Une approche hybride pour le sac à dos multidimensionnel en 0-1

Michel Vasquez
LGI2P-EMA,
Parc Scientifique G. Besse
30035 Nîmes Cedex 1
email: `vasquez@eerie.fr`

Jin-Kao Hao
LERIA Université d'Angers,
2 bd Lavoisier
49045 Angers Cedex 1
email: `hao@info.univ-angers.fr`

Résumé

Nous présentons, dans cet article, une heuristique hybride pour la résolution du sac à dos multidimensionnel en variables 0-1. Cette approche combine la programmation linéaire et la méthode tabou. L'algorithme ainsi obtenu améliore de manière significative les meilleurs résultats connus sur des instances jugées difficiles.

1 Introduction

Le sac à dos multidimensionnel en variables bivalentes (MKP01) permet de modéliser une grande variété de problèmes où il s'agit de maximiser un profit tout en ne disposant que de ressources limitées. Son expression formelle est la suivante :

$$\text{MKP01} \begin{cases} \text{maximiser } c.x \text{ avec} \\ A.x \leq b \text{ et } x \in \{0,1\}^n \end{cases}$$

avec $c \in \mathbb{N}^{*n}$, $A \in \mathbb{N}^{m \times n}$ et $b \in \mathbb{N}^m$. Les coordonnées binaires x_j du vecteur x sont des variables de décision : $x_j = 1$ si l'objet j est retenu dans le sac, 0 sinon. c_j est le profit (ou gain) associé à l'objet j. Les éléments (a_{ij}) de la matrice A représentent, pour chaque type de ressource i, la consommation de l'objet j. Enfin les b_i sont les quantités disponibles de chacune des m ressources. L'objectif est de trouver un sous ensemble d'objets qui maximise le profit tout en respectant les contraintes de limitation de ressources.

Le fait qu'on le rencontre dans des domaines d'application aussi différents que l'économie [LS55], l'industrie [Dan57, Toy75], les transports, le chargement de cargaison [BS62, Shi79] et l'informatique répartie [GP82], etc. lui confère un grand intérêt pratique. Ce problème d'optimisation combinatoire sous contraintes est cependant NP-difficile [GJ79]. Sa résolution revêt donc, aussi, un caractère théorique toujours d'actualité.

2 Stratégies de résolution

Afin de situer notre approche dans le contexte général des méthodes de résolution, nous présentons ici, quelques uns des axes de recherche explorés.

2.1 Méthodes exactes

Il existe un algorithme pseudo polynomial pour résoudre le MKP01 par la programmation dynamique [BS62, GM85]. Sa complexité $O(n\prod_{i=1}^{m}(b_i+1))$ le rend très vite impraticable en fonction de m et des b_i.

Les algorithmes d'évaluation et séparation se distinguent essentiellement par le calcul de borne qu'ils mettent en œuvre pour élaguer l'arbre de recherche. Un premier algorithme de *branch & bound* [Shi79] propose de prendre, comme borne supérieure pour l'évaluation, la valeur $min\{z_1(k),...,z_m(k)\}$ où $z_i(k)$ est l'optimum du sac à dos ne contenant que la i^{eme} contrainte au nœud k de l'arborescence. La variable de branchement étant alors celle qui correspond à l'élément fractionnaire de ce problème de sac à dos unidimensionnel. L'avantage de cette méthode est qu'elle ne traite que des sacs à dos simples. En revanche la borne utilisée est assez imprécise. L'estimation, plus élaborée, de la valeur potentielle d'un branchement à partir de multiples relaxations (Lagrangienne, composite...) et du calcul des multiplicateurs duaux [GP85] donne de meilleurs résultats que le précédent algorithme avec toutefois un accroissement sensible de la complexité de calcul. D'une manière générale les algorithmes d'énumération implicite se heurtent à une combinatoire en $O(2^n)$.

Pour élargir leur domaine d'application à des instances de taille plus grande on peut d'une part chercher de bons minorants par des heuristiques de construction efficaces, et d'autre part réduire la taille du problème par fixation de variables [GM85], élimination de contraintes [FP86] ou encadrement du nombre de variables à l'optimum [FP93]. Ces dernières méthodes font appel aux aspects les plus théoriques de la programmation linéaire.

2.2 Programmation linéaire

Pour utiliser la programmation linéaire il faut supprimer le caractère binaire des variables x_j. La relaxation des contraintes d'intégrité est une technique omniprésente dans les méthodes de résolution de MKP01. Notre approche ne déroge pas à cette règle. Nous introduisons donc, ici, le problème associé à MKP01 suivant :

$$\text{MKP}\begin{cases}\text{maximiser } c.x \text{ avec}\\ A.x \leq b \text{ et } x \in [0\text{–}1]^n\end{cases}$$

Les composantes du vecteur x sont, dans le cas de MKP, comprises dans l'intervalle continu [0–1]. MKP se résout efficacement par la méthode du simplexe [PTVF92]. Il fournit une borne supérieure $\bar{z}$ qui, associée aux coûts réduits des variables hors base, permet d'introduire une technique de fixation de variables. On retrouve cette technique dans des algorithmes exacts [GM85] et approchés [BM80, LG98].

