages 275-288.

[**BEL 57**] R. Bellman, "Dynamic Programming", Princeton University Press, 1957.

[**BER 02**] D. Bertsimas et R. Demir, "An approximate dynamic-programming approach to multidimensional knapsack problem", *Management Science*, vol. 4, 2002, pages 550-565.

[**Boy**]

[**BOY 05**] V. Boyer, D. El Baz et M. Elkihel, "Efficient heuristic for the 0-1 multidimensional knapsack problem", 2nd International Workshop on Combinatorial Scientific Computing (CSC'05), 2005, Toulouse (France).

[**BOY 06**] V. Boyer, D. El Baz et M. Elkihel, "Heuristics for the multidimensional knapsack problem", 7ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF'06), 2006, Lille (France).

[**BOY 07**] V. Boyer, D. El Baz et M. Elkihel, "A heuristic for the 0-1 multidimensional knapsack problem", European Journal of Operational Research, 2007, à paraître.

[**BRA 74**] G. H. Bradley, H. P. L et W. L., "Coefficient reduction for inequalities in 0-1 variables", *Mathematical Programming*, vol. 7, 1974, pages 263-282.

[**CHA 93a**] I. Charon et O. Hudry, "La méthode du bruitage : application au problème du voyageur de commerce", ENST 93D003, 1993, France.

[**CHA 93b**] I. Charon et O. Hudry, "The noising method : a new method for combinatorial optimization", *Operations Research Letters*, vol. 133, 1993, pages 133-137.

[**CHU 98**] P. C. Chu et J. E. Beasley, "Genetic algorithm for the multidimensional knapsack problem", *Journal of Heuristics*, vol. 4, 1998, pages 63-86.

[**COO 90**] W. Cook, R. Kannan et A. J. Schrijver, "Chvatal closures for mixed integer programming problems", *Mathematical Programming*, vol. 47, 1990, pages 155-174.

[**COR 03**] G. Cornuejols, Y. Li et D. Vandenbussche, "K-Cuts : A Variation of Gomory Mixed Integer Cuts from the LP Tableau", *INFORMS Journal on Computing*, vol. 15, 2003, pages 385-396.

[**COR 07**] G. Cornuejols, "Revival of the Gomory Cuts in the 1990's", *Annals of Operations Research*, vol. 149, 2007, pages 63-66.

[**CRA 94**] Y. Crama et J. Mazzola, "On the Strength of Relaxations of Multidimensional Knapsack Problems", *INFOR*, vol. 32, 1994, pages 219-225.

[**DAN 48**] G. B. Dantzig, "Programming in a linear structure", United States Air France, Washington D.C., 1948.

[**DAN 57**] G. B. Dantzig, "Discrete variable extremum problems", *Operations Research*, vol. 5, 1957, pages 266-277.

[**DAN 61**] G. B. Dantzig, editor, *Linear programming and Extensions*, Princeton University Press, 1961.

[**DAN 67**] G. B. Dantzig et R. Van Slyke, "Generalized Upper Bounding Techniques", *Journal of Computer and Systems Sciences*, vol. 1, 1967, pages 213-226.

[**DRE 88**] A. Drexl, "A simulated annealing approach to the multiconstraint zero-one knapsack problem", *Computing*, vol. 40, 1988, pages 1-8.

[**DUE 89**] G. Dueck et J. Wrisching, "Threshold accepting algorithms for multi-constraint 0-1 knapsack problems", IBM Heidelberg Scientific Center, 1989, Allemagne.

[**DUE 90**] G. Dueck et T. Scheurer, "Threshold accepting : a general purpose optimization algorithm", *Journal of Computational Physics*, vol. 90, 1990, pages 161-175.

[**DYE 80**] H. E. Dyer, "Calculating surrogate constraints", *Mathematical Programming*, vol. 12, 1980, pages 225-278.

[**ELB 05a**] D. El Baz et M. Elkihel, "Load balancing methods and parallel dynamic programming algorithm using dominance technique applied to the 0-1 knapsack problem", *Journal of Parallel and Distributed Computing*, vol. 65, 2005, pages 74-84.

