Propagation de coûts réduits et énumération implicite pour le problème du sac à dos multidimensionnel en 0-1

Sylvain Boussier, Yannick Vimont, Michel Vasquez

Propagation de coûts réduits et énumération implicite pour le problème du sac à dos multidimensionnel en 0-1

Sylvain Boussier **Yannick Vimont** **Michel Vasquez**

LGI2P, Parc Scientifique Georges Besse, 30035 Nimes Cedex 1, France
{Sylvain.Boussier, Yannick.Vimont, Michel.Vasquez}@ema.fr

Résumé

Dans de précédents travaux, nous avons proposé une heuristique de fixation de variables pour le problème du sac à dos multidimensionnel en variables binaires (01MDK). Cette approche tient compte des coordonnées des optima fractionnaires calculés dans des hyperplans contenant l'optimum binaire. Ce premier algorithme a obtenu les meilleurs minorants sur les instances de P.C Chu et J.E Beasley qui font partie de la OR-Library. Bien qu'intéressante du point de vue qualitatif, cette méthode ne prouve pas l'optimalité des solutions qu'elle trouve et peut fixer des variables à une valeur non optimale. Nous proposons ici une méthode d'énumération implicite basée sur une analyse des coûts réduits, qui tend à fixer exactement les variables hors base. La prise en compte d'une contrainte de distance au gap, par propagation des coûts réduits, ainsi qu'une méthode efficace de branchements nous permettent d'élaguer significativement l'arbre de recherche. Du point de vue expérimental, ces travaux apportent deux contributions principales : (1) l'obtention de plusieurs nouvelles preuves d'optimalité pour des instances difficiles de la OR-Library et (2) la réduction de certaines instances pour lesquelles l'algorithme n'a pas trouvé la solution optimale.

Abstract

In a previous work we proposed a variable fixing heuristic for the 0-1 Multidimensional knapsack problem (01MDK). This approach uses fractional optima calculated in hyperplanes containing the binary optimum. This algorithm obtained best lower bounds on the P.C Chu and J.E Beasley instances of the OR-Library. Though very attractive in terms of results, this method does not prove the optimality of the solutions and may fix variables to a non-optimal value. We propose here, an implicit enumeration based on a reduced costs analysis which tends to fix non-basic variables to their exact values. Taking into account a gap constraint by propagating reduced costs and an efficient branching method enable us to significantly prune the search tree. Experimentally, our work provides two main contributions : (1) the obtention of several new optimal solutions on hard instances of the OR-Library and (2) the reduction of some instances our algorithm did not find the optimal solution.

1 Introduction

Le problème du sac à dos multidimensionnel en variables binaires (01MDK pour *01 Multidimensionnal Knapsack Problem*) est un problème connu d'optimisation qui appartient à la famille des problèmes NP-difficiles. Il peut être défini de la manière suivante :

$$(P)\begin{cases}\text{Maximiser } c \cdot x \text{ sujet à,}\\ A \cdot x \leq b \text{ et } x \in \{0,1\}^n.\end{cases}$$

où $c \in \mathbb{N}^n$, $A \in \mathbb{N}^{m \times n}$ et $b \in \mathbb{N}^m$. Les composantes binaires x_j de x sont des variables de décision : $x_j = 1$ si l'objet j est sélectionné, 0 sinon. c_j est le profit associé à l'objet j et A_{ij} est le coût (en terme de ressource i) de la sélection de l'objet j. b_i est la quantité disponible de ressource i.

Ce problème est dérivé du problème original du sac à dos unidimensionnel ayant une seule contrainte de capacité. Son appellation provient de sa formulation originale qui décrit le problème d'un randonneur ayant à choisir un sous-ensemble d'objets à emporter dans son sac à dos sous une contrainte de capacité. Le cas multidimensionnel est modélisé dans de nombreux problèmes pratiques et de nombreux domaines tels que les

domaines du transport, de l'industrie, des télécommunications, de l'informatique, etc. Son intérêt pratique ainsi que sa complexité en ont fait l'objet de multiples travaux de recherches.

Dans de précédents travaux [6], nous avons proposé une heuristique de fixation de variables pour le 01MDK. Ce premier algorithme a obtenu les meilleurs minorants sur les instances de P.C Chu et J.E Beasley qui font partie de la OR-Library [1]. Bien qu'intéressante du point de vue qualitatif, cette méthode ne prouve pas l'optimalité des solutions qu'elle trouve et peut fixer des variables à une valeur non optimale.

Oliva et al. [4], proposent une méthode originale utilisant une contrainte basée sur les coûts réduits des variables hors base et des variables d'écarts. Leur algorithme est un Branch & Bound auquel, à chaque noeud de l'arbre de recherche est générée une nouvelle contrainte basée sur les coûts réduits à l'optimum du problème relaxé. Cette contrainte est ajoutée au sous problème courant dans le modèle Programmation Par Contrainte (PPC) afin de réduire les bornes des variables.

En s'inspirant de ces travaux, nous proposons une méthode d'énumération implicite basée sur une même analyse des coûts réduits, qui tend à fixer exactement les variables hors base.

Nous expliquerons le principe général de notre algorithme en section 2, puis nous détaillerons la contrainte des coûts réduits que nous utilisons en section 3. Nous détaillerons la méthode d'énumération implicite ainsi que l'algorithme de fixation de variables en section 4. Les section 5 et 6 font respectivement référence à une amélioration de l'algorithme de fixation de variables et à un algorithme de propagation des coûts réduits que nous avons intégrés à la procédure. Les résultats expérimentaux sont exposés en section 7.

