RAIRO Operations Research
RAIRO Oper. Res. **35** (2001) 415-438

UNE APPROCHE HYBRIDE POUR LE SAC À DOS MULTIDIMENSIONNEL EN VARIABLES 0–1

Michel Vasquez[1] et Jin-Kao Hao[2]

Communiqué par Gérard Plateau

Abstract. We present, in this article, a hybrid approach for solving the 0–1 multidimensional knapsack problem (MKP). This approach combines linear programming and Tabu search. The resulting algorithm improves on the best result on many well-known hard benchmarks.

Résumé. Nous présentons, dans cet article, une approche 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.

Mots clés : Sac-à-dos multidimensionnel, programmation linéaire, recherche tabou.

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 :

$$\text{MKP01} \left\{ \begin{array}{l} \text{maximiser } c.x \\ \text{s.c. } A.x \leq b \text{ et } x \in \{0,1\}^n \end{array} \right.$$

Reçu en mars 2000.

[1] LGI2P, Parc Scientifique Georges Besse, 30035 Nimes Cedex 1, France ;
e-mail : vasquez@site-eerie.ema.fr

[2] LERIA, Université d'Angers, 2 bd Lavoisier, 49045 Angers Cedex 1, France ;
e-mail : Jin-Kao.Hao@univ-angers.fr

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 [25], l'industrie [9, 18, 32], les transports, le chargement de cargaison [4, 31] et l'informatique répartie [16], etc. lui confère un grand intérêt pratique. Ce problème d'optimisation combinatoire sous contraintes est cependant NP-difficile [15]. Sa résolution revêt donc, aussi, un caractère théorique toujours d'actualité.

Ainsi de nombreux travaux ont été effectués sur le sujet [1,2,5,7,8,10,11,17,20, 23, 24, 31, 32] et beaucoup d'heuristiques sont encore explorées pour proposer des solutions, sinon exactes, du moins approchées à ce problème. En effet les méthodes exactes sont limitées à de petites instances ou à des instances ayant des caractéristiques bien spécifiques : matrice A creuse et quasi unimodulaire, redondance de contraintes etc. Aujourd'hui, la résolution des instances les plus difficiles de MKP01 se réalise donc essentiellement avec des approches heuristiques. L'aptitude de ces dernières à fournir des solutions de bonne qualité les rend indispensables dans le domaine pratique. Elles s'avèrent aussi très utiles pour le développement d'algorithmes exacts fondés sur des méthodes d'évaluation et séparation.

Nous proposons, dans ce papier, une approche hybride qui combine la programmation linéaire et la recherche locale tabou. L'idée fondamentale de notre approche consiste à explorer l'espace de recherche, d'une manière contrôlée, autour d'optima continus de problèmes relaxés. Ainsi nous nous distinguons des approches fondées sur le critère de choix classique du ratio *profit/ressource*. Pour mettre en œuvre cette approche, nous proposons d'abord une méthode permettant de déterminer le nombre d'objets à l'optimum de MKP01. Nous développons ensuite un algorithme de recherche locale intégrant un voisinage original et une implémentation efficace de la technique d'élimination inverse pour la gestion de la liste tabou.

Nous validons cette approche sur des benchmarks classiques ainsi que des benchmarks récents réputés difficiles [7, 20]. Nous montrons que cette approche permet non seulement de retrouver les meilleurs résultats connus pour l'ensemble des jeux expérimentés, mais aussi d'améliorer de manière significative les meilleurs résultats pour un grand nombre de jeux les plus difficiles [7, 23].

Notre exposé s'organise comme suit. Après un état de l'art sur les principales stratégies de résolution (Sect. 2), nous donnons le principe général de notre approche hybride (Sect. 3). Nous développons ensuite la phase de la programmation

linéaire (Sect. 4), ainsi que la phase de la recherche locale tabou (Sect. 5). Ces développements sont alimentés par la présentation des résultats expérimentaux (Sect. 6), ainsi qu'une analyse sur les performances de notre algorithme (Sect. 7). Nous terminons par une discussion sur de nombreuses perspectives.

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 polynômial pour résoudre le MKP01 par la programmation dynamique [4, 21]. 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* [31] 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 $i^{\text{ème}}$ 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 que l'on sait bien résoudre [28]. En revanche la borne utilisée est assez imprécise. Dans l'exemple suivant :

$$\text{exemple } 1(n = 4, m = 2) \begin{cases} c = \begin{pmatrix} 4 & 5 & 4 & 4 \end{pmatrix} \\ A = \begin{pmatrix} 1 & 3 & 3 & 2 \\ 3 & 3 & 2 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 3 \\ 3 \end{pmatrix} \end{cases}$$

$z_1 = 8$ pour $x_1 = (1, 0, 0, 1)$ et $z_2 = 8$ pour $x_2 = (0, 0, 1, 1)$, alors que la solution optimale qui tient compte des deux contraintes, est $\bar{x} = (0, 1, 0, 0)$ et vaut $\bar{z} = 5$. Non seulement $\bar{z}$ est relativement loin de $min\{z_1, z_2\}$, mais l'information sur les composantes de $\bar{x}$ est complètement fausse.

L'estimation, plus élaborée, de la valeur potentielle d'un branchement à partir de multiples relaxations (lagrangienne, composite...) et du calcul des multiplicateurs duaux [17] 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 [21], élimination de contraintes [11] ou encadrement du nombre de

variables à l'optimum [12]. Ces dernières méthodes font appels aux aspects les plus théoriques de la programmation linéaire.

2.2. Méthodes approchées

Plusieurs algorithmes approchés combinent la programmation linéaire avec une heuristique dont l'objectif est de rendre entières des variables continues. En effet, 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. Elle conduit à la résolution du problème associé à MKP01 suivant :

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

Les composantes du vecteur x sont, dans le cas de MKP, comprises dans l'intervalle continu [0–1]. MKP se résout efficacement[3] par la méthode du simplexe [30].

La résolution de MKP fournit une borne supérieure $\bar{z}$ qui, associée aux coûts réduits des variables hors base, permet d'introduire une technique de fixation de variables [21]. Une des techniques consiste à faire entrer les variables d'écart en base [2]. Ce principe a été repris et intégré à des approches heuristiques [1, 24]. Enfin, Løkketangen et Glover ont réalisé plusieurs implantation d'algorithmes tabou exploitant le tableau du simplexe [26, 27].

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 [7, 8, 20, 23].

Parmi ces dernières nous soulignons les algorithmes génétique (AG) de Chu et Beasley [7] et tabou de Hanafi et Fréville [23] qui obtiennent les résultats parmi les meilleurs sur des instances difficiles de MKP01. Ces deux algorithmes se distinguent par la métaheuristique mise en œuvre pour contrôler l'exploration de l'espace de recherche (AG/tabou) ainsi que par une petite variante dans le mécanisme de transformation des configurations x visitées par le processus d'optimisation. Cette opération correspond à l'algorithme bien connu Drop-Add suivant :

[3] C'est ici l'aspect pratique qui nous intéresse, il est vrai qu'en théorie le comportement du simplexe peut être exponentiel.

