DOSSIER DE RECHERCHE ET D'ENSEIGNEMENT

de

Vincent BOYER

Table des matières

1. Introduction

Mes activités de recherche portent sur l'étude des problèmes d'optimisation combinatoire ainsi que sur les algorithmes de résolution. J'ai découvert la recherche opérationnelle durant ma formation d'ingénieur en automatique, à l'École Nationale Supérieure d'Électrotechnique, d'Électronique, d'Informatique, d'Hydraulique et des Télécommunications de Toulouse, et j'ai voulu continuer dans ce domaine étant de plus passionné par l'informatique et la programmation.

En parallèle à ma formation d'ingénieur, l'obtention de mon diplôme de DEA en système automatique m'a ouvert les portes vers le doctorat. J'ai fait ma thèse au Laboratoire d'Architecture et d'Analyse des Systèmes (LAAS-CNRS) dans le domaine de la recherche opérationnelle dans le groupe Méthodes et Algorithmes en Commande. J'ai aussi acquis de l'expérience en algorithmique et en programmation, l'ensemble des algorithmes étudiés ayant été programmé en C/C++ sous UNIX et sous Windows.

Je m'intéresse également au calcul distribué et parallèle et notamment à l'utilisation des GPUs à des fins de calcul généraliste. Mon post-doctorat dans le groupe Calcul Distribué et Asynchronisme du LAAS m'a permis d'acquérir de l'expérience dans ce domaine. L'exploitation de la puissance de calcul des GPUs pour résoudre des problèmes d'optimisation combinatoire représente un vrai challenge du fait des structures irrégulières qui en résultent généralement.

Je souhaite continuer mes activités de recherche et d'enseignement dans le domaine de la recherche opérationnelle, de l'informatique et du calcul distribué.

2. Curriculum Vitae

2.1. Situation actuelle

Date de naissance : 21/07/1980
Nationalité : Française

E-mail : vboyer@laas.fr
Tel : 06 11 23 24 75
Adresse : Appt. E05
291 chemin de Tucaut
31270 CUGNAUX

Profession : Post-doctorant
Intitulé : Etude de la communication distribuée dans un réseau de DMEMS
Projet ANR : Smart Surface
Laboratoire : Laboratoire d'Analyse et d'Architecture des Systèmes
Groupe de Recherche : Calcul Distribué et Asynchronisme

2.2. Formation et diplômes

2004-2007 : Doctorat en Système Automatique.
Délivré par : l'Institut National des Sciences Appliquées de Toulouse
Titre : "Contribution à la programmation en nombres entiers"
Soutenu le : 14/12/2007
Lieu : Laboratoire d'Analyse et d'Architecture des Systèmes
École doctorale : École Doctorale Système
Jury :
Rapporteurs/Examinateurs :
- Said Hanafi, Professeur des universités, Université de Valencienne
- Mhand Hifi, Professeur des universités, Université d'Amiens
- Gérard Plateau, Professeur des universités, Université Paris XIII

Directeurs de thèse :
- Jean-Bernard Lasserre, Directeur de recherche, LAAS-CNRS
- Didier El Baz, Chargé de recherche, HDR, LAAS-CNRS
- Moussa Elkihel, Maître de conférence, Université Toulouse III

2001-2004 : Diplôme d'ingénieur en Automatique et Informatique Industrielle.
Diplôme d'Études Approfondies en Système Automatiques : "Méthodes hybrides et/ou mixtes pour la résolution de programmes linéaires en variables 0-1".
Délivré par : l'École Nationale Supérieure d'Électrotechnique, d'Électronique, d'Informatique, d'Hydraulique et des Télécommunications de Toulouse
Mention : bien

1998-2001 : Classe Préparatoire aux Grandes Écoles.

2.3. Expériences

09/07-09/09 : Attaché Temporaire d'Enseignement et de Recherche à l'Université Paul Sabatier de Toulouse.

09/05-09/07 : Moniteur à l'Université Paul Sabatier.

02/04-06/04 : Stage de fin d'étude de 24 semaines au LAAS -CNRS.

06/03-08/03 : Stage de 8 semaines chez ATEXIA (groupe Vinci).
Affectation au service affaire.

06/02-08/02 : Stage de 6 semaines dans le groupe SORAD.
Affectation au bureau d'études.

