7e Conférence Internationale de MOdélisation et SIMulation - MOSIM'08 - du 31 mars au 2 avril 2008 – Paris- France
« Modélisation, Optimisation et Simulation des Systèmes : Communication, Coopération et Coordination.».

APPLICATION DE « RESOLUTION SEARCH » AU PROBLÈME DU SAC Á DOS MULTIDIMENSIONNEL EN 0-1

S. BOUSSIER, M. VASQUEZ et Y.VIMONT

LGI2P, Ecole des Mines d'Alès
Parc Scientifique Georges Besse
30035 Nîmes Cedex 1
{Sylvain.Boussier, Michel.Vasquez, Yannick.Vimont}@ema.fr

RÉSUMÉ : *Le problème du sac à dos multidimensionnel en variables binaires (MKP pour Multidimensional Knapsack Problem) se rencontre dans plusieurs domaines de l'industrie (télécommunications, transport, logistique, économie etc.). De nombreuses méthodes approchées ont été développées pour ce problème NP dur mais sa résolution exacte reste un challenge difficile surtout lorsque le nombre de contraintes augmente. En 1997, Vašek Chvátal propose une alternative à l'énumération implicite pour la résolution exacte des problèmes d'optimisation à variables binaires qu'il nomme Resolution Search. Nous proposons, dans cet article, l'application de Resolution Search à la résolution du MKP basée sur une analyse des coûts réduit à l'optimum de sa relaxation linéaire. Expérimentalement, notre méthode est capable de prouver l'optimalité d' instances difficiles de la OR-Library*

MOTS-CLÉS : *MKP, Sac à dos multidimensionnel, Méthode exacte, Resolution Search, coûts réduits*

1. INTRODUCTION

Le problème du sac à dos multidimensionnel en variables binaires (MKP pour Multidimensional Knapsack Problem) est un problème classique d'optimisation combinatoire qui appartient à la famille des problèmes NP complets. Il peut être défini de la manière suivante:

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

où n est le nombre de variables, m le nombre de contraintes et $c \in N^n$, $A \in N^{m \times n}$ et $b \in N^m$. 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. Le MKP a été l'objet de nombreux travaux car il recouvre de nombreux domaines d'application. Notons entre autres les problèmes d'allocation de ressource dans les systèmes distribués, les problèmes de découpe ou les problèmes de budgétisation (Gavish et Pirkul (1985), Shih (1979), Gilmore et Gomory (1966)). Le développement de méthodes exactes pour le MKP a commencé il y a plusieurs décennies (Dantzig (1957), Balas (1965), Fayard et Plateau (1982), Martello et Toth (1988)) et plus récemment, James et Nakagawa (2005) et Vimont *et al.* (2007) ont prouvé certaines instances difficiles de la OR-library (Beasley (1990)).

En 1997, Vašek Chvátal propose une alternative à l'énumération implicite pour les problèmes d'optimisation à variables binaires qu'il nomme *Resolution Search*. Cette méthode originale inspirée des principe du *backtracking intelligent* (Ginsberg (1993)) utilise les échecs rencontrés lors de l'exploration pour éliminer certaines parties de l'espace de recherche, l'objectif étant de rendre la résolution moins dépendante de la stratégie de branchement. Dans ce papier nous nous intéressons à la résolution du MKP par *Resolution Search* en prenant en compte une contrainte basée sur les coûts réduits à l'optimum de sa relaxation linéaire et démontrons l'intérêt de cette méthode qui donne des résultats comparables aux meilleures méthodes exactes actuelles pour ce problème.

En section 2, nous décrivons le principe général de *Resolution Search*,. En section 3, nous détaillons la contrainte des coûts réduits utilisée. Nous décrivons notre algorithme en section 4 puis exposons les résultats expérimentaux en section 5.

2. PRINCIPE DE RESOLUTION SEARCH

Resolution Search a été proposé par Chvátal (1997) comme une alternative à l'énumération implicite. Cette méthode s'apparente au *Dynamic Branch & Bound* proposé par Glover et Tangedahl (1976) mais a été conçue spécifiquement pour les problèmes à variables

binaires. *Resolution Search* est basée sur le principe d'apprentissage et cherche à réduire progressivement l'espace des solutions candidates à l'amélioration de la meilleure solution connue lors de l'exploration. Dans cette section nous essayons de donner brièvement le fonctionnement de *Resolution Search* cependant la méthode n'étant pas intuitive, nous invitons le lecteur à consulter des ouvrages plus détaillés sur le sujet (Chvátal (1997), Hanafi et Glover (2002), Demassey (2003) et Palpant (2005)).

2.1 Notations

Dans ce qui suit, chaque solution partielle est considérée comme une clause au sens de la logique propositionnelle. On note $u = (u_1,...,u_n)$ une clause ou vecteur de $\{0,1,*\}^n$ correspondant à une instanciation partielle où complète du problème où $u_i = *$ si u_i est une variable libre. Soit u et v deux clauses telles que $v_j = u_j$ pour tout $u_j \neq *$, alors v est une extension de u et on note $u \prec v$ ce qui définit une relation d'ordre partiel sur $\{0,1,*\}^n$. Deux clauses C_1 et C_2 sont en contradiction si il existe exactement un littéral w tel que $w \in C_1, \overline{w} \in C_2$. Leur résolvante notée $C_1 \nabla C_2$ est définie par la clause $(C_1 - \{w\}) \cup (C_2 - \{\overline{w}\})$.