2.3 Méthodes incomplètes

Plusieurs algorithmes approchés exploitent le tableau du simplexe pour binariser les composantes de l'optimum $\bar{x}$ de MKP. Une des techniques consiste à faire entrer les variables d'écart en base [BM80]. Ce principe a été repris et intégré à la métaheuristique tabou [AJ94]. Par ailleurs on retrouve les valeurs des variables duales à l'optimum de MKP comme multiplicateur u du vecteur colonne a_j des consommations de l'objet j, dans les critères de type ratio $c_j/(u.a_j)$, pour le choix d'une variable x_j. Ce ratio $profit/ressource$ intervient, sous différentes variantes, dans la majorité des heuristiques fondées sur des mécanismes de transformation locale de la solution courante [CB98, DV93, GK96, HF98]. Parmi ces dernières nous soulignons les algorithmes génétique (AG) de Chu et Beasley [CB98] et tabou de Hanafi et Fréville [HF98] qui obtiennent les résultats parmi les meilleurs sur des instances difficiles de MKP01. Considérons cependant l'instance de MKP01 suivante :

$$\text{exemple } (n=5, m=1) \left\{ \begin{array}{lcl} c & = & (12 \quad 12 \quad 9 \quad 8 \quad 8) \\ A & = & (11 \quad 12 \quad 10 \quad 10 \quad 10) \quad b = 30 \end{array} \right.$$

Un algorithme glouton guidé par l'heuristique du ratio $profit/ressource$ (ou simplement celle du meilleur $profit$) pour ajouter des objets dans le sac construira la solution $x = (1,1,0,0,0)$ qui vaut $z = 24$ alors que la solution binaire optimale est $\dot{x} = (0,0,1,1,1)$ pour $\dot{z} = 25$. Cet exemple n'est pas une démonstration, mais il illustre l'hypothèse suivante. Le rôle de la transformation locale est mineur devant celui de la stratégie globale d'exploration de l'espace de recherche. Cette vision globale est naturelle pour un algorithme génétique ; elle correspond à la population de vecteurs x impliqués par les opérateurs de croisement, de sélection et de mutation, qui caractérisent cette métaheuristique (une centaine d'individus sont gérés simultanément dans [CB98]). Elle n'est pas systématique pour un algorithme tabou. Elle correspond, dans ce cas , à une alternance de phases d'intensification et de diversification de la recherche. Dans [HF98] cette alternance de phases est renforcée par une stratégie d'oscillations entre la zone de faisabilité ($A.x \leq b$) et la zone d'infaisabilité ($\exists i \in [1,m] \,|\, a_i.x > b_i$).

3 Approche hybride simplexe/tabou

3.1 Recherche locale RL

Nous résumons, dans ce paragraphe, les éléments de terminologie que le lecteur retrouvera par la suite. La configuration x : c'est le vecteur binaire $(x_1, \cdots, x_n)$. $\mathcal{S}$ représente l'espace de recherche. Il peut-être égal à $\{0,1\}^n$ si l'on considère toutes les configurations possibles $x \in \mathcal{S}$ réalisables ou non. Le voisinage de x : $\mathcal{N}(x)$ est le sous-ensemble des éléments de $\mathcal{S}$ accessibles depuis l'élément x en une seule itération. L'attribut est la valeur affectée à une composante. Dans le cas binaire on peut identifier sans ambiguïté un attribut à l'indice j de la composante qui change de valeur. Enfin le mouvement correspond au passage de la RL du point x à un de ces voisins x' : $mvt(x,x') \,|\, x' \in \mathcal{N}(x)$. On peut aussi écrire $mvt(i_1, i_2, ..., i_k)$ où les indices i_j sont ceux des composantes de x qui sont complémentées dans x'. Un tel mouvement est un k_change.

3.2 Principe général de notre algorithme

L'idée maîtresse de notre approche est de chercher autour de l'optimum $\bar{x}$ de MKP. Notre hypothèse est que les points binaires proches de $\bar{x}$ sont de bonne qualité. Le vecteur $\bar{x}$ contient l'information globale qui guidera notre processus de RL tout en contrôlant le mécanisme de visite du voisinage $\mathcal{N}(x)$ d'une configuration courante x. Pour ce faire nous limitons la RL aux seuls points $x' \in \mathcal{S}$ tels que $distance(x', \bar{x}) \leq \delta_{max}$. La distance géométrique intervient dans le mécanisme de mouvement. Le vecteur *ou point* $\bar{x}$ peut-être considéré comme une information à caractère stratégique pour la RL. Si l'on reprend l'exemple de la section 2.3, on obtient $\bar{z} = 30.3$ et $\bar{x} = (1, 1, \frac{7}{10}, 0, 0)$ alors que, rappelons-le, $\dot{z} = 25$ pour $\dot{x} = (0, 0, 1, 1, 1)$. De ce point de vue cette heuristique n'est donc pas plus convaincante que celle du ratio $profit/ressource$. Faisons pourtant la remarque suivante : toute(s) solution(s) de MKP01 vérifie(nt) l'égalité : $\sum_{j=1}^{n} x_j = k \in \mathbb{N}$. Si nous ajoutons cette égalité comme contrainte au problème MKP nous obtenons une série de problèmes du type :