Algorithme 1 : DROP-ADD

Enlever:

[1] $i^* = arg\max\left\{\left(\sum_{j=1}^{n} a_{ij}.x_j\right)/b_i,\ i \in [1,m]\right\}$

[2] $j^* = arg\min\{c_j/a_{i^*j} \mid x_j = 1,\ j \in [1,n]\}$

$x_{j^*} \leftarrow 0$

Ajouter:

Faire

$k^* = arg\max\left\{c_k \mid x_k = 0 \text{ et } \forall i\ \left[a_{ik} + \sum_{j=1}^{n}(a_{ij}.x_j)\right] \leq b_i,\ k \in [1,n]\right\}$

$x_{k^*} \leftarrow 1$

tant que k^* *existe*

Les lignes 1 et 2 se retrouvent dans l'algorithme [23] (ainsi que dans [8]...). La ligne 1 identifie la contrainte la plus saturée. C'est une façon de s'affranchir du calcul du vecteur multiplicateur u tout en conservant le critère du ratio *profit/ressource*. Dans [7] cette technique est remplacée par

$$j^* = arg\min\{c_j/(u.a_j) \mid x_j = 1,\ j \in [1,n]\}$$

où u correspond aux coûts marginaux des m ressources de MKP.

Considérons cependant l'instance de MKP01 suivante :

$$\text{exemple 2 } (n=5, m=1)\begin{cases} c &= \begin{pmatrix}12 & 12 & 9 & 8 & 8\end{pmatrix} \\ A &= \begin{pmatrix}11 & 12 & 10 & 10 & 10\end{pmatrix} \quad b = 30. \end{cases}$$

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 [7]). 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 [23] 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 ($\exists i \in [1,m] \,|\, a_i.x > b_i$).

Pour finir ce survol des métaheuristiques appliquées à MKP01, nous citerons le recuit simulé [10], ainsi qu'une étude des méthodes de bruitage [5] fondées sur les travaux de Charon et Hudry [6]. Les références [5, 10] ne présentent cependant que des résultats sur des instances de petites tailles. Enfin, Fréville et Plateau ont proposé une heuristique, qui combine des calculs de coefficients duaux avec des fixations de variables, une élimination de contraintes et une procédure d'énumération, pour résoudre efficacement des instances à 2 contraintes [13].

3. Principe général de notre algorithme

Tenant compte des remarques faites sur l'exemple 2 de la section 2.2, notre étude sur la résolution approchée de MKP01 par la recherche locale (RL) s'est plus particulièrement focalisée sur les deux points suivants :

1. remise en cause de l'heuristique de mouvement (ou critère de choix d'un voisin de la configuration courante) ;
2. recherche d'un moyen d'enrichir le simple processus de RL par une stratégie globale.

Nous avons trouvé un principe commun pour répondre à ces deux points d'investigation.

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 : $distance(x', \bar{x}) \leq \delta_{\max}$ [4]. 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 recherche locale.

Sur l'exemple 2 (Sect. 2.2) le simplexe donne $\bar{z} = 30.3$ 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 notre heuristique n'est donc pas plus convaincante que celle du ratio $profit/ressource$. Enfin si l'on peut considérer $\bar{x}$ comme un élément de connaissance à apporter au processus de RL, son unicité en réduit passablement le caractère global.

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 :

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

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, directement inspirée par les travaux de Fréville et Plateau [12], n'est pas isolée. En

[4] Nous préciserons en section 5.2 les notions *distance* et $\delta_{\max}$.

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 multiobjectif [14]. Illustrons cette heuristique sur ce même exemple de la section 2.2.

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

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}. Chaque séquence PL/RL^{tabou} pourra être distribuée sur plusieurs machines.

La mise en œuvre de ce principe se décompose en trois étapes :

- détermination des valeurs de k intéressantes ;
- utilisation du simplexe pour calculer les $\bar{x}_{[k]}$ correspondants ;
- exécution de la recherche locale tabou autour de ces points.

La section suivante décrit la phase simplexe, ou programmation linéaire PL, qui traite les deux premières étapes que nous venons de citer.

4. Phase simplexe

Le processus RL^{tabou} effectue un échantillonnage discret autour de l'optimum de MKP[k]. Notre hypothèse est qu'en tant que tel il ne fera pas mieux que la valeur $\bar{z}_{[k]}$ de la solution continue de ce programme. La première étape de notre approche consiste donc à trouver les valeurs de k qui fournissent des bornes supérieures $\bar{z}_{[k]}$ à un minorant z donné.

Fréville et Plateau [12] 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. Nous avons développé un algorithme plus simple dont le principe est d'utiliser RL^{tabou} pour obtenir un minorant z, puis résoudre par le simplexe les deux problèmes suivants MKPσmin[z] et MKPσmax[z] :

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

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.

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

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}.

La figure 1 résume le déroulement de cette *phase simplexe* ou PL sur l'instance CB30.250.10 due à Chu et Beasley [7]. Le programme RL^{tabou} a besoin d'un point pour amorcer la recherche, cela explique la ligne 1 de cet exemple.

1. résoudre MKP $\rightarrow \bar{x}$, $\bar{z} = 108258.07$ et $\sigma(\bar{x}) = 125.65$,
2. lancer RL^{tabou} autour de $\bar{x} \rightarrow z = 107611$,
3. $\sigma \min[z] = 122.17 \rightarrow k_{\min} = 123$,
4. $\sigma max[z] = 128.56 \rightarrow k_{\max} = 128$,
5.

k	123	124	125	126	127	128
$\bar{z}_{[k]}$	107811.87	108055.23	108212.76	**108248.62**	108138.39	107802.92

FIGURE 1. Phase PL pour CB30.250.10.

Remarquons enfin que la meilleure valeur $\bar{z}_{[k]} = 108248.62$ obtenue par les $(k_{\max} - k_{\min} + 1)$ MKP$[k]$ est plus fine, en tant que borne supérieure de MKP01, que $\bar{z} = 108258.07$ produite par la résolution de MKP.

5. PHASE RECHERCHE LOCALE

5.1. DÉFINITIONS

Avant de détailler notre algorithme tabou 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' : $mvt(x, x') \,|\, x' \in \mathcal{N}(x)$. En reprenant ce qui vient d'être dit sur les attributs 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*.

5.2. 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 $\delta_{\max}$ 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 :

$$1 > r_1 \geq r_2 \geq \cdots \geq r_q > 0.$$

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 2 illustre cette configuration limite. u est le nombre de composantes à 1 dans $\bar{x}_{[k]}$, $u + q$ est donc le nombre de composantes non nulles de $\bar{x}_{[k]}$[5]. De plus $\sigma(\bar{x}_{[k]}) = k \Rightarrow \sum_{j=1}^{q} r_j = k - u \Rightarrow \delta_{[k]} = 2 \times (u + q - k) \in 2 \times \mathbb{N}$.