2.2 Principe d'exploration

On appelle *oracle(u)*, la fonction d'évaluation d'une solution partielle. Dans le cas d'un problème de maximisation, *oracle* retournera une borne supérieure des solutions induites par la clause *u* c'est-à-dire toute solution $u^* \in \{0,1\}^n$ telle que $u \prec u^*$.

L'exploration de l'espace se fait par une fonction *obstacle* qui prend en argument une clause de départ *u* et cherche à l'étendre en *u** jusqu'à trouver une solution. Soit BI la meilleure solution connue jusqu'à présent, *obstacle* est composée de deux phases distinctes :

(1) La *Waxing Phase* qui correspond à la phase de descente. La clause de départ *u* est étendue en *u** tant que *oracle(u*)*>BI.

(2) La *Waning Phase* qui correspond à la phase de remontée. Si la phase de descente échoue, c'est à dire que *oracle(u*)* $\leq$ BI, nous cherchons à trouver la sous instanciation minimale qui est responsable de cet échec. Pour cela, on desinstancie les variables précédemment instanciées, sauf la dernière, tant que *oracle(u*)* $\leq$ BI.

On appelle *S* l'obstacle ou nogood minimal correspondant à l'échec de l'extension de *u*. *S* sera alors mémorisé d'une manière spécifique décrite en section 2.3. La figure 1 schématise l'exploration de l'arbre de recherche par *Resolution Search*.

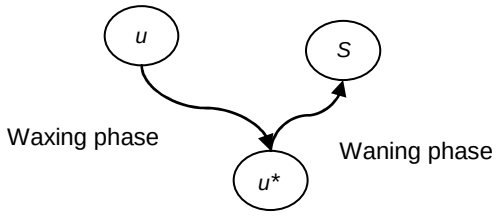


Figure 1. Représentation de la fonction *obstacle*

2.3 Principe de mémorisation des échecs

A chaque étape de l'exploration, les échecs passés permettent de guider les explorations futures de l'espace de recherche. Un échec ou obstacle est une clause S pour laquelle toute clause u telle que $S \prec u$ est soit irréalisable, soit donne une solution plus mauvaise que la meilleure solution connue. Afin de limiter l'espace mémoire consommé, tous les obstacles rencontrés ne sont pas mémorisés mais stockés dans une famille de clauses possédant une structure particulière. Cette famille est appelée *path-like family* que l'on note $\mathcal{F}$.

2.3.1 Structure

La famille $\mathcal{F}$ contient un ensemble de clauses $C_1, C_2, ..., C_m$ et des littéraux associés $w_1, w_2, ..., w_m$ tels que :

- $w_i \in C_j$ si et seulement si $j = i$,
- Si $\overline{w}_i \in C_j$ alors $j > i$,
- Si $w \in C_i$ et $\overline{w} \in C_j$, alors $w = w_i$ ou $\overline{w} = w_j$.

Un exemple de famille *path-like* (Chvátal (1997)):

$C_1 = x_2 x_3 \quad (w_1 = x_2)$,
$C_2 = x_3 x_6 \quad (w_2 = x_6)$,
$C_3 = \overline{x}_1 \overline{x}_5 \overline{x}_9 \quad (w_3 = \overline{x}_1)$,
$C_4 = \overline{x}_2 \overline{x}_5 \overline{x}_8 \quad (w_4 = \overline{x}_8)$,
$C_5 = x_3 \overline{x}_5 x_7 \overline{x}_9 \quad (w_5 = x_7)$,
$C_6 = \overline{x}_2 \overline{x}_4 \overline{x}_6 \quad (w_6 = \overline{x}_4)$,

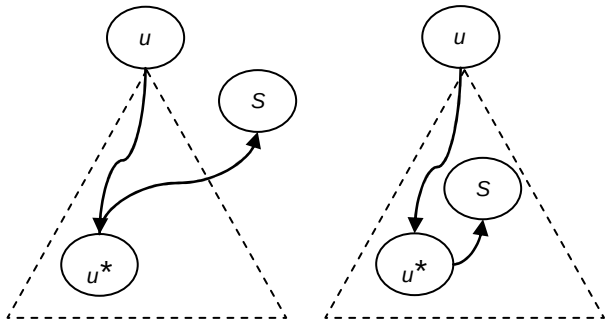
L'intérêt de cette structure est de pouvoir obtenir facilement le nœud suivant dans l'exploration, qui est représenté par la clause $u = (\bigcup_{i=1}^{m} (C_i - \{w_i\}) \cup \{\overline{w}_1, \overline{w}_2, ..., \overline{w}_m\}$. Cette clause n'est l'extension d'aucune clause de la famille $\mathcal{F}$ et donc toute extension de *u* est potentiellement un candidat pour l'amélioration de la meilleure solution connue.