$$\text{MKP}[k] \begin{cases} \text{maximiser } c.x \text{ avec} \\ A.x \leq b \text{ et } x \in [0\text{–}1]^n \text{ et} \\ \sigma(x) = k \in \mathbb{N} \end{cases}$$

où $\sigma(x)$ est la somme des composantes du vecteur x. Nous disposons donc de plusieurs points $\bar{x}_{[k]}$ pour explorer l'espace de recherche $\mathcal{S}$. Cette approche n'est pas isolée. En effet, de manière très contemporaine, l'ajout de la contrainte d'égalité a été mis en œuvre pour réduire l'espace de recherche dans le cas du sac à dos multidimensionnel multiobjecif [GF00]. Illustrons cette heuristique sur l'exemple de la section 2.3.

$$\begin{array}{llcll} \text{MKP}[1] & \rightarrow & \bar{z}_{[1]} = 12 & \text{et} & \bar{x}_{[1]} = (1, 0, 0, 0, 0) \\ \text{MKP}[2] & \rightarrow & \bar{z}_{[2]} = 24 & \text{et} & \bar{x}_{[2]} = (1, 1, 0, 0, 0) \\ \text{MKP}[3] & \rightarrow & \bar{z}_{[3]} = 25 & \text{et} & \bar{x}_{[3]} = (0, 0, 1, 1, 1) \end{array}$$

Son comportement semble plus prometteur et nous pouvons énoncer l'idée générale de notre algorithme hybride : **dans chaque hyperplan $\sigma(x) = k$ lancer un processus de *RL autour* du point $\bar{x}_{[k]}$**. Nous allons donc alterner, dans le processus d'optimisation, une phase de programmation linéaire PL avec une phase de recherche locale tabou RL^{tabou}.

4 Phase recherche locale

4.1 Réduction de l'espace de recherche

Nous allons dans cette section spécifier un sous ensemble $\mathcal{X} \subset \mathcal{S}$ dans lequel *naviguera* notre processus de recherche locale. Cette réduction de $\mathcal{S}$ reprend les idées que l'on vient d'énoncer :

1. limitation de $\mathcal{S}$ à une *sphère* d'un rayon fixé autour du point $\bar{x}_{[k]}$ solution optimale de MKP$[k]$,
2. conservation du nombre d'objets retenus dans les configurations x, produites par RL^{tabou} autour du point $\bar{x}_{[k]}$, à la valeur constante k (intersection de $\mathcal{S}$ avec l'hyperplan $\{\sigma(x) = k\}$).

Pour le point 1 nous utilisons la distance δ définie, pour x et x' binaires ou continus, par la formule $\delta(x,x') = \sum_{j=1}^{n} |x_j - x'_j|$. L'heuristique pour estimer la distance maximale δ_{max} autorisée depuis le point $\bar{x}_{[k]}$, est la suivante : soit $(1,1,\cdots,1,r_1,\cdots,r_q,0,\cdots,0)$ les composantes triées par ordre décroissant du vecteur $\bar{x}_{[k]}$. Les r_j sont les composantes fractionnaires de $\bar{x}_{[k]}$ et l'on a : $1 > r_1 \geq r_2 \geq \cdots \geq r_q > 0$.

Nous considérons que le mécanisme de RL peut *choisir*, dans le pire des cas, les objets qui correspondent à ces composantes plutôt que ceux qui correspondent aux composantes à 1. La figure suivante illustre cette configuration limite. u est le nombre de composantes

FIG. 1 – *heuristique de calcul de* $\delta_{[k]} = \delta(x_{limite}, \bar{x}_{[k]})$

x_{limite}	$\rightarrow$	0	.	0	1_1	1_2	.	.	.	.	.	1_k	0	.	0
$\bar{x}_{[k]}$	$\rightarrow$	1	.	1	1	1	r_1	r_2	.	.	.	r_q	0	.	0
$\delta_{[k]}$	$=$	$u+q-k$			$+$			$\sum_{j=1}^{q}(1-r_j)$							