[**ELB 05b**] D. El Baz, M. Elkihel et L. Gely, "Résolution efficace de problèmes en variables 0-1 avec contraintes en égalité", *ROADEF'05, Presses Universitaires François Rabelais*, , 2005, pages 15-27.

[**ELK 84a**] M. Elkihel, "Programmation dynamique et rotations de contraintes pour les problèmes d'optimisation entière", Thèse de Doctorat, 1984, Université des Sciences et Techniques de Lille (France).

[**ELK 84b**] M. Elkihel et G. Plateau, "A hybrid method for the 0-1 knapsack problem", 9ème Symposium de Recherche Opérationnelle, 1984, Osnabruck (Allemagne).

[**ELK 86a**] M. Elkihel et G. Plateau, "Constraints agregating and rotation for integer linear programming problems", Congrés EURO VIII, 1986, Lisbone (Portugal).

[**ELK 86b**] M. Elkihel et G. Plateau, "Inequality rotation for 0-1 linear programming problems", 11ème Symposium de Recherche Opérationnelle, 1986, Darmstadt (Allemagne).

[**ELK 96**] M. Elkihel, G. Authié et F. Viader, "A mixed approach to 0-1 problems", Advanced Summer Institute (ASI'96), 1996, Toulouse (France).

[**ELK 02**] M. Elkihel, G. Authié et F. Viader, "An efficient hybrid dynamic algorithm to solve 0-1 knapsack problems", International Symposium on Combinatorial Optimization (CO'2002), 2002, Paris (France).

[**FAY 75**] D. Fayard et G. Plateau, "Resolution of the 0-1 knapsack problem : Comparaison of methods", *Mathematical Programming*, vol. 8, 1975, pages 272-307.

[**FAY 82**] D. Fayard et G. Plateau, "An algorithm for the solution of the 0-1 knapsack problem", *Computing*, vol. 28, 1982, pages 269-287.

[**FIS 81**] M. L. Fisher, "The Lagrangean Relaxation Method for Solving Integer Programming Problems", *Management Science*, vol. 27, 1981, pages 1-18.

[**FRE 86**] A. Freville et G. Plateau, "Heuristics and reduction methods for multiple constraints 0-1 linear programming problems", *European Journal of Operational Research*, vol. 24, 1986, pages 206-215.

[**FRE 90**] A. Freville et G. Plateau, "Hard 0-1 multiknapsack test problems for size reduction methods", *Investigation Operativa*, vol. 1, 1990, pages 251-270.

[**FRE 91**] A. Freville, L. A. N. Lorena et G. Plateau, "Efficient subgradient algorithms for the 0-1 multiknapsack Lagrangean and surrogate duals", Rapport de recherche L.I.P.N. 90-13, 1991, Université de Paris-Nord (France).

[**FRE 93**] A. Freville et G. Plateau, "An Exact Search for the Solution of the Surrogate Dual of the 0-1 Bidimensional Knapsack Problem", *European Journal of Operational Research*, vol. 68, 1993, pages 413-421.

[**FRE 94**] A. Freville et G. Plateau, "An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem", *Discrete Applied Mathematics*, vol. 49, 1994, pages 189-212.

[**FRE 04**] A. Freville, "The multidimensional 0-1 knapsack problem : An overview", *European Journal of Operational Research*, vol. 155, 2004, pages 1-21.

[**GAV 85**] B. Gavish et H. Pirkul, "Efficient algorithms for solving multiconstraint 0-1 knapsack problems to optimality", *Mathematical Programming*, vol. 31, 1985, pages 78-205.

[**GEO 74**] A. Geoffrion, "The Lagrangean Relaxation for Integer Programming", *Mathematical Programming Study*, vol. 2, 1974, pages 82-114.

[**GIL 61**] P. C. Gilmore et R. E. Gomory, "A Linear Programming Approach to the Cutting-Stock Problem", *Operations Research*, vol. 9, 1961, pages 849-859.

[**GIL 62**] P. C. Gilmore et R. E. Gomory, "A Linear Programming Approach to the Cutting-Stock Problem - Part II", *Operations Research*, vol. 11, 1962, pages 863-888.