2.3.2 Mise à jour

Deux cas sont à considérer lors de l'ajout d'une nouvelle clause à la famille $\mathcal{F}$: Soit il y a eu une de phase de descente (voir Figure 2.) et *u* (le nœud précédemment fourni par $\mathcal{F}$) a été étendu pour donner *S* tel que $S \not\subset u$, soit il n'y a pas eu de phase de descente (voir

Figure 3.) et $S \subseteq u$ c'est-à-dire que soit *u* est un extension de *S* soit *u* est égale à *S*.

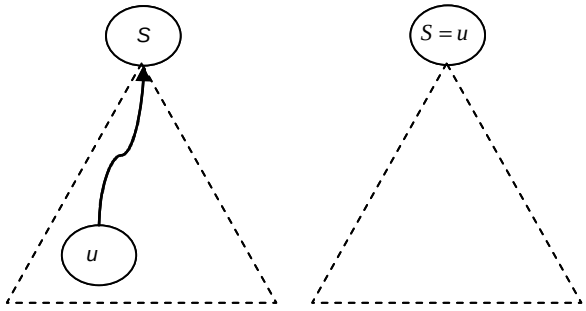
(1) $S \not\subset u$: Dans ce cas, il existe un littéral $w \in S \setminus u$ ($u \prec S$ ou $u \not\prec S$ cf. Figure 2.), on ajoute alors simplement *S* à la suite des clauses de $\mathscr{F}$ et on choisit un littéral *w* de *S* tel que $w \notin C_1 \cup C_2 \cup \ldots C_m$ et $\overline{w} \notin C_1 \cup C_2 \cup \ldots C_m$.



$u \not\prec S$ $\qquad$ $u \prec S$

Figure 2. . Mise à jour de $\mathscr{F}$ avec phase de descente

(2) $S \subseteq u$: Dans ce cas, si une variable correspondant à un littéral w_k n'a pas été désinstanciée dans la phase de remontée, elle est égale à $\overline{w}_k$ dans *S*, et pour les autres variables, soit elles sont instanciées à la même valeur dans *S* et dans C_k, soit une des deux n'est pas instanciée ($S \prec u$ ou $S = u$ cf. Figure 3.). On peut donc *réduire* la famille $\mathscr{F}$ par résolvantes successives de *S* sur les clauses de $\mathscr{F}$.



$S \prec u$ $\qquad$ $S = u$

Figure 3. Mise à jour de $\mathscr{F}$ sans phase de descente

Soit *R*=*S*, pour chaque clause C_i, $i = m,...,2,1$ si $\overline{w}_i \in R$ faire $R = R \nabla C_i$. Si $R = \varnothing$, alors la meilleure solution connue est l'optimum sinon on cherche le plus petit indice *k* tel que *R* est incluse dans la réunion des *k* premières clauses et w_k un littéral qui appartient à *R* mais à aucune des autres clauses puis on fait $C_k = R$. Ensuite on supprime toutes les clauses C_i avec *i* > *k* qui contiennent w_k ceci afin de conserver la structure *path-like* de la famille $\mathscr{F}$.

3 PRISE EN COMPTE DES COÛTS RÉDUITS

La prise en compte des coûts réduits pour la résolution du MKP n'est pas nouvelle (Saunders et Schinzinger (1970), Fayard et Plateau (1982), Oliva *et al.* (2001)) et les récents travaux de Vimont *et al.* (2007) ont prouvé l'efficacité de cette approche pour la résolution d'instances difficiles de MKP. La prise en compte des coûts réduits et la connaissance d'une bonne borne inférieure du problème nous permet d'une part de fixer certaines variables et d'autre part de couper certains nœuds de l'arbre de recherche. Nous verrons en section 3.1 la contrainte des coûts réduits utilisée, en section 3.2 les fixations de variables possibles grâce à cette contrainte et en section 3.3 les obstacles qu'elle permet d'identifier.

3.1 Contrainte des coûts réduits

Considérons la relaxation linéaire (01MKP) du problème établie comme suit :

$$\begin{aligned} &\text{Maximiser} \sum_{j=1}^{n} c_j x_j \\ &\text{sujet à} \quad Ax \le b \\ &x \in [0,1]^n \end{aligned} \qquad (1)$$

Dans la forme standard, l'inégalité (1) doit être transformée en égalité en introduisant des variables d'écart à valeurs positives ou nulles, ce qui nous donne :

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

où s est le vecteur des m variables d'écart. Soit $(\bar{x}, \bar{s})$ une solution optimale du programme linéaire 01MKP 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 01MKP peut également être écrit de manière totalement équivalente comme suit :

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

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 connaissons une borne inférieure BI∈ IN du programme linéaire en nombres entiers original, alors chaque solution meilleure que BI doit satisfaire la contrainte suivante :

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

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

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, ainsi leur somme est 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 une contrainte que nous appellerons *contrainte des coûts réduits* :

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

3.2 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 du 01MKP 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 :

$$\bar{x}_j = 1 \Rightarrow x_j \geq \left\lceil 1 - \frac{BS - BI}{|\bar{c}_j|} \right\rceil \qquad (3)$$

$$\bar{x}_j = 0 \Rightarrow x_j \leq \left\lfloor \frac{BS - BI}{|\bar{c}_j|} \right\rfloor \qquad (4)$$

où $\bar{x}_j$ est la valeur de x_j à l'optimum du 01MKP et $\bar{c}_j$ son coût réduit correspondant.