à 1 dans $\bar{x}_{[k]}$, $u+q$ est donc le nombre de composantes non nulles de $\bar{x}_{[k]}$. De plus $\sigma(\bar{x}_{[k]}) = k \Rightarrow \sum_{j=1}^{q} r_j = k - u \Rightarrow \delta_{[k]} = 2 \times (u+q-k) \in 2 \times \mathbb{N}$. Dans le cas limite où $u = k$ nous obtenons $\delta_{[k]} = 0$ ce qui est tout à fait logique puisque dans ce cas $\bar{x}_{[k]}$ est entièrement binaire. Pratiquement, et selon les instances, nous prendrons : $\delta_{max} \approx \delta_{[k]}$. Chaque processus RL^{tabou} lancé autour de $\bar{x}_{[k]}$ a donc son propre espace de recherche $\mathcal{X}_k$:

$$\mathcal{X}_k = \{x \in \{0,1\}^n \,|\, \sigma(x) = k \,\wedge\, \delta(x, \bar{x}_{[k]}) \leq \delta_{max}\}$$

Notons ici que les $\mathcal{X}_k$ sont disjoints. Cela réduit à 0 les risques de redondance d'exploration des processus RL^{tabou} qui peuvent donc s'exécuter indépendamment.

4.2 Voisinage

Le mouvement le plus simple pour respecter la contrainte $\sigma(x) = k$ correspond au retrait d'un objet et l'ajout d'un autre. Nous ne dépasserons pas ce cas dont la complexité est déjà en $O((n-k) \times k)$. L'ensemble $\mathcal{N}(x)$ des voisins d'une configuration x est donc défini par la formule suivante :

$$\mathcal{N}(x) = \{x' \in \mathcal{X}_k | \delta(x,x') = 2\}$$

Le mouvement induit par ce voisinage est un cas particulier de *2_change* (c.f. définitions § 3.1). Nous noterons indifféremment $mvt(x,x')$ et $mvt(i,j)$ avec $x'_i = 1 - x_i$ et $x'_j = 1 - x_j$.

4.3 Gestion dynamique de la liste tabou

La méthode d'élimination inverse (MEI), proposée par Fred Glover [Glo90], permet de définir le statut tabou d'un mouvement de manière exacte. Son principe consiste à mémoriser dans une liste (*running list*) les attributs des mouvements effectués. Pour savoir si un nouveau mouvement est tabou il faut parcourir la *running list* à l'envers. Ce faisant on construit une autre liste, la séquence d'annulation résiduelle ou SAR dans laquelle soit on recopie les attributs de la *running list* (y compris ceux du nouveau mouvement), s'ils n'y sont pas déjà, soit on les en enlève. La condition $SAR = \emptyset$ implique que le nouveau mouvement nous ramène à un point déjà visité. Le lecteur trouvera plus de détails sur cette méthode dans [DV93, Glo90]. La complexité de la MEI est en $O(iter^2)$.

Pour notre voisinage à k constant nous parcourons une seule fois la *running list* et chaque fois que $|SAR| = 2$ nous rendons tabou le mouvement qui implique les attributs SAR_0 et SAR_1. Soit *iter* le compteur de mouvements. L'algorithme suivant est associé à la proposition logique :

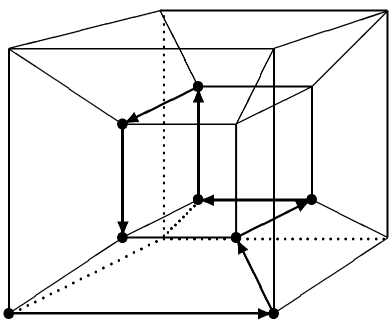
$$iter = tabou[i][j] \Longleftrightarrow mvt(i,j)\, tabou.$$

Algorithme 1 : MAJ_TABOU

```
i = frl % indice de fin de la running list
répéter
    j = running list[i]
    si j ∈ SAR alors
        SAR = SAR ⊖ j
    sinon
        SAR = SAR ⊕ j
    si |SAR| = 2 alors
        tabou[SAR0][SAR1] = iter
        tabou[SAR1][SAR0] = iter
    i = i − 1
jusqu'à i < 0
```

Le principe de la liste stricte peut provoquer le blocage de la RL dans une *impasse* x, tous les points $x' \in \mathcal{N}(x)$ ayant déjà été visités. La figure suivante illustre ce cas pour $\mathcal{S} = \{0,1\}^4$.

FIG. 2 – *blocage avec 1_change dans* $\{0,1\}^4$



En 7 itérations, avec un voisinage à *1_change*, le chemin parcouru aboutit à une impasse alors que $\mathcal{S}$ n'a pas été complètement visité.