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
$\delta_{[k]}$	=	$u + q - k$			+			$\sum_{j=1}^{q}(1 - r_j)$							

FIGURE 2. Heuristique de calcul de $\delta_{[k]} = \delta(x_{\text{limite}}, \bar{x}_{[k]})$.

[5]Notons que $u < k < u + q$ sinon le problème est trivial.

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 $\mathcal{X}_k$:

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

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.

5.3. Voisinage

L'ensemble $\mathcal{N}(x)$ des voisins d'une configuration x est donc défini par la formule suivante :

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

À cause de la contrainte $\sigma(x) = k$ (implicite dans $\mathcal{X}_k$), $\mathcal{N}(x)$ n'est qu'un sous-ensemble des points à distance 2 de x. Sa cardinalité est :

$$|\mathcal{N}(x)| = (n - k) \times k.$$

Le mouvement induit par ce voisinage correspond au retrait d'un objet et l'ajout d'un autre, c'est un cas particulier de 2_*change* (*cf.* définitions Sect. 5.1). Nous noterons indifféremment $mvt(x, x')$ et $mvt(i, j)$ avec $x'_i = 1 - x_i$ et $x'_j = 1 - x_j$. Cette formulation du voisinage va être modifiée à la fin de la section suivante pour tenir compte d'éléments liés à la gestion de la liste tabou.

5.4. Gestion dynamique de la liste tabou

La méthode d'élimination inverse (MEI), proposée par Glover [19], permet de définir le statut tabou d'un mouvement de manière exacte. Cela signifie qu'elle est équivalente à l'enregistrement complet des configurations visitées. Elle est qualifiée de liste stricte [3]. 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. Cette étape est appelée *trace* des attributs du mouvement. Si au cours de cette étape on rencontre la condition $SAR = \emptyset$ alors le nouveau mouvement nous ramène à un point déjà visité. Il faut donc le rendre tabou. La complexité de la MEI est donc $O(iter^2)$.

Dammeyer et Voß [8] ont réalisé une première mise en œuvre de la MEI sur le MKP01, dans laquelle la MEI analyse un nombre variable d'échanges causés

par l'algorithme DROP-ADD. Ce voisinage nécessite la gestion de *traces* à nombre variable d'attributs. Cela accroît la complexité de la procédure. Dans le cas de 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. L'algorithme suivant est associé à la proposition logique : $mvt(i,j)\, tabou \Leftrightarrow tabou[i][j] = iter$.

Algorithme 2 : MAJ_TABOU

$i = frl$ % indice de fin de la *running list*
répéter
  $j =$*running list*$[i]$
  si $j \in SAR$ **alors**
    $SAR = SAR \ominus j$
  sinon
    $SAR = SAR \oplus j$
  si $|SAR| = 2$ **alors**
    $tabou[SAR_0][SAR_1] = iter$
    $tabou[SAR_1][SAR_0] = iter$
  $i = i - 1$
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 $\mathcal{S} = \{0,1\}^4$.

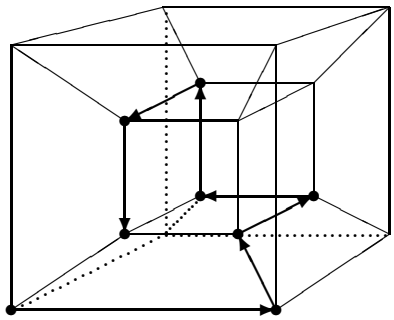

FIGURE 3. Blocage avec 1_change dans $\{0,1\}^4$.

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. Ce phénomène de blocage est d'une très faible probabilité dans $\{0,1\}^n$ avec $n \geq 100$. Nous ne l'introduisons que pour sensibiliser le lecteur à un aspect connexe beaucoup plus gênant : un processus de recherche locale contrôlé par une liste tabou stricte construit un chemin qui peut devenir une barrière entre $\mathcal{N}(x)$ et une zone de $\mathcal{X}_k$ de points potentiellement intéressants. Il faut donc autoriser, de temps en temps, le cyclage par un mécanisme d'aspiration [19, 22].

Nous contournons cet écueil par une remise à zéro de la *running list*. Toutefois, pour que ce mécanisme n'entraîne 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. Évidemment pour que cela 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 :

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

5.5. 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.

5.6. 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 $\bar{x}_{[k]}$, solution optimale de MKP[k] produit par simplexe.

Reprenons le schéma de la section 5.2 et construisons une suite de points (x^i)[6] qui part de x_{limite} tout en décalant, à chaque étape, les k composantes à 1 vers la gauche :

[6]Pour ne pas surcharger le texte nous avons utilisé la notation en exposant : x^i représente un vecteur complet à ne pas confondre à la $i^{\text{ème}}$ composante x_i du vecteur x.

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

FIGURE 4. Construction de x_{init}.

Comme on peut le voir sur la figure 4 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 :

$$\delta_i = \delta(x_i, \bar{x}_{[k]}) = \delta_{i-1} + (r_{q-i+1} - 1) = \delta_{i-1} + \epsilon_i \text{ avec } \forall i\, \epsilon_i < 0.$$

La suite des distances (δ_i) correspondant à la suite (x_i) est strictement décroissante depuis la valeur $\delta_{[k]}$ ce 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 $\mathcal{X}_k$.

5.7. ALGORITHME RL^{tabou}

L'algorithme RL^{tabou} (Algorithme 3) n'optimise pas directement MKP[k]. Il résout[7] une suite de problèmes de décision du type :

$$\text{Existe-t-il } x \in \{0,1\}^n \text{ tel que } \begin{cases} c.x \geq z_{\min} & et \\ A.x \leq b & et \\ \sigma(x) = k & \end{cases}$$

où $(z_{\min})$ est une suite positive strictement croissante. Il explore les zones de l'espace de recherche $\mathcal{X}_k$ pour lesquelles $z > z_{\min}$. Son objectif premier n'est pas de maximiser un critère du type $profit/ressource$ mais de minimiser v_b.

Nous notons $|R.L.|$ la taille de la *running list* qui correspond au nombre maximum d'itérations sans production d'une configuration x réalisable ($A.x \leq b$). Passé cette limite le processus s'interrompt. S'il ne trouve pas de configuration respectant les contraintes de sac, cet algorithme retourne le vecteur nul. Enfin pour des raisons de clarté le facteur aléatoire a été omis. Sa mise en œuvre consiste simplement à parcourir les composantes du vecteur x dans un ordre aléatoire.

Une analyse de la complexité de RL^{tabou} sera faite en section 7 à la suite de la présentation des résultats.