2 Principe général de notre algorithme

L'idée de base de nos travaux a été d'utiliser la contrainte des coûts réduits mentionnée par Oliva et al. [4] et de l'intégrer différemment dans notre procédure d'énumération. Premièrement, nous appliquons la réduction de bornes seulement aux variables hors base (au lieu d'englober les variables d'écart) dans une approche directe de fixation de variables et deuxièmement, nous propageons cette contrainte implicitement aux noeuds parents de l'affectation partielle courante (et non pas explicitement en l'ajoutant au modèle). De plus nous avons cherché à affiner les branchements à chaque noeud de l'arbre de recherche. La contrainte des coûts réduits (section 3) nécessite la connaissance d'un minorant et d'une borne supérieure du problème. Elle prend en compte la distance au gap des coûts réduits des variables hors base à l'optimum. RL^{tabou} [5, 6] est une approche hybride très performante qui combine la programmation linéaire et la recherche locale tabou. Son principe général est d'utiliser la méthode du simplexe pour obtenir des points continus autour desquels on lance un algorithme tabou. Elle a obtenu les meilleurs minorants sur les instances à 500 variables de P.C Chu et J.E Beasley qui font partie de la OR-Library. L'idée a été d'utiliser RL^{tabou} afin d'obtenir les meilleurs minorants possibles pour les instances sur lesquelles nous expérimentons notre algorithme. Notre démarche s'articule autour des trois points suivants :

- Utiliser RL^{tabou} pour obtenir de bons minorants.
- Utiliser la contrainte des coûts réduits pour fixer des variables hors base.
- Propager les coûts réduits du noeud courant aux contraintes des noeuds parents afin de couper certains noeuds de l'arbre de recherche.

Par ailleurs, nous avons apporté des améliorations à l'algorithme de fixation de variables et à la méthode de branchement qui rendent notre énumération implicite particulièrement efficace, elle sont décrites section 4.3 et 5.

3 La contrainte des coûts réduits

Considérons la relaxation linéaire ($\bar{0}1$MDK) du problème établie comme suit :

$$\begin{aligned} \text{Max} \quad & z = \sum_{j=1}^{n} c_j x_j \\ \text{s.à.} \quad & Ax \leq b \\ & x \in [0,1]^n \end{aligned} \tag{1}$$

Sa résolution par la méthode du simplexe nécessite qu'il soit exprimé sous forme standard : l'inégalité (1) doit être transformée en égalité en introduisant des variables appelées *variables d'écart* à valeurs positives ou nulles, ce qui nous donne :

$$\begin{aligned} \text{Max} \quad & z = \sum_{j=1}^{n} c_j x_j \\ \text{s.à.} \quad & Ax + Ss = b \\ & x \in [0,1]^n, s \geq 0 \end{aligned}$$

où s est le vecteur des m variables d'écart. Soit $(\bar{x}, \bar{s})$ une solution optimale du programme linéaire $\bar{0}1$MDK et BS sa valeur. Soit $(\bar{c}, \bar{u})$ le vecteur des coûts réduits correspondant aux variables (x, s) pour la solution de base $(\bar{x}, \bar{s})$. Le $\bar{0}1$MDK peut également être écrit de

manière totalement équivalente comme suit :

$$\begin{aligned} \text{Max} \quad & BS + \sum_{j \in N^-} \bar{c}_j x_j - \sum_{j \in N^+} \bar{c}_j (1 - x_j) + \sum_{j \in M} \bar{u}_i s_i \\ \text{s.à.} \quad & \bar{A}x + \bar{S}s = \bar{b} \\ & x \in [0,1]^n, s \geq 0 \end{aligned}$$

où $\bar{A}$, $\bar{S}$, $\bar{b}$ sont les valeurs correspondantes à la base associée à $(\bar{x}, \bar{s})$, N^- représente l'ensemble des variables hors base à leur borne inférieure et N^+ l'ensemble des variables hors base à leur borne supérieure. Supposons que nous connaissions une borne inférieure $BI \in \mathbb{N}$ du programme linéaire en nombres entiers original, alors chaque solution meilleure que BI doit satisfaire la contrainte suivante :

$$BS + \sum_{j \in N^-} \bar{c}_j x_j - \sum_{j \in N^+} \bar{c}_j (1 - x_j) + \sum_{j \in M} \bar{u}_i s_i \geq BI$$

$$- \sum_{j \in N^-} \bar{c}_j x_j + \sum_{j \in N^+} \bar{c}_j (1 - x_j) - \sum_{j \in M} \bar{u}_i s_i \leq BS - BI$$

Dans notre cas, nous ne tenons pas compte des variables d'écart. En effet, à l'optimum, les variables d'écart sont positives et leur coût réduit associé est négatif, la somme $\sum_{j \in M} \bar{u}_i s_i$ est donc toujours négative. Nous pouvons donc enlever les variables d'écarts de l'inégalité au dessus sans perte de sens. De plus, nous savons qu'à l'optimalité du programme linéaire les variables hors base de coût réduit négatif sont égales à leur borne inférieure ($\bar{c}_j < 0 \Rightarrow x_j \in N^-$) et les variables hors base de coût réduit positif sont égales à leur borne supérieure ($\bar{c}_j > 0 \Rightarrow x_j \in N^+$), nous pouvons donc considérer les valeurs absolues des coûts réduits au lieu des coûts réduits eux-mêmes. Par ce procédé, nous obtenons donc la contrainte des coûts réduits :

$$\sum_{j \in N^-} |\bar{c}_j| x_j + \sum_{j \in N^+} |\bar{c}_j| (1 - x_j) \leq BS - BI \qquad (2)$$