[**GIL 65**] P. C. Gilmore et R. E. Gomory, "Multistage Cutting Stock Problems of Two and More Dimensions", *Operations Research*, vol. 13, 1965, pages 94-120.

[**GIL 66**] P. C. Gilmore et E. Gomory, "The theory and computation of knapsack functions", *Operations Research*, vol. 17, 1966, pages 437-454.

[**GLO 65**] F. Glover, "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem", *Operations Research*, vol. 13, 1965, pages 879-919.

[**GLO 68**] F. Glover, "Surrogate constraints", *Operations Research*, vol. 16, 1968, pages 741-749.

[**GLO 75**] F. Glover, "Surrogate constraint duality in mathematical programming", *Operations Research*, vol. 23, 1975, pages 434-453.

[**GLO 77**] F. Glover, "Heuristics for integer programming using surrogate constraints", *Decision Sciences*, vol. 8, 1977, pages 156-166.

[**GLO 89**] F. Glover, "Tabu search : Part i", *ORSA journal on Computing*, vol. 1, 1989, pages 190-209.

[**GLO 90**] F. Glover, "Tabu search : Part ii", *ORSA journal on Computing*, vol. 2, 1990, pages 4-32.

[**GLO 97**] F. Glover et M. Laguna, *Tabu search*, Kluwer Academic Publishers, 1997.

[**GOL 89**] D. E. Goldberg, *Genetics algorithms in search, optimization and machine learning*, Addison-Wesley, 1989.

[**GOM 58**] R. E. Gomory, "Outline of an algorithm for integer solutions to linear programs", *Bulletin of the American Mathematical Society*, vol. 64, 1958, pages 275-278.

[**GOM 63**] R. E. Gomory, "*Recent Advances in Mathematical Programming*", Chapitre An algorithm for integer solution to linear programs, pages 269-302, McGraw-Hill, 1963.

[**GRE 70**] H. Greenberg et W. Pierskella, "Surrogate mathematical programs", *Operations Research*, vol. 18, 1970, pages 924-939.

[**HAM 75**] P. L. Hammer, M. W. Padberg et U. N. Peled, "Constraint pairing in integer programming", vol. 13, 1975, pages 68-81.

[**HAN 96**] S. Hanafi, A. Freville et A. El Abdellaoui, "*Meta-Heuristics : Theory and Application*", Chapitre Comparaison of heuristics for the 0-1 multidimensional knapsack problem, pages 446-465, Kluwer Academic, 1996.

[**HAN 98**] S. Hanafi et A. Freville, "An efficient tabu search approach for the 0-1 muldimensional knapsack problem", *European Journal of Operational Research*, vol. 106, 1998, pages 659-675.

[**HAN 00**] S. Hanafi et A. Freville, "Résolution du Dual Composite du Sac à Dos Bidimensionnel en variables 0-1 par une méthode de Branch-and-Bound", proceedings of *Francoro*, 2000.

[**HAN 07**] S. Hanafi et F. Glover, "Exploiting nested inequalities and surrogate constraints", *European Journal of Operational Research*, vol. 179, 2007, pages 50-63.

[**HIL 69**] F. S. Hillier, "Efficient heuristics procedures for integer linear programming with an interior", *Operations Research*, vol. 17, 1969, pages 600-637.

[**HOR 74**] E. Horowitz et S. Sahni, "Computing partitions with applications to the knapsack problem", *Journal of ACM*, vol. 21, 1974, pages 277-292.

[**IBA 75**] O. H. Ibarra et C. E. Kim, "Fast approximation algorithms for the knapsack and sum of subset problems", *Journal of ACM*, vol. 22, 1975, pages 463-468.

[**ING 73**] G. P. Ingargiola et J. F. Korsh, "Reduction algorithm for zero-one single knapsack problems", *Management Science*, vol. 20, 1973, pages 460-463.

[**KEL 04**] H. Kellerer, U. Pferschy et D. Pisinger, *Knapsack Problems*, Springer, 2004.