Nous contournons cet écueil par une remise à zéro de la *running list*. Toutefois, pour que ce mécanisme n'entraine pas un cyclage systématique, nous imposons que les configurations x visitées après cette remise à zéro soient d'une valeur $c.x = z > z_{min}$ où z_{min} représente la valeur de la meilleure configuration réalisable rencontrée jusque là. Evidemment pour que celà soit possible sans tomber dans le cadre d'une simple *descente* les configurations non réalisables $(\exists i \in [1, m] \,|\, a_i.x > b_i)$ sont admises. Nous introduisons donc une mesure du niveau d'infaisabilité $v_b(x) = \sum_{i \,|\, a_i x > b_i} (a_i.x - b_i)$, que nous chercherons à minimiser au cours de la RL. A chaque fois que $v_b(x)$ vaut 0 nous effaçons la *running list* et nous mettons à jour z_{min}. Voici, pour finir, la version définitive du voisinage :

$$\mathcal{N}(x) = \{x' \in \mathcal{X}_k | (\delta(x, x') = 2) \wedge (c.x' > z_{min}) \wedge (mvt(x, x')\, non\, tabou)\}$$

4.4 Fonction d'évaluation et heuristique de mouvement

Nous précisons, dans cette section, comment à partir d'un point x, nous choisissons parmi les points $x' \in \mathcal{N}(x)$ le point y qui fera l'objet du mouvement $mvt(x, y)$. La fonction d'évaluation d'une configuration x a deux composantes : $v_b(x)$ et $z(x) = c.x$. La première composante est prioritaire sur la seconde. L'heuristique du choix d'un voisin de x est donc :

$$y = \left\{ \begin{array}{l} x' \in \mathcal{N}(x) \,|\, \forall\, x'' \in \mathcal{N}(x) \\ (v_b(x') < v_b(x'')) \vee (v_b(x') = v_b(x'') \wedge z(x') \geq z(x'')) \end{array} \right.$$

En cas d'égalité de la fonction d'évaluation le choix est aléatoire.

4.5 Configuration initiale

Pour amorcer le processus RL^{tabou} il faut construire un x_{init} qui appartienne à $\mathcal{X}_k$. Nous allons montrer qu'il suffit, pour cela, de choisir les k objets correspondant aux plus fortes composantes de $\bar{x}_{[k]}$. Reprenons le schéma de la section 4.1 et construisons une suite de points (x_i) qui part de x_{limite} tout en décalant, à chaque étape, les k composantes à 1 vers la gauche : *pour ne pas surcharger les notations nous avons abusé de l'indexation : x_i représente un vecteur complet et non pas, comme partout ailleurs, la i^{eme} composante du vecteur x.* Comme on peut le voir sur la figure 3 la distance de x_1 à $\bar{x}_{[k]}$ est égale à celle de x_0 plus $(r_q - 1)$ soit à $\delta_{[k]} + (r_q - 1)$. Par définition r_q est strictement inférieur à 1 donc x_1 s'est rapproché de $\bar{x}_{[k]}$. On a la relation de récurrence : $\delta_i = \delta(x_i, \bar{x}_{[k]}) = \delta_{i-1} + (r_{q-i+1} - 1) = \delta_{i-1} + \epsilon_i$ avec $\forall i\, \epsilon_i < 0$. La suite des distances (δ_i) correspondant à la suite (x_i) est strictement décroissante depuis la valeur $\delta_{[k]}$ ce

FIG. 3 – *construction de $x_{(0)}$*

$\bar{x}_{[k]}$	$\rightarrow$	1	.	1	1	1	r_1	r_2	.	.	.	r_q	0
$x_0 = x_{limite}$	$\rightarrow$	0	.	0	1_1	1_2	.	.	.	.	.	1_k	0
x_1	$\rightarrow$	0	.	1_1	1_2	.	.	.	.	.	1_k	0	0
...	$\rightarrow$	.	.	.	.	.	.	.	.	.	.	.	.
$x_{init} = x_{u+q-k}$	$\rightarrow$	1_1	1_2	.	.	.	.	.	1_k	0	.	0	0

qui garantit : $\delta_{u+q-k} < \delta_{[k]}$. Nous pouvons prendre comme configuration initiale x_{init} le point x_{u+q-k} qui appartient bien à $\mathcal{X}_k$.

4.6 Algorithme RL^{tabou}

L'algorithme RL^{tabou} n'optimise pas directement MKP[k]. Il traite une suite de problèmes de décision du type :

$$\exists ?\, x \in \{0,1\}^n \,|\, [c.x \geq z_{min}] \wedge [A.x \leq b] \wedge [\sigma(x) = k].$$

où (z_{min}) est une suite positive strictement croissante. Nous notons $|R.L.|$ la taille de la *running list*, c'est le nombre maximum d'itérations sans amélioration de la fonction d'évaluation.