4 Procédure d'énumération implicite

L'algorithme suit les principales stratégies d'un Branch & Bound : Brancher sur une variable selon un critère spécifique, la fixer à une valeur, évaluer le sous problème correspondant : Si la borne supérieure du sous problème est inférieure à un minorant donné ou si la configuration partielle courante ne satisfait pas les contraintes du problème, alors l'algorithme effectue un retour en arrière (backtrack) et crée une nouvelle branche dans laquelle la variable est fixée à la valeur opposée. Dans l'autre cas, le processus et réitéré et la variable suivante est fixée à une valeur donnée etc. Le processus s'arrête lorsque toutes les solutions ont été "implicitement" explorées. Nous détaillons dans cette section les différentes phases de notre algorithme d'énumération : L'évaluation des configurations partielles, l'algorithme de fixation de variables ainsi que les différentes phases de branchement.

4.1 Evaluation

Considérons S_t une configuration partielle (ou noeud de l'arbre de recherche), t étant sa profondeur dans l'arbre. Une configuration partielle S_t est représentée par un vecteur $(x_1, \ldots, x_n)$ où $x_i \in \{0, 1, *\}^n$ et $x_i = *$ si x_i est une variable libre. S_t est composée du sous-ensemble $F(S_t)$ des indices des variables fixées à zéro ou un et du sous-ensemble $L(S_t)$ des indices des variables libres. $F(S_t)$ est partitionné en deux sous-ensembles $F_0(S_t)$ et $F_1(S_t)$ contenant respectivement les indices des variables fixées à zéro et ceux des variables fixées à un. Par exemple, une configuration (ou affectation) partielle pourrait être la suivante :

$$S_t = (\overbrace{0,0,0}^{F_0(S_t)}, \overbrace{*,*,*}^{L(S_t)}, \overbrace{1,1,1}^{F_1(S_t)})$$

L'évaluation de S_t consiste à résoudre un sous-problème où les variables de décision sont les variables libres. En prenant en compte les variables fixées à zéro ou un dans le modèle, le sous problème ($\bar{0}\bar{1}\text{MDK}_{S_t}$) est établi de la manière suivante :

$$\begin{aligned} \text{Max} \quad & z = \sum_{j \in L(S_t)} c_j x_j + \sum_{j \in F_1(S_t)} c_j \\ \text{s.à.} \quad & \sum_{j \in L(S_t)} a_{ij} x_j \leq b_i - \sum_{j \in F_1(S_t)} a_{ij} \quad i = 1, \ldots, m \\ & x_j \in [0,1] \qquad\qquad\qquad j \in L(S_t) \end{aligned}$$

En effet, nous ajoutons une constante positive à la fonction objectif qui correspond à la somme des coûts réduits des variables fixées à un ($\sum_{j \in F_1(S_t)} c_j$) et nous soustrayons le coût de ces variables au membre de droite de chaque contrainte $i = 1, \ldots, m$ où elles apparaissent. L'évaluation de S_t correspond à la valeur optimale (borne supérieure) de $\bar{0}\bar{1}\text{MDK}_{S_t}$ plus la somme des coûts des variables fixées à un. Soit $\bar{x}^{S_t}$ la solution optimale du programme linéaire $\bar{0}\bar{1}\text{MDK}_{S_t}$ et $\bar{z}^{S_t}$ sa valeur, l'évaluation de S_t est définie comme suit :

$$BS(S_t) = \sum_{j \in F_1(S_t)} c_j + \bar{z}^{S_t} \qquad (3)$$

4.2 Algorithme de fixation de variables

La contrainte des coûts réduits (2) nous permet de fixer certaines variables hors base à zéro ou un. En

effet, si nous isolons une variable x_j hors base à l'optimum de $0\bar{1}\text{MDK}_{S_t}$ et considérons toutes les autres variables hors base à leur borne, c'est à dire à zéro pour les variables de coût réduit négatif et à un pour les variables de coût réduit positif, la contrainte (2) nous permet de déduire les relations suivantes :

$$\bar{x}_j^{S_t} = 1 \qquad \Rightarrow x_j \geq \lceil 1 - \frac{BS(S_t) - BI}{|\bar{c}_j^{S_t}|} \rceil \qquad (4)$$

$$\bar{x}_j^{S_t} = 0 \qquad \Rightarrow x_j \leq \lfloor \frac{BS(S_t) - BI}{|\bar{c}_j^{S_t}|} \rfloor \qquad (5)$$

où $\bar{x}_j^{S_t}$ est la valeur de x_j à l'optimum de $0\bar{1}\text{MDK}_{S_t}$ et $\bar{c}_j^{S_t}$ est son coût réduit correspondant. En utilisant les relations (4) et (5) nous établissons l'algorithme de fixation de variable suivant :

Pour chaque variable x_j hors base à l'optimum de $0\bar{1}MDK_{S_t}$, si $|\bar{c}_j^{S_t}| > BS(S_t) - BI$: Fixer x_j à zéro (resp. un) si et seulement si $\bar{x}_j^{S_t}$ est égale à zéro (resp. un).

Ainsi chaque configuration partielle est évaluée en résolvant le programme linéaire 01MDK_{S_t} mentionné au dessus et nous fournit la borne supérieure $BS(S_t)$, puis en considérant les coûts réduits des variables hors base à l'optimum, nous fixons les variables hors base selon l'algorithme de fixation de variable.