En utilisant les relations (4) et (5) nous pouvons établir la règle de fixation de variable suivante : pour chaque variable x_j hors base à l'optimum du 01MKP, si $\bar{c}_j > BS - BI$, alors x_j peut-être fixée à zéro (resp. un) si et seulement si $\bar{x}_j$ est égale à zéro (resp. un).

```
Mettre à jour la consommation de
    ressources b ;
```

3.3 Identification d'obstacles

La contrainte des coûts réduits (2) nous permet de faire l'observation suivante : chaque variable hors base x_j fixée à l'opposé de sa valeur à l'optimum du 01MKP $\bar{x}_j$ a pour conséquence d'augmenter le terme de gauche de la contrainte des coûts réduits de la valeur $|\bar{c}_j|$ correspondante. Il apparaît alors clairement que l'on pourra fixer des variables hors base à leur valeur opposée seulement si la somme des coûts réduits de celles-ci ne dépasse pas la valeur *BS-BI*. On définit alors une valeur *gap* comme étant la « quantité » de coût réduit disponible. Initialement *gap=BS-BI* et pour chaque variable x_j fixée à *1-*$\bar{x}_j$, *gap* est décrémentée de la valeur de son coût réduit $|\bar{c}_j|$. Si *gap* < 0 alors la configuration partielle correspondante est inconsistante avec la contrainte des coûts réduits et elle représente un obstacle.

4 DESCRIPTION DE L'ALGORITHME

Dans cette section nous décrivons l'implémentation de l'algorithme général. En section 4.1 nous décrivons rapidement l'algorithme de calcul de borne inférieure utilisé, en section 4.2 nous décrivons une procédure d'encadrement du nombre d'objets à l'optimum et en section 4.3, nous décrivons une pré-procédure de fixation de variables. Nous décrivons ensuite l'implémentation de *Resolution Search*, à savoir la fonction *oracle* en section 4.4, la fonction *obstacle* en section 4.5 et une stratégie de branchement dans la mise à jour de la famille $\mathcal{F}$ en section 4.6.

4.1 Calcul d'une borne inférieure

La borne inférieure est calculée avec l'algorithme $Fix + RL^{tabou}$ proposé par Vasquez et Vimont (2005) qui est un algorithme hybride combinant la programmation linéaire et la recherche tabou couplée avec une heuristique de fixation de variables. Notons que nous avons limité le nombre d'itérations de l'algorithme afin d'avoir un ratio (consommation c.p.u. / qualité de la borne) approprié. Ceci explique également pourquoi les bornes fournies sont moins bonnes que les solutions précédemment publiées dans Vasquez et Vimont (2005) (cf. section 5).

4.2 Encadrement du nombre d'objets à l'optimum

Dans de précédents travaux (Vasquez et Hao (2001), Vasquez et Vimont (2005), Vimont *et al.* (2007)) nous soulignons le fait que le nombre d'objets *k* peut être facilement encadré par deux valeurs $k_{\min}$ et $k_{\max}$. En effet, soit *BI* 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 :

$$\begin{cases} \text{Minimiser } 1.x, \text{ sujet à} \\ A.x \le b \text{ et } cx \ge BI \text{ et } x \in [0,1]^n \end{cases}$$

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

$$\begin{cases} \text{Maximiser } 1.x, \text{ sujet à} \\ A.x \le b \text{ et } cx \ge BI \text{ et } x \in [0,1]^n \end{cases}$$

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 $c.x \ge BI$ ne sera pas satisfaite et si nous prenons plus de $\lfloor z^*_{\max} \rfloor$ objets, l'une au moins des contraintes $Ax \le b$ ne sera plus vérifiée. La première phase de notre algorithme consiste donc à calculer les ($k_{\max}$ - $k_{\min}$ + 1) différentes valeurs de *k*, l'exploration se limitera ensuite à chaque hyperplan *k*.

4.3 Réduction du problème par fixation de variables

Pour chaque hyperplan *k*, nous définissons le problème 01MKP(*k*) qui correspond à la relaxation linéaire du MKP avec l'ajout d'une contrainte sur le nombre d'objets *1.x=k*.

$$(01MKP(k)) \begin{cases} \text{Maximiser } c.x, \text{ sujet à} \\ A.x \le b, 1.x = k \text{ et } x \in [0,1]^n \end{cases}$$

La résolution par l'algorithme du simplexe de chaque 01MKP(*k*) nous fournit les valeurs optimales de chaque variable et les coûts réduits correspondants. Suivant la règle de fixation de variable énoncée en section 3.2, nous fixons toutes les variables hors base de coût réduit supérieur à *BS-BI* à leur valeur optimale et nous résolvons le sous problème correspondant.