Algorithme 2 : RL^{tabou}

$iter = 0$ % Compteur de mouvements
$frl = 0$; $tabou[n][n] \leftarrow (-1)$; $x \leftarrow x_{init}$
si $\upsilon_b(x) = 0$ **alors** $z_{min} = z(x)$; $x^* \leftarrow x$ **sinon** $z_{min} = 0$; $x^* \leftarrow (0)$
répéter
    $z' = 0$; $\upsilon_{min} = \infty$
    pour $i \,|\, x_i = 1$ **faire**
        pour $j \,|\, x_j = 0$ **faire**
            si $tabou[i][j] \neq iter$ **alors**
                $(x_i, x_j) = (0, 1)$ %On évalue le mouvement $mvt(i,j)$
                si $\delta(x, \bar{x}_{[k]}) \leq \delta_{max} \wedge z(x) > z_{min}$ **alors**
                    si $(\upsilon_b(x) < \upsilon_{min}) \vee (\upsilon_b(x) = \upsilon_{min} \wedge z(x) > z')$ **alors**
                        $(i', j') = (i, j)$
                        $z' = z(x)$; $\upsilon_{min} = \upsilon_b(x')$
                $(x_i, x_j) = (1, 0)$
    si $z' \neq 0$ **alors**
        $(x_{i'}, x_{j'}) = (0, 1)$ %On effectue le mouvement $mvt(i', j')$
        si $\upsilon_{min} = 0$ **alors**
            $frl = 0$ %On efface la *running list*
            $z_{min} = z(x)$
            $x^* = x$
        sinon
            $iter = iter + 1$
            $running\ list = running\ list \oplus i' \oplus j'$; $frl = frl + 2$
            MAJ_TABOU
jusqu'à $(z' = 0) \vee (frl \geq |R.L.|)$

S'il ne trouve pas la faisabilité, cet algorithme retourne le vecteur nul. Le parcours des composantes du vecteur x se fait dans un ordre aléatoire.

5 Phase simplexe

Il nous reste à déterminer, pour une instance donnée, les valeurs de k intéressantes pour RL^{tabou}. Fréville et Plateau [FP93] proposent un algorithme pour l'encadrement du nombre de variables à l'optimum de MKP01 qui fait appel à plusieurs outils de la programmation linéaire. Ne disposant pas du code de cet algorithme, et pour rester homogène à notre approche PL/RL^{tabou}, nous avons développé un algorithme dont le principe est d'utiliser RL^{tabou} pour obtenir un minorant z, puis résoudre par le simplexe les deux problèmes suivants :

$$\text{MKP}\sigma min[z] \begin{cases} \text{minimiser } \sigma(x) \text{ avec} \\ A.x \leq b \text{ et } x \in [0\text{–}1]^n \text{ et} \\ c.x \geq (z+1) \end{cases}$$

Soit $\sigma min[z]$ la valeur optimale de ce problème. Si nous prenons moins d'objets que $k_{min} = \lceil \sigma min \rceil$ la contrainte $c.x \geq (z+1)$ ne sera plus vérifiée.

$$\text{MKP}\sigma max[z] \begin{cases} \text{maximiser } \sigma(x) \text{ avec} \\ A.x \leq b \text{ et } x \in [0\text{–}1]^n \text{ et} \\ c.x \geq (z+1) \end{cases}$$

soit $\sigma max[z]$ la valeur optimale de ce problème. Si nous prenons plus d'objets que $k_{max} = \lfloor \sigma max \rfloor$ l'une au moins des contraintes $A.x \leq b$ ne sera plus vérifiée. En conséquence, les seules valeurs de k intéressantes pour notre processus de recherche locale sont celles comprises entre k_{min} et k_{max}. Nous calculons donc, par l'algorithme du simplexe, les $(k_{max} - k_{min} + 1)$ MKP$[k]$ qui nous fournissent les points cherchés pour la phase RL^{tabou}.

6 Résultats

Pour les instances que nous présentons ici, RL^{tabou} a été exécuté avec les 10 germes (0..9) de la fonction standard *srand()*. La taille de la *running list* ($|R.L.|$) est fixée à 100000. Le nombre maximum de mouvements sans amélioration de la fonction d'évaluation est donc de 50000.

La première série de tests est constituée des 7 dernières instances proposées par Glover et Kochenberger [GK96]. La colonne $[k]$ indique l'intervalle des points intéressants pour la RL, $\bar{z}$ est l'optimum de MKP.

GK	$n \times m$	$[k]$	$\bar{z}$	T_{HF}	z^*	k^*	$iter^*$	$sec.^*$
18	100 × 25	58..64	4545.79	4524	**4528**	61	3683	10
19	100 × 25	49..55	3886.80	3866	**3869**	51	3144	9
20	100 × 25	61..74	5198.64	5177	**5180**	70	2080	5
21	100 × 25	40..46	3219.92	3195	**3200**	42	1465	4
22	100 × 25	31..37	2544.03	2521	**2523**	34	512	2
23	200 × 15	119..126	9245.67	9231	**9235**	123	16976	131
24	500 × 25	116..125	9080.45	9062	**9070**	119	9210	268