4.3 Phase de branchements

Partant de l'hypothèse que chaque solution du 01MDK satisfait la propriété : $1 \cdot x = k$ où $k \in \mathbb{N}$ correspond au nombre d'objets sélectionnés, l'idée est de partitionner l'espace de recherche en plusieurs sous-problèmes $01\text{MDK}(k)$ avec différentes valeurs de k possibles :

$$01\text{MDK}(k) \begin{cases} \text{Maximiser } c \cdot x \text{ sujet à,} \\ A \cdot x \leq b \text{ and } x \in [0,1]^n, \\ 1 \cdot x = k \in \mathbb{N} \end{cases}$$

Dans nos précédents travaux [5], nous soulignons le fait que le nombre d'objets k peut facilement être encadré par deux valeurs k_{min} et k_{max}. Soit z^* une borne inférieure du problème, alors k_{min} est l'entier le plus proche supérieur ou égal à la solution optimale du programme linéaire suivant :

$$\begin{cases} \text{Minimiser } \sum_i x_i \text{ sujet à,} \\ A \cdot x \leq b \text{ et } c \cdot x \geq z^* \text{ et } x \in [0,1]^n, \end{cases}$$

et k_{max} est le plus proche entier inférieur ou égal à la solution optimale du programme linéaire suivant :

$$\begin{cases} \text{Maximiser } \sum_i x_i \text{ sujet à,} \\ A \cdot x \leq b \text{ et } c \cdot x \geq z^* \text{ et } x \in [0,1]^n, \end{cases}$$

En effet, soit z^*_{min} la valeur optimale du premier problème et z^*_{max} la valeur optimale du second problème. Si nous prenons moins de $\lceil z^*_{min} \rceil$ objets alors la contrainte $c \cdot x \geq z^*$ ne sera pas satisfaite et si nous prenons plus que $\lfloor z^*_{max} \rfloor$ objets, l'une au moins des contraintes $A \cdot x \leq b$ ne sera plus vérifiée.

La première phase de notre algorithme est donc de calculer les $(k_{max} - k_{min} + 1)$ différentes valeurs de k en utilisant une borne inférieure fournie par RL^{tabou}. Nous créons ensuite le sous problème $01\text{MDK}(k)$ avec un espace de solutions réalisables correspondant $S[k]$ pour chaque valeur de k au noeud racine.

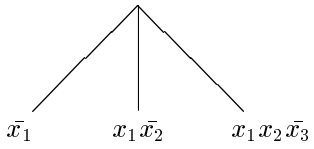
Nous procédons ensuite à l'énumération de chaque branche correspondant à chaque hyperplan $(1 \cdot x = k$ $k \in \mathbb{N})$. Pour chaque configuration partielle S_t nous résolvons le $0\bar{1}\text{MDK}_{S_t}(k)$ qui est la relaxation linéaire du $01\text{MDK}(k)$ dans le but d'obtenir la borne supérieure $BS(S_t)$ et les coûts réduits des variables hors base. Puisque nous avons ajouté une contrainte au problème original $0\bar{1}\text{MDK}_{S_t}$, nous devons la prendre en compte pour l'évaluation des configurations partielles. Nous définissons donc le $0\bar{1}\text{MDK}_{S_t}(k)$ de la manière suivante :

$$\begin{aligned} \text{Max} \quad & z = \sum_{j \in L(S_t)} c_j x_j + \sum_{j \in F_1(S_t)} c_j \\ \text{s.à.} \quad & \sum_{j \in L(S_t)} a_{ij} x_j \leq b_i - \sum_{j \in F_1(S_t)} a_{ij} \quad i = 1,..,m \\ & \sum_{j \in L(S_t)} x_j = k - |F_1(S_t)| \qquad (6) \\ & x_j \in [0,1] \qquad j \in L(S_t) \end{aligned}$$

où la contrainte (6) assure que nous explorons l'hyperplan $(1 \cdot x = k)$. Nous trions ensuite les variables x_j $j \in L(S_t)$ par ordre décroissant de coût réduit, l'intérêt étant de brancher en priorité sur des variables à coût réduit de forte valeur absolue. Premièrement nous fixons les variables qui peuvent être fixées par l'algorithme de fixation de variable (i.e. les variables x_j hors base à l'optimum de $01\text{MDK}_{S_t}(k)$ et de coût réduit $|\bar{c}_j^{S_t}| > BS(S_t) - BI$ à leur valeur optimale) puis nous branchons sur chaque variable libre x_j $j \in L(S_t)$ en les fixant à leur valeur opposée et à chaque fois nous créons un sous-problème. Pour une variable x_j donnée, nous ne créons pas deux sous-problèmes l'un avec x_j fixée à zéro, l'autre avec x_j fixée à un comme il se fait usuellement. Supposons que $\bar{x}_j^{S_t} = 0$, alors créer un problème fils S_{t+1} avec x_j fixée à zéro serait équivalent à réitérer l'évaluation de S_t. Nous explorons donc l'espace des solutions d'une autre manière : Supposons que $\bar{x}^{S_t} = (x_1^{S_t}, x_2^{S_t}, x_3^{S_t}) = (0,0,0)$ à l'optimalité de $0\bar{1}\text{MDK}_{S_t}(k)$, nous générons les problèmes fils suivants : $(1,*,*),(0,1,*),(0,0,1),(0,0,0)$ (où $*$ représente une variable libre). A chaque noeud,

nous créons donc un problème fils dans lequel nous fixons une variable à l'opposé de sa valeur optimale, puis nous créons le problème fils suivant en fixant cette même variable à sa valeur optimale et la variable suivante à sa valeur opposée, ainsi de suite. La figure 1 illustre schématiquement cette méthode de branchement qui nous permet d'explorer implicitement une partie importante de l'arbre de recherche.