4.4 Fonction *oracle*

La fonction $oracle(u, k, \bar{x}, \bar{c})$ nous donne une évaluation d'une clause ou solution partielle *u* sur l'hyperplan *k*. Cette évaluation ou borne supérieure est calculée en résolvant le 01MKP(*k*) uniquement sur l'ensemble des variables libres de la clause *u*, que nous appellerons 01MKP(*u,k*). Le vecteur $\bar{x}$ nous donne les valeurs optimales et $\bar{c}$ les coûts réduits correspondants. Soit F_0 l'ensemble des indices des variables fixées à zéro dans *u*, F_1 l'ensemble des indices des variables fixées à un et *L* l'ensemble des indices des variables libres. $oracle(u, k, \bar{x}, \bar{c})$ retourne la valeur de la solution optimale du problème 01MKP(*u,k*) définit ci-dessous à laquelle on ajoute la somme des coûts des variables fixées à 1.

$$(01MKP(u,k)) \quad \text{Maximiser} \sum_{j \in L} c_j x_j + \sum_{j \in F_1} c_j$$

$$\text{sujet à} \quad \sum_{j \in L} a_{ij} x_j \le b_i - \sum_{j \in F_1} a_{ij} \quad i = 1,...m$$

$$\sum_{j \in L} x_j = k - |F_1|$$

$$x_j \in [0,1] \quad j \in L$$

La solution optimale du (01MKP(u, k)) est notée $\bar{z}$. L'évaluation d'une solution (partielle ou complète) *u* est donc définie de la façon suivante :

$$oracle(u, k, \bar{x}, \bar{c}) = \bar{z} + \sum_{j \in F_1} c_j$$

Dans le cas où le nombre de variables libres est inférieur ou égal à 20, nous préférons énumérer ces variables par une procédure de recherche en profondeur d'abord (*Depth First Search)*. En effet, l'énumération des variables nous donnera la solution optimale au lieu d'une borne supérieure et de plus, cette énumération coûte moins en termes de temps c.p.u. que la résolution du simplexe pour des problèmes de cette taille.

4.5 Fonction *obstacle*

La fonction *obstacle(u,S)* est la fonction d'exploration de l'espace de recherche. Elle comporte deux phases : la phase de descente (*Waxing Phase*) décrite en 4.5.1 et la phase de remontée (*Waning Phase*) décrite en 4.5.2.

4.5.1 Phase de descente

Partant du nœud de départ *u* donné par la famille *path-like* et connaissant une borne inférieure *BI* qui est la valeur de la meilleure solution connue, la *Waxing Phase* descend l'arbre de recherche par branchements successifs. Elle remplace les variables libres * de *u* par des valeurs 0 ou 1 jusqu'à obtenir une extension *u** de *u* telle que : soit *u** viole les contraintes du problème, soit $u^* \in \{0,1\}^n$. Dans le deuxième cas, *u** devient la nouvelle meilleure solution connue.

Notre stratégie consiste à « chercher l'erreur » afin de couper au plus vite la descente et générer les obstacles les plus petits possibles au plus tôt. Comme nous l'avons mentionné en section 3.3, la contrainte des coûts réduits nous donne l'information que pour toute solution valide (partielle ou complète), la somme des valeurs absolues des coûts réduits des variables hors base fixées à l'opposé de leur valeur optimale est inférieure à au gap (*BS-BI*). Un obstacle lié à la contrainte des coûts réduits correspond à une solution partielle telle que la somme des coûts réduits des variables hors base fixées à leur valeur opposée est supérieure au gap. Pour une solution partielle *u* donnée, la fonction $oracle(u, k, \bar{x}, \bar{c})$ nous fournit la contrainte des coûts réduits sur les variables

libres du nœud u : le gap *BS-BI*, les valeurs optimales $\bar{x}$ et leurs coûts réduits $\bar{c}$ correspondants.

```
Données:
    Borne inférieure (BI)
    Clause de départ (u)
    Nombre d'objets (k)
    Contrainte globale ( x̄^g , c̄^g , gapg)

bound = oracle(u,k,x̄,c̄);
do{
    choisir j tel que u_j=* et |c̄_j| maximal;
    u*_j = x̄_j ;
} While( c̄_j > gap);
bound = oracle(u,k,x̄,c̄);
if(bound ≥ BI) {
 while(u ∉ {0,1}^n ) {
    gap = bound - BI;
    choisir j tel que u_j=* et |c̄_j| maximal;
    if( x̄_j > 0.5) u_j=0;
    else u_j=1;

    // Contrainte locale
    if( x̄_j hors base et c̄_j > gap)
        break;
    else {
        gap = gap - c̄_j ;
        do{
            choisir k tel que x̄_k hors base, u_k=*
            et |c̄_k| maximal;
            u*_k = x̄_k ;
        } While( c̄_k > gap);
    }

    //Contrainte globale
    if( x̄^g_j hors base et u_j == 1 - x̄^g_j ) {
        if( c̄^g_j > gapg)
            break;
        else {
            gapg = gapg - c̄^g_j ;
            do{
                choisir k tel que x̄^g_k hors base
                u_k=* et |c̄^g_k |maximal;
                u*_k = x̄^g_k ;
            } while( c̄^g_k > gapg)
        }
    }

    bound = oracle(u,k,x̄,c̄);
    if(bound < BI) break;
 }
}
```

Figure 4. Algorithme de descente ou Waxing phase

Dans la phase de descente, nous considérons deux contraintes des coûts réduits : (1) la contrainte dite « globale » qui est la contrainte correspondant aux coûts réduits au nœud racine de l'hyperplan k et (2) la contrainte dite « locale » qui est celle considérant les variables libres du nœud u^* lors de l'extension du nœud u. L'algorithme de descente ou *Waxing Phase* consiste à rechercher des obstacles S tels que soit $oracle(S,k,\bar{x},\bar{c}) \leq BI$, soit S est inconsistante avec la contrainte des coûts réduits.