[7] L'objet ici est de trouver une solution mais pas de prouver qu'il n'en existe pas.

Algorithme 3 : RL^{tabou}

```
iter = 0
frl = 0
tabou[n][n] ← (−1)
x ← x_init %Configuration initiale (figure 4)
si v_b(x) = 0 alors z_min = z(x); x* ← x
sinon z_min = 0; x* ← (0)
répéter
    z' = 0; v_min = ∞
    pour i | x_i = 1 faire
        pour j | x_j = 0 faire
            si tabou[i][j] ≠ iter alors
                (x_i, x_j) = (0, 1) %On évalue le mouvement mvt(i, j)
                si δ(x, x̄_[k]) ≤ δ_max ∧ z(x) > z_min alors
                    si (v_b(x) < v_min) ∨ (v_b(x) = v_min ∧ z(x) > z') alors
                        (i', j') = (i, j)
                        z' = z(x); v_min = v_b(x')
                (x_i, x_j) = (1, 0)
    si z' ≠ 0 alors
        (x_i', x_j') = (0, 1) %On effectue le mouvement mvt(i', j')
        si v_min = 0 alors
            frl = 0 %On efface la running list
            z_min = z(x)
            x* = x
        sinon
            iter = iter + 1
            running list = running list ⊕ i' ⊕ j'; frl = frl + 2
            MAJ_TABOU
jusqu'à (z' = 0) ∨ (frl ≥ |R.L.|)
```

6. Résultats

Nous avons expérimenté notre approche sur 3 séries de jeux tests. Pour toutes ces instances RL^{tabou} a été exécuté, de façon distribuée, avec les 10 germes (0..9) de la fonction standard *srand()*. L'algorithme est codé en C et tourne sur des configurations (machine/système) aussi diverses que : (PII350/Win.NT), (PII450/Win.NT), (PIII500/Win.NT), (Ultra Sparc 5/Unix) et (Ultra Sparc 30/Unix). Les temps d'exécution en secondes, qui figurent dans les colonnes *t.t.*, *sec.** et *sec.*, sont donc donnés seulement à titre indicatif.

6.1. Jeux classiques

Nous commençons par les 56 instances classiques que l'on retrouve notamment dans [1, 2, 5, 7, 8, 10–12, 20, 24, 31, 32]. Ces jeux ne sont plus considérés comme difficiles aujourd'hui [7, 23]. Cependant :

- d'une part ils constituent une référence à laquelle une méthode approchée ne peut se soustraire sans arguments *a priori* ;
- d'autre part ils permettront une analyse comparative des valeurs $\delta_{[k]}$ qui fournira, justement, un début d'argumentation pour caractériser *a priori* la difficulté des problèmes.

Nous avons, pour ces jeux, fixé la taille de la *running list* à 2000 ($|R.L.| = 2000$). Les résultats sont présentés dans les tableaux 1 et 2.

TABLEAU 1. Instances classiques.

Pb.	$n \times m$	$[k]$	z^*	k^*	δ^*	$\delta_{[k^*]}$	$iter^*$	$t.t.$
fp1	27×4	14..23	3090	17	4.14	6.00	79	0
fp2	34×4	14..28	3186	23	7.48	6.00	7	1
fp3	19×2	2..6	28642	3	2.73	2.00	1	0
fp4	29×2	11..20	95168	14	2.18	2.00	1	1
fp5	20×10	9..11	2139	10	3.55	8.00	1	0
fp6	40×30	7..12	776	9	5.16	8.00	27	1
fp7	37×30	15..20	1035	17	3.55	8.00	1	1
peter1	6×10	3..4	3800	3	1.16	4.00	0	0
peter2	10×10	3..7	87061	5	4.25	2.00	2	0
peter3	15×10	8..11	4015	9	1.38	2.00	1	0
peter4	20×10	9..11	6120	9	0.07	2.00	0	0
peter5	28×10	15..18	12400	18	2.00	2.00	1	1
peter6	39×5	24..33	10618	27	4.14	6.00	331	9
peter7	50×5	26..40	16537	35	7.34	8.00	852	31
hp1	28×4	15..24	3418	18	4.14	6.00	79	3
hp2	35×4	14..28	3186	23	7.34	8.00	7	9
weing1	28×2	12..15	141278	14	2.00	2.00	1	1
weing2	28×2	10..12	130883	11	2.00	4.00	1	1
weing3	28×2	5..9	95677	6	2.73	2.00	1	2
weing4	28×2	10.17	119337	15	4.78	2.00	127	4
weing5	28×2	6..9	98796	9	0.09	2.00	0	0
weing6	28×2	10..12	130623	11	4.00	2.00	19	0
weing7	105×2	86..90	1095445	87	2.62	4.00	1	8
weing8	105×2	27..36	624319	30	2.18	2.00	1	24
sento1	60×30	18..23	7772	20	5.16	8.00	27	16
sento2	60×30	31..36	8722	33	3.55	8.00	1	10

PL/RL^{tabou} trouve la solution optimale (connue) dans tous les cas ($z^* = \dot{z}$). Les valeurs de la phase PL n'ont pas d'intérêt en tant que bornes puisque l'optimum est connu. Nous indiquons tout de même l'intervalle $[k]$ des hyperplans $\sigma(x) = k$ explorés par RL^{tabou}. Pour cette série et la suivante (à des fins de comparaison) nous donnons la distance δ^* de la meilleure solution x^* à $\bar{x}_{[k^*]}$, ainsi que la valeur $\delta_{[k^*]}$ (Sect. 5.2) à partir de laquelle on estime le rayon de recherche autour du point $\bar{x}_{[k^*]}$. La colonne $iter^*$ correspond au nombre de mouvements pour atteindre x^*. La colonne $t.t.$ représente la somme des temps sur tous les processus dans chaque hyperplan $\sigma(x) = k$.

TABLEAU 2. Instances classiques (suite et fin).