Jeux Glover et Kochenberger : GK18↔GK24

La colonne THF indique les résultats obtenus par l'algorithme tabou de Hanafi et Fréville [HF98] que l'on peut comparer avec la valeur z^* de la meilleure configuration x^*, à k^* objets, trouvée par RL^{tabou}. Nous améliorons strictement tous ces résultats. Les colonnes *iter** et *sec.** sont le nombre de mouvements et le temps en secondes que l'algorithme a mis pour trouver x^*. RL^{tabou} a été lancé sur des machines aussi diverses que PII350, PIII500, ULTRASPARC5 et 30. Les temps *sec.** sont donc donnés seulement à titre indicatif. Nous savons que l'algorithme a effectué 50000 itérations après *iter**. Cette *fin* de processus d'optimisation a pris en moyenne 380 secondes pour les jeux GK18 à GK22, 600 secondes pour GK23 et 1100 secondes pour GK24. Pour ce dernier jeu l'ensemble des phases PL a duré 200 secondes.

La seconde série concerne les instances les plus importantes ($n = 500$, $m = 30$) proposées par Chu et Beasley [CB98].

r	$[k]$	$\bar{z}$	AG_{CB}	z^*	k^*	$iter^*$	$sec.^*$
0	128..133	116619.01	115868	**115950**	130	17841	397
1	125..131	115370.13	114667	**114810**	128	104866	2264
2	125..132	117342.45	116661	**116683**	128	73590	1203
3	125..131	115946.40	115237	**115301**	128	71820	1587
4	125..133	117079.29	116353	**116435**	127	75909	1784
5	128..134	116377.55	115604	**115694**	131	33391	684
6	126..132	114689.65	113952	**114003**	128	107994	2851
7	125..132	114847.83	114199	**114213**	129	87593	1503
8	125..132	115902.61	115247	**115288**	127	75243	1495
9	125..132	117668.77	116947	**117055**	129	39044	869
10	249..254	218601.52	217995	**218068**	251	6828	116
11	249..254	215074.71	214534	**214562**	251	89201	2478
12	248..253	216401.07	215854	**215903**	250	60074	1311
13	249..255	218350.48	217836	**217910**	251	50732	1121
14	248..254	216094.51	215566	**215596**	251	62524	1262
15	250..257	216327.35	215762	**215842**	253	34201	633
16	250..256	216376.30	215772	**215838**	252	54476	1003
17	250..257	217014.09	216336	**216419**	253	40683	947
18	250..257	217839.18	217290	**217305**	253	64489	1475
19	250..256	215218.48	214624	**214671**	252	18531	368
20	373..378	302038.76	301627	**301643**	375	1298	17
21	372..379	300455.00	299985	**300055**	374	78278	1532
22	373..379	305501.21	304995	**305028**	375	64926	1161
23	373..379	302456.21	301935	**302004**	375	26901	1110
24	374..379	304901.35	304404	**304411**	376	20483	333
25	371..377	297409.44	296894	**296961**	374	31403	462
26	372..377	303765.88	303233	**303328**	373	43398	757
27	374..380	307402.50	306944	**306999**	376	33810	1366
28	373..379	303605.92	303057	**303080**	374	17647	350
29	373..379	301020.63	300460	**300532**	376	6948	150

Jeux Chu et Beasley : CB500.30.r

De façon similaire à la section précédente, la colonne AG_{CB} représente les meilleures valeurs obtenues par l'algorithme génétique de Chu et Beasley [CB98]. Nous améliorons de manière significative tous les résultats. Le temps moyen des 50000 dernières itérations est de 1200 secondes pour ces instances.

Nous finissons cette section par une comparaison synthétique, sur ces 30 dernières instances, des moyennes des écarts à la borne $\bar{z}$ (optimum du programme relaxé dans le continu).

$[r]$	α	AG_{CB}	RL^{tabou}
0..9	1/4	0.00612	0.00552
10..19	1/2	0.00264	0.00239
20..29	3/4	0.00166	0.00149

Moyennes des $(\bar{z} - z^*)/\bar{z}$ pour CB.500.30.r

Ce tableau donne un rapide aperçu du gain en qualité des solutions produites par notre approche hybride. α est le taux de débordement des ressources : $b_i / \sum_{j=1}^{n} c_{ij}$.

7 Conclusion

L'idée d'échantillonner $\{0,1\}^n$ autour d'optima de $[0\text{–}1]^n$ s'est avérée performante. Notre heuristique hybride, programmation linéaire/recherche locale, améliore de manière significative les derniers résultats connus sur des instances difficiles [CB98, HF98].

Ce travail peut apporter une contribution dans le cadre des méthodes exactes et cela à deux titres :

- réduction de l'espace de recherche par encadrement du nombre de variables à l'optimum : la colonne $[k]$ des tableaux de résultats donnent les valeurs potentiellement intéressantes pour l'optimum ;
- fixation de variables à partir d'un minorant z de bonne qualité, sa configuration x et les coûts réduits des variables hors base [BM80, GM85] : si l'on applique ce principe sur l'instance GK024 à 500 variables, due à Glover et Kochenberger [GK96] ; PL/RL^{tabou} nous fournit $x \mid c.x = 9070$. On peut alors fixer 38 variables à 0.