FIG. 1 – Schémas de la stratégie de branchement à chaque noeud



Puisque nous ne traitons que des variables hors base, il se produit systématiquement le cas où elles ont toutes été fixées à une valeur et où les variables libres restantes sont des variables de base. Dans ce cas l'algorithme de fixation de variables ne pouvant plus fixer ces variables, nous procédons à une énumération dite *en profondeur d'abord* ou encore *Depth First Search* afin de fixer les variables restantes. Dans le cas opposé, l'exploration d'un noeud S_t s'arrête lorsque la somme des variables fixées à un est égale à k ou lorsque la borne supérieure $BS(S_t)$ est inférieure ou égale au minorant BI.

5 Amélioration apportée à l'algorithme de fixation de variables

Après la résolution par le simplexe de chaque $0\bar{1}$MDK$_{S_t}$, les variables hors base satisfont la contrainte des coûts réduits (2). Comme nous l'avons mentionné précédemment, brancher sur une variable x_j consiste à la fixer à une valeur opposée à sa valeur à l'optimum du simplexe, or changer la valeur de x_j a pour effet d'augmenter le membre de gauche de la contrainte (2) de la valeur $\bar{c}_j$. Ainsi, si l'on suppose que la variable x_j est fixée à sa valeur opposée au noeud S_t, si la somme du coût réduit de x_j plus le coût réduit d'une autre variable libre x_i excède le gap ($\bar{c}_j^{S_t} + \bar{c}_i^{S_t} \geq BS(S_t) - BI$), alors la deuxième variable ne pourra jamais être fixée à sa valeur inverse. Nous pouvons donc fixer cette deuxième variable à sa valeur courante $\bar{x}_j^{S_t}$. Nous intégrons donc à chaque noeud l'algorithme de fixation par cumul des coûts réduits suivant :

Supposons que l'on branche sur x_j $j \in L(S_t)$. Pour chaque variable libre hors base x_i ($i \in L(S_t)$, $i \neq j$), si $\bar{c}_j^{S_t} + \bar{c}_i^{S_t} \geq BS(S_t) - BI$ alors fixer x_i à sa valeur $\bar{x}_i^{S_t}$, sinon la laisser libre.

Ainsi à chaque noeud, deux fixations de variables sont opérées : (1) La fixation directe des variables hors base de coût réduit supérieur au gap et (2) après le branchement sur une variable x_j à sa valeur inverse, la fixation des variables x_i vérifiant $\bar{c}_j^{S_t} + \bar{c}_i^{S_t} \geq BS(S_t) - BI$ (voir les algorithmes 1 et 2 section 6).

6 Propagation de la contrainte des coûts réduits

Comme nous l'avons mentionné précédemment, fixer une variable x_j ($j \in L(S_t)$) à sa valeur opposée a pour conséquence d'augmenter le terme de gauche de la contrainte des coûts réduits (3) de la valeur $|\bar{c}_j^{S_t}|$ correspondante. Il apparaît alors clairement que l'on pourra fixer des variables à leur valeur opposée seulement si la somme des coûts réduits de celles-ci ne dépasse pas la valeur $BS(S_t) - BI$. On définit alors une valeur $gap(S_t)$ comme étant la "quantité de coût réduit disponible" au noeud S_t. $gap(S_t)$ est initialisé à $BS(S_t) - BI$ pour chaque noeud S_t et à chaque variable x_j fixée à sa valeur opposée, $gap(S_t)$ est décrémentée du coût réduit $|\bar{c}_j^{S_t}|$ correspondant. Si au cours de la procédure, il s'avère que $gap(S_t) < 0$ alors la configuration partielle S_t est inconsistante avec la contrainte des coûts réduits. De plus il est clair que la configuration de chaque sous problème doit être consistante avec la contrainte des coûts réduits de chacun de ses problèmes parents. Donc, supposons que l'algorithme de fixation de variables nous donne l'information que x_j doit être fixée à une valeur v après la résolution de $0\bar{1}$MDK$_{S_t}$, alors pour chaque noeud parent S_i $i = (t, \ldots, 0)$ tel que $\bar{x}_j^{S_i} = 1 - v$, la valeur $gap(i)$ sera décrémentée du coût $|\bar{c}_j^{S_i}|$ correspondant. Si $gap(i) < 0$ pour un noeud parent S_i, alors nous pouvons affirmer que la configuration partielle courante S_t est irréalisable et donc couper la recherche à ce noeud.

Les algorithmes 1 et 2 détaillent la procédure récursive d'énumération. L'algorithme 1 est dédié à la fixation de variables et aux branchements. L'algorithme 2 à la création des sous problèmes et à la propagation des coûts réduits.