Pb.	$n \times m$	$[k]$	z^*	k^*	δ^*	$\delta_{[k^*]}$	$iter^*$	$t.t.$
weish01	30×5	12..14	4554	12	0.72	2.00	0	2
weish02	30×5	13..15	4536	14	3.28	2.00	3	2
weish03	30×5	12..14	4115	12	1.41	2.00	1	2
weish04	30×5	11..12	4561	12	2.00	2.00	1	1
weish05	30×5	11..12	4514	12	2.00	2.00	1	1
weish06	40×5	20..21	5557	20	3.20	2.00	8	1
weish07	40×5	19..20	5567	20	2.70	2.00	1	2
weish08	40×5	21..22	5605	21	2.55	2.00	1	2
weish09	40×5	15..16	5246	16	0.19	2.00	0	1
weish10	50×5	19..23	6339	21	2.69	2.00	1	5
weish11	50×5	18..20	5643	19	0.82	4.00	0	4
weish12	50×5	20..22	6339	21	2.48	2.00	1	3
weish13	50×5	19..22	6159	21	2.30	2.00	1	4
weish14	60×5	23..26	6954	26	0.63	4.00	0	5
weish15	60×5	26..27	7486	26	2.00	4.00	1	3
weish16	60×5	25..28	7289	26	2.44	2.00	1	4
weish17	60×5	38..41	8633	41	2.00	2.00	1	4
weish18	70×5	39..41	9580	40	2.26	2.00	1	4
weish19	70×5	26..29	7698	27	2.00	2.00	1	7
weish20	70×5	35..37	9450	35	0.31	2.00	0	4
weish21	70×5	32..35	9074	32	2.00	2.00	1	6
weish22	80×5	32..35	8947	33	4.00	2.00	12	7
weish23	80×5	30..33	8344	32	4.00	2.00	461	12
weish24	80×5	45..47	10220	45	0.06	2.00	0	8
weish25	80×5	40..42	9939	40	2.00	2.00	1	7
weish26	90×5	35..39	9584	36	2.00	2.00	1	14
weish27	90×5	37..39	9819	38	1.43	2.00	1	7
weish28	90×5	36..38	9492	37	1.34	4.00	1	9
weish29	90×5	36..38	9410	36	0.02	2.00	0	6
weish30	90×5	50..51	11191	51	2.00	2.00	1	2

Nous ne faisons pas figurer le nombre total d'itérations effectuées par RL^{tabou} puisqu'il peut se déduire par la simple formule :

$$iter^* + \left(\frac{1}{2} \times |R.L.|\right).$$

Nous remarquons qu'en général δ^*, qui est pour ces jeux la distance de la solution optimale à $\bar{x}_{[k]}$, est assez faible. Cela explique la rapidité de notre algorithme à résoudre ces instances. On constate par ailleurs que sur 10 instances ($iter^* = 0$), l'heuristique du plus proche point suffit à construire la solution optimale et qu'il suffit d'un mouvement pour 31 autres instances. Pour ces 41 instances il n'est pas utile de mettre en œuvre la métaheuristique tabou pour trouver l'optimum.

Nous rappelons que *t.t.* est le temps total, pour 10 relances de 1000 itérations (une par germe aléatoire), pour chaque hyperplan : ainsi *t.t.* est-il plus grand pour *weish*26 que pour *weish*23. *weish*26 contient 10 variables de plus et exige l'exploration d' un hyperplan supplémentaire. $iter*$ est le nombre d'itérations, dans l'hyperplan $\sigma(x) = k^*$, pour atteindre la meilleure configuration : sa valeur n'est pas obligatoirement corrélée à celle de *t.t.*

6.2. Jeux Glover et Kochenberger

Cette deuxième série de tests est constituée des 7 dernières instances proposées par Glover et Kochenberger [20]. Le tableau 3 résume les valeurs clés de la phase PL. $\bar{\bar{z}} = \max(\bar{z}_{[k]})$ est la meilleure valeur parmi les optima des programmes $\text{MKP}_{[k]}$. La dernière colonne (sec.) indique les durées cumulées, en secondes, pour la résolution de MKP ainsi que des $\text{MKP}_{[k]}$.

Tableau 3. Phase PL sur GK18 $\leftrightarrow$ GK24.

GK	$n \times m$	$[k]$	$\bar{\bar{z}}$	$\bar{z}$	sec.
18	100×25	58..64	4545.66	4545.79	4
19	100×25	49..55	3886.45	3886.80	4
20	100×25	61..74	5198.54	5198.64	5
21	100×25	40..46	3219.73	3219.92	5
22	100×25	31..37	2544.02	2544.03	4
23	200×15	119..126	9245.53	9245.67	12
24	500×25	116..125	9080.44	9080.45	220

Tableau 4. Phase RL^{tabou} sur GK18 $\leftrightarrow$ GK24.

GK	T_{HF}	T'_{HF}	z^*	δ^*	$\delta_{[k^*]}$	k^*	$iter^*$	sec.*	sec.
18	4524	4526	**4528**	13.55	24.00	61	3683	10	395
19	3866	3867	**3869**	15.29	20.00	51	3144	9	382
20	5177	5179	**5180**	15.95	22.00	70	2080	5	366
21	3195	3197	**3200**	10.78	22.00	42	1465	4	366
22	2521	**2523**	**2523**	14.24	28.00	34	512	2	383
23	9231	9233	**9235**	19.69	16.00	123	16976	131	723
24	9062	9064	**9070**	15.21	22.00	119	9210	268	2027

Le tableau 4 correspond à la phase recherche locale. Pour cette série de problèmes, ainsi que la suivante, la taille de la *running list* est fixée à 100 000. La colonne T_{HF} indique les résultats obtenus par l'algorithme tabou de Hanafi et Fréville [23] que l'on peut comparer avec la valeur z^*, signalée en caractères gras, de la meilleure configuration x^*, à k^* objets, trouvée par RL^{tabou}. La colonne T'_{HF} contient les récentes valeurs obtenues par Hanafi avec son algorithme tabou. Les colonnes sec.* et sec. indiquent respectivement le temps d'obtention de x^* et

le temps total d'exécution d'une phase RL^{tabou}. Nous améliorons strictement la majorité des résultats sur ces instances.

Par rapport aux jeux précédents (Sect. 6.1), le nombre d'itérations et, par voie de conséquence le temps CPU, nécessaires à l'obtention de ces solutions ont augmentés. En effet :

- les nombres n et m, plus grands pour ces instances, interviennent dans la complexité de la fonction d'évaluation du voisinage ;
- $\delta_{[k^*]}$ a augmenté et $C_n^{\delta_{[k^*]}}$ [8] donne une idée de la combinatoire à traiter.

Notre processus doit parcourir un espace de recherche $\mathcal{X}_k$ plus vaste. À échantillonnage équivalent de $\mathcal{X}_k$, il faut plus d'itérations. Comme nous l'avons évoqué au début de la section sur les jeux classiques nous avons, avec les valeurs $\delta_{[k]}$, une mesure expérimentale de la difficulté d'une instance de MKP01.

6.3. Jeux Chu et Beasley

270 instances de MKP01 de tailles allant de 100 à 500 variables et de 5 à 30 contraintes sont proposées, très récemment, par Chu et Beasley [7] et constituent une partie de la OR-Library [9].

Tableau 5. 24 instances CB$m.n.r$.