Les deux contraintes sont alors prises en compte. (1) Pour la contrainte globale, la résolution du 01MKP(k) au nœud racine nous donne un vecteur solution $\bar{x}^g$, un vecteur des coûts réduits $\bar{c}^g$ et une valeur *gapg* correspondante. Si la somme des coûts réduits des variables hors base fixées à l'opposé de leur valeur optimale dans u^* est supérieure à *gapg*, alors la descente est stoppée et $S=u^*$ est un obstacle. (2) Pour la contrainte locale, à chaque branchement, la résolution du 01MKP(u^*,k) nous donne un vecteur solution $\bar{x}$, un vecteur de coûts réduits $\bar{c}$ et un *gap*. Si l'on branche sur une variable hors base et la fixons à sa valeur opposée alors si son coût réduit est supérieur au gap, la descente est stoppée et $S=u^*$ est un obstacle. Dans les deux cas, si le branchement ne coupe pas la contrainte des coûts réduits, le gap est mis à jour et toutes les variables de coût réduit supérieur au gap (selon la contrainte locale ou globale) sont fixées à leur valeur dans la contrainte correspondante. L'objectif ici est d'exploiter au maximum l'impact de la fixation de la variable de branchement à sa valeur opposée. La fixation des variables hors base par la contrainte des coûts réduits peut nous permettre de converger rapidement vers une solution réalisable. L'algorithme figure 4. décrit cette phase de descente.

4.5.2 Phase de remontée

La phase de remontée cherche, à partir d'un obstacle u^*, à générer un obstacle minimal S tel que $S \prec u^*$. Pour cela, Chvátal (1997) propose de mémoriser dans une liste toutes les variables instanciées sur u^* (sauf la dernière) et de désinstancier sur u^* chaque variable de la liste dans l'ordre inverse d'instanciation tant que $oracle(u,k,\bar{x},\bar{c}) < BI$.

Cette version de la phase de remontée ne donne cependant pas de bons résultats car les appels multiples à l'algorithme du simplexe dans la fonction *oracle* ralentissent considérablement le processus de recherche.

Nous avons choisit d'implémenter une version plus légère de la phase de remontée en ne considérant que la consistance de la solution partielle avec la contrainte des coûts réduits globale. Si après la phase de descente, la contrainte des coûts réduits globale n'est plus satisfaite, nous désinstancions chaque variable dans l'ordre inverse d'instanciation tant que cette contrainte est violée.

```
Données:
    Clause de départ (u*)
    liste de variables instanciées dans u*(list)
    Contrainte des coûts réduits x̄^g, c̄^g,
    gap de la contraintes mère (gapg)

S = u*;
if(gapg < 0) {
    while(list n'est pas vide) {
        sortir j le premier de la liste;
        if( x̄_j^g est hors base et S_j == 1- x̄_j^g ) {
            if(gapg + c̄_j^g < 0) {
                S_j=*;
                gapg = gapg + c̄_j^g ;
            }
        }
    }
}
```

Figure 5. Algorithme de remontée ou Waning phase

4.6 Choix des littéraux associés aux obstacles

Comme nous l'avons dit précédemment, la famille $\mathcal{F}$ contient un ensemble de clauses C_k *k=1,…,m* représentant des obstacles de l'exploration. Toute clause *v* qui est l'extension d'au moins une clause de $\mathcal{F}$ vérifie *oracle*(v)$\leq BI$. L'ajout d'une nouvelle clause C_k à la famille $\mathcal{F}$ est conditionné par le choix d'un littéral w_k appartenant à C_k de manière à ce que le prochain nœud de l'exploration $u = (\cup_{i=1}^{m}(C_i - \{w_i\}) \cup \{\overline{w}_1, \overline{w}_2, ..., \overline{w}_m\}$ ne soit l'extension d'aucune clause de $\mathcal{F}$. Le choix du littéral w_k peut donc influer de manière conséquente sur l'efficacité de *Resolution Search* car il correspond à fixer la variable x_k à la valeur $\overline{w}_k$ dans le prochain nœud de l'exploration. Notre politique de branchement consiste à choisir le littéral w_k correspondant à la variable x_k ayant le plus gros coût réduit au nœud racine afin de fixer cette variable à l'opposé de sa valeur dans l'obstacle. Cette heuristique de choix est basée sur le principe qu'un variable de fort coût réduit a plus de chances d'être responsable d'un échec dû à la contrainte (2) qu'une variable de faible coût réduit. L'idée globale est de chercher une complémentarité entre la fonction *obstacle* qui cherche les échecs au plus tôt et la famille $\mathcal{F}$ qui utilise ces échecs pour diriger la recherche vers les espaces les plus prometteurs.