Cette méthode nous permet de propager implicitement les coûts réduits sans ajouter explicitement de nouvelles contraintes au modèle. Nous considérons effectivement une seule contrainte qui est appliquée à chaque noeud (sous-problème) de l'arbre. La propagation est ensuite effectuée en remontant l'arbre de recherche du noeud courant à la racine. A chaque noeud, nous vérifions que la contrainte des coûts ré-

Algorithm 1: Énumération

Données: $A, b, c, k, F_0(S_t), F_1(S_t)$
si $k = 0$ **alors**
  $return$ solution
$\bar{z}^{S_t}$ = Résoudre $\bar{0}1\text{MDK}_{S_t}(k)$
% $CR(S_t)$ Vecteur des coûts réduits
$BS(S_t) = \bar{z}^{S_t} + \sum_{j \in F_1} c_j$
$gap(S_t) = BS(S_t) - BI$
tri $CR(S_t)$ % Tri décroissant des coûts réduits.
$i \leftarrow 0$
tant que $CR(S_t)_i > BS(S_t) - BI$ **faire**
  $i \leftarrow i + 1$
répéter
  % Branchement
  si $\bar{x}_i^{S_t} = v$ **alors**
    $F_{\bar{v}}(S_t) \leftarrow i$
  % Fixation
  pour $j = 0$ *à* $i - 1$ **faire**
    si $\bar{x}_j^{S_t} = v$ **alors**
      $F_v(S_t) \leftarrow j$
  % Fixation par cumul des coûts réduits
  pour $j = i + 1$ *à* $n - 1$ **faire**
    si $\bar{c}_i^{S_t} + \bar{c}_j^{S_t} > BS(S_t) - BI$ **alors**
      si $x_j^{S_t} = v$ **alors**
        $F_v(S_t) \leftarrow j$
  $programme\ auxiliaire(A, b, c, k, F_0(S_t), F_1(S_t))$
jusqu'à $i = n - 1$

Algorithm 2: Programme auxiliaire

Données: $A, b, c, k, F_0(S_t), F_1(S_t)$
% Création du sous problème
$\_k \leftarrow k - |F_1(S_t)|$ % nouvelle valeur de k
pour $i = 0$ *à* $m - 1$ **faire**
  $u = 0$
  pour $j = 0$ *à* $n - 1$ **faire**
    si $j \in F_1(S_t)$ **alors**
      $\_b_i \leftarrow b_i - a_{ij}$
    sinon
      si $j \notin F_0(S_t)$ *et* $j \notin F_1(S_t)$ **alors**
        $\_c_u \leftarrow c_j$
        $\_a_{iu} \leftarrow a_{ij}$
        $u \leftarrow u + 1$
% propagation des coûts réduits
pour $j = 0$ *à* $n - 1$ **faire**
  si $j \in F_v(S_t)$ **alors**
    pour $l = t$ *à* 0 **faire**
      si $\bar{x}_j^{S_l} \neq v$ **alors**
        $gap(S_l) = gap(S_l) - \bar{c}_j^{S_l}$
      si $gap(l) < 0$ **alors**
        $return$
$énumération(\_A, \_b, \_c, \_k, F_0(S_{t+1}), F_1(S_{t+1}))$

duits est valide, dans le cas contraire le noeud courant est coupé. Contrairement aux méthodes classiques de Branch & Bound, cette propagation nous interdit de mettre la borne inférieure à jour lorsqu'une meilleure solution est trouvée. En effet, chaque sous problème doit faire référence à la même contrainte des coûts réduits et donc à la même borne inférieure, dans le cas contraire la propagation n'aurait aucun sens. Cependant, cela n'est valable que pour l'exploration d'un hyperplan donné ($1 \cdot x = k$). Il est toujours possible d'actualiser la borne inférieure d'un hyperplan par la borne inférieure d'un autre hyperplan déjà exploré.

7 Résultats exprimentaux

Les tests expérimentaux ont été réalisés sur les instances de la OR-Library produites par P.C Chu et J.E Beasley. Cette collection contient des problèmes avec n=100, 250 et 500 variables et m=5, 10 et 30 contraintes. 30 problèmes ont été générés pour chaque n-m combinaison, donnant un total de 270 problèmes. Les valeurs entières a_{ij} ont été générées aléatoirement entre 0 et 1000 et les valeurs des membres de droite on été générées en corrélation avec les a_{ij} tel que $b_i = \alpha \sum_{j=1}^{n} a_{ij}$ où $\alpha = 1/4, 1/2$ and $3/4$. Les coûts de la fonction objectif (c_j) sont générés en corrélation avec les a_{ij} tels que $c_j = \sum_{i=1}^{m} a_{ij}/m + 500d_j$ où d_j est un nombre aléatoire compris entre 0 et 1.

Notre algorithme a été testé sur un P4 cadencé à 3.2 GHz avec 1GB de RAM. Nos résultats sont comparés à ceux produit par le solveur CPLEX 9.2. Le nom complet de chaque instance est cbm.n_r où m est le nombre de contraintes, n le nombre de variables et r le numéro de l'instance. Chaque ensemble de 30 instances avec une combinaison m et n donnée est divisée en 3 sous-ensembles de ratio α différent : $\alpha = 1/4$ pour les instances 0 à 9, $\alpha = 1/2$ pour les instances 10 à 19 et $\alpha = 3/4$ pour les instances 20 à 29. Les tables 1 et 2 détaillent respectivement les résultats obtenus sur les instances à 10 contraintes et 250 variables et celles à 5 contraintes et 500 variables. La description des données par colonne est :

- Pb. : Le numéro r de l'instance.
- z^* : Le minorant obtenu avec Rl^{tabou} pour cette instance.
- z^{opt} : La solution optimale prouvée par notre algorithme.
- k^{opt} : Le nombre d'objets dans la solution optimale.
- $t(s)$: Temps requis en secondes par l'énumération pour trouver la solution optimale (sans prendre en