CB$m.n.r$	$[k]$	$\bar{z}$	AG_{CB}	z^*	k^*	$iter^*$	$sec.^*$
5.100.0	28..31	24585.90	*24381*	*24381*	29	1	1
5.100.10	51..55	42939.52	*42757*	*42757*	52	187	1
5.100.20	75..78	60016.56	*59822*	*59822*	78	0	0
10.100.0	26..29	23480.64	*23064*	*23064*	27	2749	3
10.100.10	50..53	41712.64	*41395*	*41395*	51	173	1
10.100.20	75..78	57626.33	*57375*	*57375*	77	1	1
30.100.0	23..26	22579.07	21946	21946	24	420	1
30.100.10	48..51	41276.36	40767	40767	49	7625	15
30.100.20	73..75	57987.77	57494	57494	73	42	1
5.250.0	71..74	59442.47	*59312*	*59312*	73	252	1
5.250.10	130..135	109220.64	*109109*	*109109*	132	2984	10
5.250.20	189..194	149765.67	*149659*	*149659*	192	9663	31
10.250.0	66..70	59489.34	59187	59187	68	12373	60
10.250.10	126..131	111147.15	110863	**110889**	128	468	2
10.250.20	186..190	152031.39	151790	**151801**	188	24536	99
30.250.0	61..65	57430.15	56693	**56796**	63	2221	14
30.250.10	123..128	108258.07	107689	**107770**	125	65392	1399
30.250.20	186..190	150574.32	150083	**150163**	187	27015	332
5.500.0	144..149	120234.92	120130	**120134**	146	39350	656
5.500.10	265..270	218500.08	218422	**218426**	267	6761	90
5.500.20	381..387	295896.38	295828	295828	383	5012	72
10.500.0	132..139	118019.48	117726	**117746**	134	55794	776
10.500.10	254..260	217552.92	217318	**217343**	256	48536	648
10.500.20	376..382	304555.03	304344	**304350**	379	68008	1321

[8] Nous rappelons que $\delta_{[k^*]}$ est un entier pair (Sect. 5.2).

[9] accessible à l'adresse : `http//mscmga.ms.ic.ac.uk/`.

Nous présentons, dans un premier temps, un tableau synthétique sur 24 instances couvrant toutes les caractéristiques offertes par leurs auteurs ($n \times m \times \alpha$[10]) exceptées $n = 500$ et $m = 30$ qui font l'objet d'un autre tableau. Le format générique du libellé de ces instances est le suivant : CB$m.n.r$ avec $0 \leq r \leq 29$. La colonne $n \times m$ n'est donc plus utile. De même, la valeur α peut se déduire à partir du rang r de l'instance : $0 \leq r \leq 9 \Rightarrow \alpha = 0.25$, $10 \leq r \leq 19 \Rightarrow \alpha = 0.50$ et $20 \leq r \leq 29 \Rightarrow \alpha = 0.75$. De façon similaire à la section précédente, la colonne AG_{CB} représente les meilleures valeurs obtenues par l'algorithme génétique de Chu et Beasley [7]. Les valeurs optimales, lorsqu'elles sont connues, figurent en italique. RL^{tabou} produit bien une solution optimale dans ce cas. Nous améliorons la majorité des résultats pour lesquels cet optimum n'est pas connu (valeurs indiquées en caractères gras).

Nous améliorons également de manière significative les meilleurs résultats obtenus par Chu et Beasley [7] sur les 30 instances les plus importantes de la OR-Library (*cf.* Tab. 6). Sur cette série d'instances, la valeur $\bar{\bar{z}}$ est sensiblement plus fine que la borne $\bar{z}$.

Tableau 6. 30 instances CB30.500.

r	$[k]$	$\bar{\bar{z}}$	$\bar{z}$	AG_{CB}	z^*	k^*	$iter^*$	sec.*
0	128..133	116601.41	116619.01	115868	**115950**	130	17841	397
1	125..131	115365.72	115370.13	114667	**114810**	128	104866	2264
2	125..132	117330.33	117342.45	116661	**116683**	128	73590	1203
3	125..131	115936.14	115946.40	115237	**115301**	128	71820	1587
4	125..133	117078.97	117079.29	116353	**116435**	127	75909	1784
5	128..134	116362.03	116377.55	115604	**115694**	131	33391	684
6	126..132	114682.47	114689.65	113952	**114003**	128	107994	2851
7	125..132	114833.73	114847.83	114199	**114213**	129	87593	1503
8	125..132	115901.29	115902.61	115247	**115288**	127	75243	1495
9	125..132	117661.71	117668.77	116947	**117055**	129	39044	869
10	249..254	218597.06	218601.52	217995	**218068**	251	6828	116
11	249..254	215074.67	215074.71	214534	**214562**	251	89201	2478
12	248..253	216395.94	216401.07	215854	**215903**	250	60074	1311
13	249..255	218345.99	218350.48	217836	**217910**	251	50732	1121
14	248..254	216094.49	216094.51	215566	**215596**	251	62524	1262
15	250..257	216326.72	216327.35	215762	**215842**	253	34201	633
16	250..256	216375.27	216376.30	215772	**215838**	252	54476	1003
17	250..257	217013.40	217014.09	216336	**216419**	253	40683	947
18	250..257	217830.74	217839.18	217290	**217305**	253	64489	1475
19	250..256	215218.07	215218.48	214624	**214671**	252	18531	368
20	373..378	302038.59	302038.76	301627	**301643**	375	1298	17
21	372..379	300453.13	300455.00	299985	**300055**	374	78278	1532
22	373..379	305499.91	305501.21	304995	**305028**	375	64926	1161
23	373..379	302447.30	302456.21	301935	**302004**	375	26901	1110
24	374..379	304895.74	304901.35	304404	**304411**	376	20483	333
25	371..377	297409.08	297409.44	296894	**296961**	374	31403	462
26	372..377	303763.56	303765.88	303233	**303328**	373	43398	757
27	374..380	307397.96	307402.50	306944	**306999**	376	33810	1366
28	373..379	303605.11	303605.92	303057	**303080**	374	17647	350
29	373..379	301014.80	301020.63	300460	**300532**	376	6948	150

[10] α est le taux de débordement des ressources : $b_i / \sum_{j=1}^{n} A_{ij}$.

La taille de la *running list* étant fixée à 100 000, nous savons que l'algorithme a effectué 50 000 itérations après $iter^*$. Nous donnons, dans la section suivante, une analyse de la complexité temporelle de cette *fin* de processus d'optimisation.

Nous terminons cette étude comparative par la synthèse des résultats sur l'ensemble des jeux à 500 variables de la OR-Library. Nous ajoutons les résultats obtenus par Osorio *et al.* [29] lors de récents travaux dont l'objectif était d'améliorer les performances de l'outil de programmation linéaire en nombres entiers CPLEX (*V6.5.2*) ; travaux qui portent sur un algorithme de fixation de variables avec adjonction de coupes (résultats en colonne *Fix+Cuts*). Nous retranscrivons également, en colonne *CPLEX*, les valeurs trouvées par ce logiciel sans les modifications effectuées par les auteurs du rapport [29].