5 RÉSULTATS EXPÉRIMENTAUX

Les tests expérimentaux ont été réalisés sur les instances de la `OR-Library` produites par Beasley (1990) (http://people.brunel.ac.uk/~mastjjb/jeb/info.html). Cette collection contient des instances de sac à dos multidimensionnel 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 α =1/4, 1/2 et 3/4. Les coûts de la fonction objectif (c_j) sont générés en corrélation avec les a_{ij} tels que $c_j = \sum_{i=1}^{m} a_{ij} / m + 500 d_j$ 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 différent : α = 1/4 pour les instances 0 à 9, α = 1/2 pour les instances 10 à 19 et α = 3/4 pour les instances 20 à 29. La table 1 détaille les résultats obtenus sur les instances à 5 contraintes et 500 variables. La description des données par colonne est la suivante :

- nom : nom de l'instance.
- α : ratio de l'instance.
- z^* : valeur de la borne inférieure.
- $t^{lb}(h)$: temps mis pour le calcul de la borne inférieure par algorithme $Fix + RL^{tabou}$ (en heures).
- z^{opt} : valeur optimale de l'instance.
- N : nombre de nœuds (simplexes calculés).
- k^{opt} : nombre d'objets dans la solution optimale.
- $t^{opt}(h)$: temps mis pour le calcul de l'optimum (en heures).
- $t^{total}(h)$: temps total = $t^{lb}(h) + t^{opt}(h)$.
- $t^{cpx}(h)$: temps mis par CPLEX pour le calcul de la solution optimale (en heures), ERROR s'il n'a pas trouvé la solution du à une consommation excessive de mémoire.

Les valeurs en gras indiquent : soit que notre algorithme a résolu l'instance en un temps plus court, soit qu'il a prouvé l'instance alors que CPLEX a généré un message d'erreur.

Notre algorithme a résolu toutes les instances à 100 variables avec 5, 10 et 30 contraintes, celles à 500 variables et 5 contraintes et celles à 250 variables avec 5 et 10 contraintes. Le tableau 1. expose les résultats obtenus sur les instances à 5 contraintes et 500 variables, les temps de résolution sont comparables à ceux de CPLEX : 1/3 des instances sont résolues en un temps de calcul inférieur mais CPLEX ne parvient pas à résoudre l'instance cb5.500_8.

On constate que notre méthode se trouve être particulièrement efficace sur les instances fortement corrélées généralement plus difficiles à résoudre. Pour les instances à 500 variables et 5 contraintes : 7/10 sont prouvées en un temps plus rapide que CPLEX pour $\alpha = 0.25$, 3/10 pour $\alpha = 0.5$ et seulement 2/10 pour $\alpha = 0.75$,. Ce résultat est probablement dû à la politique de branchement de notre algorithme qui consiste à chercher l'erreur en priorité lors de l'exploration. Cette politique se trouve être pénalisée sur les instances plus faciles mais semble plus efficace sur les instances difficiles.

Instance		lb		opt						CPLEX
nom	α	z^*	$t^{lb}(h)$	z^{opt}	N	gap($\times 10^2$)	k^{opt}	$t^{opt}(h)$	$t^{total}(h)$	$t^{cpx}(h)$
cb5.500_0	0.25	120114	0,13	120148	8955346	0.028	147	1,89	**2,02**	2,22
cb5.500_1	0.25	117844	0,16	117879	5812286	0.029	148	1,2	1,36	**0,23**
cb5.500_2	0.25	121105	0,25	121131	1694480	0.021	144	0,29	**0,54**	1,3
cb5.500_3	0.25	120785	0,13	120804	2071869	0.015	149	0,41	**0,54**	1,68
cb5.500_4	0.25	122291	0,11	122319	2070893	0.022	147	0,48	0,59	**0,43**
cb5.500_5	0.25	121984	0,16	122024	3603536	0.032	153	0,67	**0,83**	1,02
cb5.500_6	0.25	119117	0,18	119127	1931570	0.008	145	0,33	**0,51**	1,92
cb5.500_7	0.25	120551	0,16	120568	1634778	0.014	150	0,27	**0,43**	0,74
cb5.500_8	0.25	121566	0,14	121586	4430453	0.016	149	0,81	**0,95**	ERROR
cb5.500_9	0.25	120663	0,21	120717	19194184	0.044	151	4,63	4,84	**1,59**
cb5.500_10	0.5	218411	0,17	218428	2563068	0.007	267	0,45	0,62	**0,4**
cb5.500_11	0.5	221118	0,16	221202	18558504	0.037	265	5,02	5,18	**0,25**
cb5.500_12	0.5	217523	0,23	217542	7898428	0.008	264	1,52	1,75	1,03
cb5.500_13	0.5	223534	0,23	223560	3366637	0.011	263	0,65	**0,88**	1,39
cb5.500_14	0.5	218966	0,13	218966	150903	0	267	0,02	0,15	**0,13**
cb5.500_15	0.5	220520	0,18	220530	1298392	0.004	262	0,25	**0,43**	0,86
cb5.500_16	0.5	219973	0,14	219989	815864	0.007	266	0,17	**0,31**	**0,33**
cb5.500_17	0.5	218194	0,18	218215	1093395	0.009	265	0,21	0,39	**0,3**
cb5.500_18	0.5	216976	0,16	216976	709211	0	262	0,14	**0,3**	0,65
cb5.500_19	0.5	219689	0,17	219719	5028042	0.013	267	1,06	1,23	**0,7**
cb5.500_20	0.75	295828	0,21	295828	87509	0	383	0,01	0,22	**0,01**
cb5.500_21	0.75	308078	0,15	308086	723190	0.002	384	0,13	0,28	**0,13**
cb5.500_22	0.75	299788	0,20	299796	336419	0.002	385	0,06	0,26	**0,05**
cb5.500_23	0.75	306476	0,15	306480	230011	0.001	384	0,03	**0,18**	0,32
cb5.500_24	0.75	300340	0,15	300342	662502	0	385	0,11	0,26	**0,05**
cb5.500_25	0.75	302565	0,16	302571	246272	0.001	385	0,04	**0,2**	0,27
cb5.500_26	0.75	301327	0,12	301339	168570	0.003	385	0,02	0,14	**0,1**
cb5.500_27	0.75	306430	0,13	306454	498057	0.007	383	0,09	0,22	**0,04**
cb5.500_28	0.75	302809	0,17	302828	1184776	0.006	384	0,2	0,37	**0,1**
cb5.500_29	0.75	299904	0,11	299910	1249228	0.002	378	0,18	0,29	**0,13**