TAB. 1 – Résultats obtenus sur les problèmes **cb10.250**

Pb	z^*	z^{opt}	k^{opt}	t(s)	CP t(s)	Pb	z^*	z^{opt}	k^{opt}	t(s)	CP t(s)
0	59187	59187	68	**4928**	-	15	110845	110845	130	**5670**	-
1	58710	58781	69	43618	**4369**	16	106077	106077	129	**7514**	-
2	58097	58097	69	**1194**	6746	17	106686	106686	128	**5105**	-
3	61000	61000	70	**7152**	-	18	109825	109829	127	**5257**	-
4	58092	58092	67	**28589**	-	19	106723	106723	131	**1106**	5716
5	58824	58824	68	**952**	7394	20	151809	151809	187	**764**	4695
6	58704	58704	67	**883**	8255	21	148772	148772	188	**2112**	-
7	58933	58936	69	**86274**	-	22	151909	151909	189	**416**	5048
8	59387	59387	68	**3544**	-	23	151324	151324	189	**469**	2234
9	59208	59208	69	**6658**	-	24	151966	151966	191	**738**	5727
10	110913	110913	129	**4708**	-	25	152109	152109	189	**498**	2826
11	108717	108717	127	**4882**	-	26	153131	153131	189	**82**	374
12	108932	108932	128	**3158**	-	27	153578	153578	187	**5429**	-
13	110086	110086	131	**14688**	-	28	149160	149160	187	**572**	1638
14	108485	108485	128	**2094**	9869	29	149704	149704	190	**596**	2487

compte le calcul du minorant).

– CP t(s) : Temps requis en secondes à CPLEX pour trouver la solution optimale ou - si la solution optimale n'est pas trouvée en moins de 4h de temps de calcul.

De nouvelles preuves d'optimalité ont été obtenues pour les instances à 10 contraintes et 250 variables (cb10.250 cf. table 1), ces solutions n'avaient jamais été obtenues auparavant. CPLEX résout 48% des instances en moins de 4 heures, alors que notre méthode parvient à en résoudre 87% dans le même temps. Notre algorithme a prouvé l'optimalité de toutes les instances à 500 variables et 5 contraintes, cependant les solutions optimales étaient connues grâce à James *et al.* [3] avec leur méthode d'énumération basée sur la résolution répétée de multiples sous problèmes. Les instances cb5.100, cb10.100, cb30.100 et cb5.250 ont toutes été prouvées par notre algorithme et le solveur CPLEX dans un temps inférieur à 4 heures. Pour toutes ces instances, notre méthode dépasse CPLEX en temps de calcul. Un des avantages de notre approche est qu'elle a permis d'effectuer de très longs calculs (24 heures pour l'instance cb10.250_7) sans excéder la consommation de mémoire alors que CPLEX dépasse la capacité de mémoire après 4 heures. De plus, notre approche permet d'utiliser le calcul distribué. Il est en effet possible d'explorer parallèlement chaque 01MDK(k) avec $k \in \{k_{min}, \ldots, k_{max}\}$, ce qui permettrait de résoudre des instances plus larges.

Les instances à 250 variables et 30 contraintes demeurent trop difficiles à prouver, nous avons donc lancé notre énumération dans un temps limite de 4 heures afin d'essayer de réduire la taille des instances. En effet, dans le cas où notre algorithme est arrêté prématurement, il nous fournit des résultats intéressants : (1)Il peut fixer exactement certaines variables pour un hyperplan exploré et (2) il peut prouver qu'aucune solution meilleure que le minorant donné n'est dans un hyperplan $1 \cdot x = k$. L'idée première étant de réduire l'encadrement du nombre d'objets à l'optimum (donc de couper certains hyperplans), nous nous sommes interrogés sur la meilleure façon d'explorer ces hyperplans.

En considérant le fait que la fonction $\bar{z}(\mu) = c \cdot \bar{x}_\mu$ (l'optimum de $\bar{0}$1MDK(μ) où $\mu \in \mathbb{R}$) est convexe[1], dans notre cas, la meilleure manière d'explorer les hyperplans est de partir des valeurs extrêmes possibles de $[k_{min}, k_{max}]$ jusqu'aux valeurs centrales entières : k_{min}, k_{max}, $k_{min}+1$, $k_{max}-1$,... L'objectif étant de prouver que pour un hyperplan $1 \cdot x = k$, il n'existe aucune solution $z > z^*$ appartenant à cet hyperplan. Nous explorerons ainsi en premier lieu les hyperplans ayant les "moins bonnes" valeurs $\bar{z}(k)$ potentielles.

Nous exposons table 3 les nouveaux encadrements du nombre d'objets à l'optimum ($k_{min} \leq k^{opt} \leq k_{max}$) que nous avons obtenus pour les instances à 30 contraintes et 250 variables. Les valeurs en gras indiquent les nouvelles bornes obtenues comparativement à celles obtenues avec la méthode exposée en section 4.3. Nous avons réduit les bornes de l'encadrement du nombre d'objets à l'optimum de 76% des instances dans un temps limite de 4 heures. La table 4 fournit le nombre de variables fixées exactement pour les hyperplans pour lesquels soit l'on a trouvé une solution aussi bonne que le minorant, soit l'algorithme a atteint la limite de temps imposée.