Tableau 7. Moyennes des meilleurs résultats par groupe de 10 instances CBm.500.

m	α	AG_{CB}	*Fix+Cuts*	*CPLEX*	PL/RL	$(\bar{z}-z^*)/\bar{z}$
5	1/4	120616	120610	120619	**120623**	0.0008
	1/2	219503	219504	219506	**219507**	0.0004
	3/4	302355	**302361**	302358	302360	0.0002
10	1/4	118566	118584	118597	**118600**	0.0020
	1/2	217275	217297	217290	**217298**	0.0009
	3/4	302556	302562	302573	**302575**	0.0007
30	1/4	115470	115520	115497	**115547**	0.0055
	1/2	216187	216180	216151	**216211**	0.0024
	3/4	302353	302373	302366	**302404**	0.0015

Fix+Cuts et *CPLEX* sont interrompus après 3 heures de calcul (sur un PIII500) ou lorsque l'arbre de recherche dépasse les 250 Mo d'occupation mémoire. Nous constatons que, plus le nombre de contraintes est élevé, plus la moyenne des écarts à l'optimum continu est grande (dernière colonne), plus notre approche se distingue des autres algorithmes en conservant une avance qualitative significative.

7. Analyse de la complexité

7.1. Voisinage et *running list*

L'estimation du temps CPU pris par RL^{tabou} en fonction du nombre total d'itérations n'est pas facile. En effet la *running list* est remise à zéro à chaque fois que l'on rencontre une configuration réalisable (Sect. 5.4). Étant donné que toute phase RL^{tabou} finit par un nombre constant ($|R.L.|/2$) de mouvements on peut proposer, pour cette étape, la formule suivante :

$$sec. \approx (\mathcal{A} \times iter \times (n-k) \times k \times m) + \left(\mathcal{B} \times \frac{iter \times (iter-1)}{2}\right)$$

où : $\mathcal{A}$ représente le coût moyen, en secondes par itération, du calcul de la fonction d'évaluation du voisinage, $\mathcal{B}$ celui de la mise à jour de la liste tabou. Dans cette formule, le coefficient $\mathcal{A}$ est assez imprécis : en effet nous nous limitons à un rayon $\delta_{\max}$ autour d'un point $\bar{x}_{[k]}$. Nous n'avons donc pas systématiquement la mesure de $z(x)$, mais surtout celle de $v_b(x)$ qui coûte m, à effectuer. Faute de pouvoir évaluer correctement $\mathcal{A}$ et $\mathcal{B}$ nous avons regroupé dans le tableau 8

Tableau 8. Temps de calcul en secondes.

$\|R.L.\|$	n=100 m=5	100 10	100 30	250 5	250 10	250 30	500 5	500 10	500 30
25000	21	22	25	56	59	82	199	222	244
50000	70	73	78	142	148	193	425	479	526
100000	253	262	273	410	424	512	1001	1106	1193
200000	1046	1076	1100	1371	1427	1608	2566	2771	2975

les temps d'exécution sur des instances à $\alpha = 0.50$ pour toutes les combinaisons $n \times m$ et pour 4 tailles de la *running list*. La machine utilisée est un PIII500. Nous avons modifié l'algorithme RL^{tabou} en supprimant la remise à zéro de la *running list* pour avoir un nombre d'itérations directement lié à la taille de cette dernière. Le facteur quadratique dû à la gestion de la liste tabou est assez faible. Le temps maximum est de moins de 20 minutes pour $|R.L.| = 100\,000$. C'est la valeur à rajouter aux temps indiqués dans les colonnes *sec.** des tableaux 5 et 6.

Dans le cas d'une parallélisation des processus tabou avec une machine par germe de *srand()*, ce temps n'est pas prohibitif. Nous pouvons même accroître la taille de la *running list* pour tenter d'améliorer encore les résultats (Tab. 9) : nous

Tableau 9. Augmentation de la *running list*.

	$\|R.L.\| = 100\,000$		$\|R.L.\| = 300\,000$				
CB	δ_1^*	z_1^*	z_2^*	δ_2^*	$iter^*$	$t.t.$	$\delta(x_1^*, x_2^*)$
5.500.10	5.8	218426	**218428**	9.26	173312	6441	14.0
30.500.0	15.93	115950	**115991**	18.61	264408	9760	24.0

constatons un gain par rapport aux valeurs des tableaux 5 et 6. Nous remarquons aussi que les distances δ_2^* aux optima $\bar{x}_{[267]}$ et $\bar{x}_{[130]}$ ont augmenté. En terme de combinatoire la distance $\delta(x_1^*, x_2^*)$ entre la nouvelle solution x_2^* et la précédente x_1^* est relativement importante. Le processus tabou a exploré *plus loin*. Le temps total $t.t.$ (*en secondes*) devient important mais cette évolution nous encourage à travailler sur la complexité de la procédure RL^{tabou}.

7.2. Le paramètre $\delta_{\max}$

Enfin le paramètre $\delta_{\max}$ est un facteur crucial de l'efficacité de notre algorithme. Le tableau 10 illustre bien l'influence de $\delta_{\max}$ à la fois sur la stabilité de RL^{tabou}

vis-à-vis de la discrimination aléatoire des meilleurs candidats dans $\mathcal{N}(x)$ et sur le temps total d'exécution de RL^{tabou}. La colonne g contient la valeur du germe de la fonction *srand()*. Nous soulignons en caractère gras les temps minimums et maximums d'exécution complète de la phase tabou pour chacune des valeurs de $\delta_{\max}$.

TABLEAU 10. Influence de $\delta_{\max}$ sur RL^{tabou} : GK24, $k^* = 119$, $\delta_{[119]} = 22$.

g	$\delta_{\max} = 0.75 \times \delta_{[119]} = 16.5$					$\delta_{\max} = 0.6 \times \delta[119] = 13.2$				
	$iter^*$	sec.*	sec.	δ^*	z^*	$iter^*$	sec.*	sec.	δ^*	z^*
0	67354	2200	4014	10.87	9070	25172	626	2120	10.87	9070
1	26735	763	2581	16.17	9067	68809	1924	**3409**	10.87	9070
2	46175	1373	3171	13.81	9067	30508	779	2297	10.87	9070
3	15882	438	2282	14.75	9067	289	8	**1445**	10.87	9070
4	87218	2983	**4810**	10.87	9070	1754	39	1470	10.87	9070
5	4879	142	1949	16.04	9066	54029	1338	2809	10.87	9070
6	79848	2509	4237	14.90	9068	71834	1838	3290	10.87	9070
7	9210	268	**2027**	15.21	9070	6758	150	1616	10.87	9070
8	41101	1205	2901	14.69	9067	44282	1212	2633	10.87	9070
9	17572	507	2234	10.87	9070	25378	624	2082	10.87	9070

Remarquons toutefois que des valeurs $\delta_{\max}$ trop faibles risquent d'interdire des zones intéressantes de $\mathcal{X}_k$ (*cf.* valeurs de δ_1^* et δ_2^* Tab. 9). Ce paramètre de réglage augmente bien sûr la complexité globale de notre approche. Nous nous sommes limités à $\delta_{\max} = coef \times \delta$ avec $coef \in \{0.75, 1, 1.5, 2\}$ et même exceptionnellement 3 pour certains jeux de la section 6.1 pour lesquels $\delta_{[k^*]}$ était trop petit (voir Tabs. 1 et 2).