Tableau 1: Résultats obtenus sur les instances à 5 contraintes et 500 variables de la `OR-Library`

CONCLUSION

Notre procédure basée sur *Resolution Search* et sur la prise en compte des coûts réduits nous permet de prouver l'optimalité d'instances difficiles de MKP. La prise en compte de la contrainte des coûts réduits semble bien adaptée à *Resolution Search* car elle permet d'une part de couper la recherche lors de la phase de descente et d'autre part d'identifier les obstacles de manière efficace lors de la phase de remontée. Expérimentalement, notre méthode se trouve être particulièrement performante sur les instances fortement corrélées. Elle parvient à résoudre toutes les instances à 100 variables, celles à 5 et 10 contraintes et 250 variables et celles à 5 contraintes et 500 variables de la Or-Library.

REFERENCES

Balas E.(1965) An Additive Algorithm for Solving Linear Programs with Zero-One Variables. *Operations Research 13* 517-546.

Beasley J.E. (1990) Or-library : Distributing test problems by electronic mail. *J. Operational Research Society 41* :1069_1072.

Chvátal V. (1997) Resolution search. *Discrete Applied Mathematics*, 73 :81-99.

Dantzig G.B.(1957) Discrete Variables Problems. *Operations Research 5* 266-277.

Demassey S. (2003), Méthodes Hybrides de Programmation par Contraintes et Programmation Linéaire pour le Problème d'Ordonnancement de Projet à Contraintes de Ressources. *Thèse de doctorat, Université d'Avignon.*

Fayard D., Plateau G. (1982) An algorithm for the solution of the 0–1 knapsack problem. *Computing 28* (3):269–287.

Fréville A., Plateau G. (1994) An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem. *Discret Appl Math 49*:189–212

Gavish B. and Pirkul H.(1985) Efficient Algorithms for Solving Multiconstraint Zero-One Knapsack Prob-

lems to Optimality. *Mathematical Programming* 31 78-105.

Gilmore P. C. and Gomory R. E.(1966), The Theory and Computation of Knapsack Functions. *Operations Research 14* 1045-1075.

Ginsberg M. (1993) Dynamic Backtracking. *Journal Of Artificial Intelligence Research*, 1:25-46.

Glover F. and L. Tangedahl (1976) Dynamic Strategies for Branch and Bound. *Omega*, vol. 4, no. 5, 571-576.

Hanafi S., Glover F. (2002) Resolution search and dynamic branch-and-bound. *Journal of Combinatorial Optimization*, 6(4) :401-423

James RJW., Nakagawa Y. (2005) Enumeration methods for repeatedly solving multidimensional knapsack-sub-problems. *Technical report E88-D*, 10:83-103, *The Institute of Electronics, Information andCommunication Engineers*

Martello S., Toth P. (1988), A new algorithm for the 0–1 knapsack problem. *Management Sciences 34* 633–644.

Oliva C, Michelon P, Artigues C (2001) Constraint and linear programming: using reduced costs for solving-the zero/one multiple knapsack problem. In *International conference on constraint programming, CP 01, proceedings of the workshop on cooperative solvers in constraint programming* (CoSolv 01), pp 87-98

Palpant M. (2005) Recherche exacte et approchée en optimisation combinatoire schémas d'intégration et applications. *Thèse de doctorat, Université d'Avignon*

Saunders RM, Schinzinger R (1970) A shrinking boundary algorithm for discrete system models. *IEEE Trans Syst Sci Cybern* 6:133–140

Shih W (1979) A Branch and Bound Method for the Multiconstraint Zero-one Knapsack Problem, *J. Opl. Res. Soc. 30* 369-378.

Vasquez M, Vimont Y (2005) Improved results on the 0-1 multidimensional knapsack problem. *European Journal of Operations Research* 165(1):70–81

Vimont Y., Boussier S. and Vasquez M. (2007) Reduced costs propagation in an efficient implicit enumeration for the 01 multidimensional knapsack problem. To appear in *Journal of Combinatorial Optimization.*