8 Conclusion

Dans ce papier nous avons démontré l'efficacité d'une énumération implicite combinant la propagation de contraintes et la programmation linéaire pour résoudre des instances difficiles du sac à dos multidimensionnel en variables binaires. La prise en compte d'une

[1] Cette propriété peut être prouvée en appliquant directement le théorème 10.2 du livre Linear Programming [2]

TAB. 2 – Résultats obtenus sur les problèmes **cb5.500**

Pb	z^*	z^{opt}	k^{opt}	t(s)	CP t(s)	Pb	z^*	z^{opt}	k^{opt}	t(s)	CP t(s)
0	120148	120148	147	**168**	8001	15	220530	220530	262	**78**	3122
1	117879	117879	148	**14**	855	16	219989	219989	266	**28**	1211
2	121131	121131	144	**56**	4687	17	218215	218215	265	**32**	1096
3	120804	120804	149	**54**	6050	18	216976	216976	262	**93**	2373
4	122319	122319	147	**54**	1578	19	219719	219719	267	**124**	2522
5	122024	122024	153	**72**	3678	20	295828	295828	383	**7**	47
6	119127	119127	145	**113**	6937	21	308086	308086	384	**58**	475
7	120568	120568	150	**47**	2672	22	299796	299796	385	**17**	187
8	121583	121586	149	**107**	-	23	306480	306480	384	**40**	1182
9	120717	120717	151	**95**	5756	24	300342	300342	385	**55**	204
10	218428	218428	267	**111**	1457	25	302571	302571	385	**16**	988
11	221202	221202	265	**19**	905	26	301339	301339	385	**24**	366
12	217542	217542	264	**170**	3728	27	306454	306454	383	**6**	148
13	223560	223560	263	**158**	5009	28	302828	302828	384	**33**	367
14	218966	218966	267	**12**	485	29	299910	299910	378	**126**	478

TAB. 3 – Nouveaux encadrements du nombre d'objets à l'optimum pour les problèmes cb30.250

Pb	k_{min}	k_{max}	Pb	k_{min}	k_{max}
cb30.250_0	**62**	65	cb30.250_15	**125**	128
cb30.250_1	64	67	cb30.250_16	**125**	**128**
cb30.250_2	62	66	cb30.250_17	**124**	**127**
cb30.250_3	62	66	cb30.250_18	**123**	128
cb30.250_4	63	65	cb30.250_19	**125**	128
cb30.250_5	62	65	cb30.250_20	**187**	**189**
cb30.250_6	**62**	66	cb30.250_21	**187**	**188**
cb30.250_7	**63**	**65**	cb30.250_22	**186**	**188**
cb30.250_8	**63**	66	cb30.250_23	**186**	**188**
cb30.250_9	62	65	cb30.250_24	186	**188**
cb30.250_10	**124**	**127**	cb30.250_25	**186**	**188**
cb30.250_11	**124**	127	cb30.250_26	**186**	**188**
cb30.250_12	**123**	**126**	cb30.250_27	**186**	**188**
cb30.250_13	123	127	cb30.250_28	**186**	**188**
cb30.250_14	**124**	**127**	cb30.250_29	**186**	**188**

TAB. 4 – Nombre de variables fixées exactement par hyperplan sur les problèmes cb30.250

Pb	k	#	Pb	k	#
cb30.250_0	65	173	cb30.250_15	128	80
cb30.250_1	64	120	cb30.250_16	125	74
cb30.250_2	62	166	cb30.250_17	124	57
cb30.250_3	62	106	cb30.250_18	126	84
cb30.250_4	63	24	cb30.250_19	128	168
cb30.250_5	62	115	cb30.250_20	187	119
cb30.250_6	66	101	cb30.250_21	187	165
cb30.250_7	63	35	cb30.250_22	186	132
cb30.250_8	66	132	cb30.250_23	186	161
cb30.250_9	62	128	cb30.250_24	187	84
cb30.250_10	124	98	cb30.250_25	186	94
cb30.250_11	127	105	cb30.250_26	186	154
cb30.250_12	123	69	cb30.250_27	186	55
cb30.250_13	123	175	cb30.250_28	186	109
cb30.250_14	124	22	cb30.250_29	186	108

contrainte de distance au gap, par propagation des coûts réduits, ainsi que la méthode efficace de branchements intégrée nous permettent d'élaguer significativement l'arbre de recherche. Les nouvelles preuves d'optimalité ainsi que la réduction de l'encadrement du nombre d'objets à l'optimum pour les problèmes à 30 contraintes et 250 variables, constituent les deux contributions de notre approche. Pour les instances non résolues en un temps raisonnable, nous envisageons d'expérimenter une version distribuée de notre algorithme dont la structure se prête bien au parallélisme.

Références

[1] J.E. Beasley. Or-library : Distributing test problems by electronic mail. *J. Operational Research Society*, 41 :1069–1072, 1990.

[2] V. Chvátal. *Linear Programming.* W.H. Freeman and Company, 1983.

[3] Ross J.W. James and Yuji Nakagawa. Enumeration methods for repeatedly solving multidimensional knapsack sub-problems. Technical Report 10, The Institute of Electronics, Information and Communication Engineers, october 2005.

[4] Cristian Oliva, Philippe Michelon, and Christian Artigues. Constraint and linear programming : Using reduced costs for solving the zero/one multiple knapsack problem. *International Conference on Constraint Programming, CP 01, Proceedings of the workshop on Cooperative Solvers in Constraint Programming(CoSolv 01)*, pages 87,98, 2001.

[5] Michel Vasquez and Jin-Kao Hao. An hybrid approach for the 0-1 multidimensional knapsack problem. *Proc. 17th International Joint Conference on Artificial Intelligence, IJCAI-01, Seattle, WA*, August 2001.

[6] Michel Vasquez and Yannick Vimont. Improved results on the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, (165) :70–81, 2005.