8. CONCLUSION

Nous avons mis en œuvre 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 lancer un algorithme tabou. Nous avons introduit une caractéristique intéressante ($\delta_{[k]}$) pour les problèmes MKP01 et proposé une alternative au critère de choix $profit/ressource$ pour le mécanisme de transformation locale d'une configuration x. Nous avons aussi développé une version relativement efficace de la méthode d'élimination inverse dont l'exploitation peut renforcer d'autres algorithmes tabou.

L'idée d'échantillonner $\{0,1\}^n$ autour d'optima de $[0\text{–}1]^n$ s'est avérée très bénéfique et performante. En effet, notre algorithme hybride améliore de manière significative les derniers résultats connus sur des instances difficiles [7, 23].

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 3, 5 et 6 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 [2, 21] : si l'on applique ce principe sur l'instance GK024 à 500 variables, due à Glover et Kochenberger [20], PL/RL^{tabou} nous fournit x, tel que $c.x = 9070$ (Sect. 4). On peut alors fixer 38 variables à 0.

Plusieurs voies d'amélioration sont encore à explorer :

- une étude plus précise sur la distance $\delta_{\max}$ 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 $PL \leftrightarrow RL^{\text{tabou}}$ 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 ↔ GK24 (Sect. 6.2) sur lesquels il a travaillé ainsi que pour ses amples commentaires sur la MEI.

Références

[1] R. Aboudi et K. Jörnsten, Tabu Search for General Zero-One Integer Programs using the Pivot and Complement Heuristic. *ORSA J. Comput.* **6** (1994) 82-93.
[2] E. Balas et C.H. Martin, Pivot and Complement a Heuristic for 0–1 Programming. *Management Sci.* **26** (1980) 86-96.
[3] R. Battiti et G. Tecchiolli, The reactive tabu search. *ORSA J. Comput.* **6** (1994) 128-140.
[4] R. Bellman et D. Stuart, *Applied Dynamic Programming.* Princeton University Press (1962).
[5] P. Boucher et G. Plateau, Étude des méthodes de bruitage appliquées au problème du sac à dos à plusieurs contraintes en variables 0–1, dans *JNPCC'99* 5$^{\text{es}}$ *journées nationales sur la résolution pratique de problèmes NP-complets* (1999) 151-162.

[6] I. Charon et O. Hudry, The noising method: A new method for combinatorial optimization. *Oper. Res. Lett.* **14** (1993) 133-137.
[7] P.C. Chu et J.E. Beasley, A genetic algorithm for the multidimensional knapsack problem. *J. Heuristic* **4** (1998) 63-86.
[8] F. Dammeyer et S. Voß, Dynamic tabu list management using the reverse elimination method. *Ann. Oper. Res.* **41** (1993) 31-46.
[9] G.B. Dantzig, Discrete-variable extremum problems. *Oper. Res.* **5** (1957) 266-277.
[10] A. Drexl, A simulated annealing approach to the multiconstraint zero-one knapsack problem. *Computing* **40** (1988) 1-8.
[11] A. Fréville et G. Plateau, Heuristic and reduction methods for multiple constraints 0–1 linear programming problems. *Eur. J. Oper. Res.* **24** (1986) 206-215.
[12] A. Fréville et G. Plateau, Sac à dos multidimensionnel en variable 0–1 : encadrement de la somme des variables à l'optimum. *RAIRO: Oper. Res.* **27** (1993) 169-187.
[13] A. Fréville et G. Plateau, The 0–1 bidimensional knapsack problem: Toward an efficient high-level primitive tool. *J. Heuristics* **2** (1997) 147-167.
[14] X. Gandibleux et A. Fréville, The multiobjective tabu search method customized on the 0/1 multiobjective knapsack problem: The two objectives case. *J. Heuristics* (à paraître).
[15] M. Garey et D. Johnson, *Computers & Intractability A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company (1979).
[16] B. Gavish et H. Pirkul, Allocation of data bases and processors in a distributed computting system. *Management of Distributed Data Processing* **31** (1982) 215-231.
[17] B. Gavish et H. Pirkul, Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Math. Programming* **31** (1985) 78-105.
[18] P.C. Gilmore et R.E. Gomory, The theory and computation of knapsack functions. *Oper. Res.* **14** (1966) 1045-1074.
[19] F. Glover, Tabu search. *ORSA J. Computing* **2** (1990) 4-32.
[20] F. Glover et G.A. Kochenberger, Critical event tabu search for multidimensional knapsack problems, edité par I.H. Osman et J.P. Kelly, *Metaheuristics: The Theory and Applications*. Kluwer Academic Publishers (1996) 407-427.
[21] M. Gondran et M. Minoux, *Graphes & algorithmes*. Eyrolles (1985).
[22] S. Hanafi, A. El Abdellaoui et A. Fréville, Extension de la Méthode d'Élimination Inverse pour une gestion dynamique de la liste tabou. *RAIRO* (à paraître).
[23] S. Hanafi et A. Fréville, An efficient tabu search approach for the 0–1 multidimensional knapsack problem. *Eur. J. Oper. Res.* **106** (1998) 659-675.
[24] J.S. Lee et M. Guignard, An approximate algorithm for multidimensional zero-one knapsack problems a parametric approach. *Management Sci.* **34** (1998) 402-410.
[25] J. Lorie et L.J. Savage, Three problems in capital rationing. *J. Business* **28** (1955) 229-239.
[26] A. Løkketangen et F. Glover, Solving zéro-one mixed integer programming problems using tabu search. *Eur. J. Oper. Res.* **106** (1998). Special Issue on Tabu Search.
[27] A. Løkketangen et F. Glover, Candidate list and exploration strategies for solving 0/1 mip problems using a pivot neighborhood, dans *Metaheuristics*. Kluwer Academic Publishers (1999).
[28] S. Martello et P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley (1990).
[29] M.A. Osorio, F. Glover et P. Hammer, *Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions*, Technical report. Hearin Center for Enterprise Science. Report HCES-08-00 (2000).
[30] W.H. Press, S.A. Teukolsky, W.T. Vetterling et B.P. Flannery, *Numerical Recipes in C*. Cambridge University Press (1992).
[31] W. Shih, A branch & bound method for the multiconstraint zero-one knapsack problem. *J. Oper. Res. Soc.* **30** (1979) 369-378.
[32] Y. Toyoda, A simplified algorithm for obtaining approximate solutions to zero-one programming problem. *Management Sci.* **21** (1975) 1417-1427.