[**KIA 71**] F. Kianfar, "Stronger inequalities for the 0-1 interger programming using knapsack functions", *Operations Research*, vol. 25, 1971, pages 1374-1392.

[**KIA 76**] F. Kianfar, "Stronger inequalities for the 0-1 interger programming computational refinements", vol. 24, 1976, pages 581-585.

[**KIR 83**] S. Kirkpatrick, Jr. Gelatt et M. P. Vecchi, "Optimization by simulated annealing", *Science*, vol. 220, 1983, pages 671-680.

[**KO 93**] I. Ko, "Using AI techniques and learning to solve multi-level knapsack problems", Thèse de Doctorat, 1993, Université du Colorado (Etats-Unis).

[**KOL 67**] P. J. Kolesar, "A branch and bound algorithm for the knapsack problem", *Management Science*, vol. 13, 1967, pages 723-735.

[**LAN 60**] A. H. Land et A. G. Doig, "An automatic method for solving discrete programming problems", *Econometrica*, vol. 28, 1960, pages 497-520.

[**LAS 03a**] J. B. Lasserre, "Generating functions and duality for integer programs", *Discrete Optimization*, vol. 1, 2003, pages 167-187.

[**LAS 03b**] J. B. Lasserre, "*Optimization : Structure and Applications*", Kluwer Academic Publishers, 2003.

[**LAS 04a**] J. B. Lasserre, "A discrete Farkas lemma", *Discrete Optimization*, vol. 1, 2004, pages 67-75.

[**LAS 04b**] J. B. Lasserre, "The Integer Hull of a Convex Rational Polytope", *Discrete & Computational Geometry*, vol. 32, 2004, pages 129-139.

[**LAS 04c**] J. B. Lasserre, "Integer programming duality", *Proceedings of Symposia in Applied Mathematics*, vol. 61, 2004, pages 67-76.

[**LOO 92**] W. Loots et T. H. C. Smith, "A parallel three phase sorting procedure for a k-dimensional hypercube and a transputer implementation", *Parallel Computing*, vol. 18, no. 3, 1992, pages 335-344.

[**LOU 79**] R. Loulou et E. Michaelides, "New greedy-like heuristics for the multidimensionnal 0-1 knapsack problem", *Operations Research*, vol. 27, 1979, pages 1101-1114.

[**MAG 84**] M. J. Magazine et O. Oguz, "A heuristics algorithm for the multidimensional zero-one knapsack problem", *European Journal of Operational Research*, vol. 16, 1984, pages 319-326.

[**MAR 77**] S. Martello et P. Toth, "An upper bound for the zero-one knapsack problem and a branch and bound algorithm", *European Journal of Operational Research*, vol. 1, 1977, pages 169-175.

[**MAR 82**] S. Martello et P. Toth, "A mixed algorithm for the Subset-Sum Problem", EURO V. TIMS, 1982, Lausanne (France).

[**MAR 90**] S. Martello et P. Toth, editors, *Knapsack Problems - Algorithms and Computer Implementations*, Wiley & Sons, 1990.

[**MAR 97**] S. Martello et P. Toth, "Upper bounds and algorithms for hard 0-1 knapsack problems", *Operations Research*, vol. 45, 1997, pages 768-778.

[**MCC 74**] B. A. McCarl, A. Kochenberg et F. P. Wynmann, "A heuristic for general integer programming", *Decision Sciences*, vol. 5, 1974, pages 36-44.

[**NEM 88**] L. Nemhauser et A. Wolsey, *Integer and combinational optimization*, Wiley Interscience, 1988.

[**NEM 90**] L. Nemhauser et A. Wolsey, "A recursive procedure to generate all cuts for 0-1 mixed integer programs", *Mathematical Programming*, vol. 46, 1990, pages 379-390.

[**OHL 93**] M. Ohlsson, C. Peterson et B. Soderberg, "Neural networks for optimization problems with inequality constraints : the knapsack problem", *Neural Computation*, vol. 5, 1993, pages 331-339.

[**OSO 02**] M. Osorio, F. Glover et P. Hammer, "Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions", *Annals of Operations Research*, vol. 117, 2002, pages 71-93.