Plusieurs voies d'amélioration sont encore à explorer :

- une étude plus précise sur la distance δ_{max} qui limite l'espace de recherche autour des points continus $\bar{x}_{[k]}$ augmentera l'efficacité de RL^{tabou} ;
- la mise en œuvre d'un mécanisme de relance permettra éventuellement de trouver des solutions de meilleure qualité ;
- la prise en compte, dans une version distribuée, d'une population d'optima locaux, pour intégrer de nouvelles contraintes aux programmes relaxés MKP$[k]$ puis, par le simplexe, générer de nouveaux points continus pour relancer la RL.

L'idée la plus intéressante semble bien être de transformer le schéma à sens unique PL/RL^{tabou}, qui correspond à une simple alternance entre le simplexe et tabou par une relation réactive $PL \leftrightarrow RL^{tabou}$ plus riche qui construirait, de manière dynamique, un problème relaxé dont la solution continue serait plus attractive pour la recherche locale.

Voici donc de nouvelles perspectives pour la résolution approchée du sac à dos multidimensionnel en variables bivalentes, qui nous l'espérons, apporteront des résultats encore meilleurs dans un proche avenir.

Remerciements

Nous tenons à remercier Saïd Hanafi pour nous avoir procuré les jeux de la série GK18 $\leftrightarrow$ GK24 sur lesquels il a travaillé ainsi que pour ses amples commentaires sur la méthode d'élimination inverse.

Références

[AJ94] R. Aboudi et K. Jörnsten. Tabu Search for General Zero-One Integer Programs using the Pivot and Complement Heuristic. *ORSA Journal of Computing*, 6(1) :82–93, 1994.

[BM80] E. Balas et C.H. Martin. Pivot and Complement a Heuristic for 0-1 Programming. *Management Science*, 26 :86–96, 1980.

[BS62] R. Bellman et D. Stuart. *Applied Dynamic Programming*. Princeton University Press, 1962.

[CB98] P.C. Chu et J.E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristic*, 4 :63–86, 1998.

[Dan57] G.B. Dantzig. Discrete-variable extremum problems. *Operations Research*, 5 :266–277, 1957.

[DV93] F. Dammeyer et S. Voß. Dynamic tabu list management using the reverse elimination method. *Annals of Operations Research*, 41 :31–46, 1993.

[FP86] A. Fréville et G. Plateau. Heuristic and reduction methods for multiple constraints 0-1 linear programming problems. *European Journal of Operational Research*, 24 :206–215, 1986.

[FP93] A. Fréville et G. Plateau. Sac à dos multidimensionnel en variable 0-1 : encadrement de la somme des variables à l'optimum. *Recherche Opérationnelle*, 27(2) :169–187, 1993.

[GF00] X. Gandibleux et A. Fréville. The multiobjective tabu search method customized on the 0/1 multiobjective knapsack problem : the two objectives case. *Journal of Heuristics*, 2000. à paraître.

[GJ79] M. Garey et D. Johnson. *Computers & Intractability A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[GK96] F. Glover et G.A. Kochenberger. Critical event tabu search for multidimensional knapsack problems. In I.H. Osman J.P. Kelly, editor, *Metaheuristics : The Theory and Applications*, pages 407–427. Kluwer Academic Publishers, 1996.

[Glo90] F. Glover. Tabu search. *ORSA Journal of Computing*, 2 :4–32, 1990.

[GM85] M. Gondran et M. Minoux. *Graphes & algorithmes*. Eyrolles, 1985.

[GP82] B. Gavish et H. Pirkul. Allocation of data bases and processors in a distributed computting system. *Management of Distributed Data Processing*, 31 :215–231, 1982.

[GP85] B. Gavish et H. Pirkul. Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Mathematical Programming*, 31 :78–105, 1985.

[HF98] S. Hanafi et A. Fréville. An efficient tabu search approach for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 106 :659–675, 1998.

[LG98] J.S. Lee et M. Guignard. An approximate algorithm for multidimensional zero-one knapsack problems a parametric approach. *Management Science*, 34(3) :402–410, 1998.

[LS55] J. Lorie et L.J. Savage. Three problems in capital rationing. *Journal of Business*, 28(3) :229–239, 1955.

[PTVF92] W.H. Press, S.A. Teukolsky, W.T. Vetterling et B.P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 1992.

[Shi79] W. Shih. A branch & bound method for the multiconstraint zero-one knapsack problem. *Journal of the Operational Research Society*, 30 :369–378, 1979.

[Toy75] Y. Toyoda. A simplified algorithm for obtaining approximate solutions to zero-one programming problem. *Management Science*, 21(12) :1417–1427, 1975.