[**PAN 93**] S. N. N. Pandit et M. Ravi Kumar, "A lexicographic search for strongly correlated 0-1 knapsack problems", *Operations Research*, vol. 30, 1993, pages 97-116.

[**PEN 94**] M. Penn, D. Hasson et M. Avriel, "Solving the 0-1 proportional knapsack problem by sampling", *Journal of Optimization Theory and Applications*, vol. 80, 1994, pages 289-297.

[**PET 67**] C. C. Petersen, "Computational experience with variants of the Balas algorithm applied to the selection of R&D projects", *Management Science*, vol. 13, 1967, pages 736-750.

[**PIR 87**] H. Pirkul, "A heuristic solution procedure for the multiconstraint zero-one knapsack problem", *Naval Research Logistics*, vol. 34, 1987, pages 161-172.

[**PIS 95**] D. Pisinger, "A minimal algorithm for the Bounded Knapsack Problem", proceedings of *Integer Programming and Combinatorial Optimization, Fourth IPCO conference*, J. Balas E. & Clausen, editor, vol. 920 of *Lecture Notes in Computer Science*, Springer, Berlin, 1995, pages 95-109.

[**PLA 85**] G. Plateau et M. Elkihel, "A hybrid method for the 0-1 knapsack problem", *Methods of Operations Research*, vol. 49, 1985, pages 277-293.

[**SAH 75**] S. Sahni, "Approximate Algorithms for the Knapsack Problem", *Journal of ACM*, vol. 22, 1975, pages 115-124.

[**SEN 68**] S. Senju et Y. Toyoda, "An approach to linear programming with 0-1 variables", *Management Science*, vol. 15, 1968, pages 196-207.

[**SHI 79**] W. Shi, "A branch and bound method for the multiconstraint zero one knapsack problem", *Journal of Operational Research Society*, vol. 30, 1979, pages 369-378.

[**TOT 80**] P. Toth, "Dynamic programming algorithm for the zero-one knapsack problem", *Computing*, vol. 25, 1980, pages 29-45.

[**TOY 75**] Y. Toyoda, "A simplified algorithm to obtain approximate solutions to zero-one programming problems", *Management Science*, vol. 21, 1975, pages 1417-1427.

[**VAS 01a**] M. Vasquez et J. K. Hao, "A hybrid approach for the 0-1 multidimensional knapsack problem", IJCAI-01, 2001.

[**VAS 01b**] M. Vasquez et J. K. Hao, "Une approche hybride pour le problème de sac-à-dos multidimensionnel", *RAIRO Operational Research*, vol. 35, 2001, pages 415-438.

[**VIA 98**] F. Viader, "Méthodes de programmation dynamique et de recherche arborescente pour l'optimisation combinatoire : utilisation conjointe des deux approches et parallélisation d'algorithmes", Thèse de Doctorat, 1998, LAAS-CNRS Toulouse (France).

[**WAL 76**] S. Walukiewicz et I. Kaliszewski, "Tighter equivalent formulations of integer programming problems", Survey of Mathematical Programming Proceeding, 1976.

[**WEI 67**] H. M. Weingartner et D. N. Ness, "Methods for the solution of the multi-dimensional 0/1 knapsack problem", *Operations Research*, vol. 15, 1967, pages 83-103.

[**WIL 74**] H. P. Williams, "Experiments in the formulation of integer programming problem", *Mathematical Programming Study*, vol. 2, 1974, pages 180-190.

[**WIL 06**] C. Wilbaut, S. Hanafi, A. Freville et S. Balev, "Tabu search : global intensification using dynamic programming", *Control and Cybernetics*, vol. 35, 2006, pages 579-598.

[**ZAN 77**] S. H. Zanackis, "Heuristics 0-1 linear programming : An experimental comparison of three methods", *Management Science*, vol. 24, 1977, pages 91-104.

[FAY 75] [FAY 82] [WIL 06] [GIL 66] [BAL 08] [KEL 04][LAS 04b] [LAS 04a] [ELK 96]