2.4. Compétences informatique

- Programation : Java et langage C++ sous UNIX et Windows.
  CUDA (programmation sur GPU).
- Outils informatiques : CPLEX - XPRESS - MAPLE 12 - MATLAB (Simulink).
- Systèmes d'exploitation : UNIX et Windows.
- Bureautique : Latex - Office - Star Office.

2.5. Langues parlées

- Français.
- Anglais (score au TOEIC : 865).

3. Activités de recherches

3.1. Introduction

Mes activités de recherche portent sur la résolution de problèmes d'optimisation combinatoire ainsi que sur le calcul parallèle. Mes travaux de thèse se sont intéressés au problème du sac à dos multidimensionnel sur lequel j'ai proposé une heuristique efficace basée sur la programmation dynamique ainsi qu'une méthode coopérative originale combinant programmation dynamique et Branch & Bound. Ensuite, j'ai étudié la résolution du problème du partage équitable (Knapsack Sharing Problem) en utilisant la programmation dynamique et le principe de dominance qui permet de réduire considérablement l'occupation mémoire.

Mon post-doctorat m'a permis d'acquérir des compétences en parallélisme et en calcul distribué. Sur le projet SmartSurface, j'étudie et teste la communication distribuée dans un réseau de DMEMS ce qui a donné lieu à un logiciel de simulation permettant de valider les algorithmes développés. En parallèle à ce travail, je m'intéresse aussi à l'utilisation de la puissance des cartes graphiques actuelles pour la résolution de problèmes d'optimisation. Une première étude a mené à la parallèlisation de la programmation dynamique dense pour le problème du sac à dos sur GPU et a permis d'avoir des accélérations importantes.

3.2. Le problème du sac à dos multidimensionnel

Le problème du sac à dos multidimensionnel, ou Multi knapsack Problem (MKP), peut-être défini comme suit :

$$(MKP) \begin{cases} \max \sum_{j \in N} p_j.x_j = z(MKP), \\ \text{s.c.} \sum_{j \in N} w_{i,j}.x_j \leq c_i \ , \text{i} \in M, \\ x_j \in \{0,1\}, j \in N, \end{cases}$$

p_j, $w_{i,j}$ et c_i, $(i,j) \in N \times M$, étant des entiers positifs.

Le problème du sac à dos multidimensionnel, ou MKP pour Multi Knapsack Problem, est un problème classique d'optimisation combinatoire appartenant à la classe des problèmes NP-complets. Il fait l'objet d'un grand intérêt de la part de la communauté scientifique, les applications dans ce domaine sont en effet très nombreuses et concernent des problèmes de logistique, de productique, d'économie ou de finances. En particulier, les problèmes de sac à dos multidimensionnels interviennent dans les domaines suivants :
– Gestion des chaînes logistiques (problème de répartition),
– Ordonnancement de production,
– Gestion des ressources humaines (emploi du temps),

– Maintenance (service après vente, ordre des visites, minimisation de parcours),
– Gestion des sièges de trains, avions en fonction des tarifs,
– Gestion de ressources (départ, arrivée, placement des avions),
– Gestion des containers dans les ports.
Par ailleurs, le problème MKP intervient comme un sous-problème de nombreux problèmes d'optimisation combinatoire.

Travaux

J'ai proposé des méthodes heuristiques efficaces basées sur la programmation dynamique et la relaxation surrogate, ainsi qu'une méthode coopérative permettant de résoudre de manière exacte le MKP. Mes travaux ont aussi porté sur la nature de la difficulté des problèmes en optimisation combinatoire et j'ai proposé deux techniques pour engendrer aléatoirement des instances difficiles qui se basent sur les problèmes en contraintes égalités et l'analyse de la transformée en Z du MKP (cf. [9]).

La trame générale des algorithmes de résolution étudiés peut-être schématisée comme suit :
– Etape 1 : Relaxation surrogate
– Etape 2 : Programmation dynamique
– Etape 3 : Méthode d'exploration
La relaxation surrogate du problème de départ (MKP) permet d'obtenir un problème à une contrainte. Ce dernier est résolu via un algorithme basé sur la programmation dynamique qui a été modifié afin de garantir la réalisabilité de la solution obtenue pour le problème multi-contraints. Ces deux premières phases donnent lieu à une première heuristique qui peut être complétée par une méthode d'exploration visant à améliorer la valeur de la borne obtenue et même parfois à obtenir une solution optimale.

Du point de vue heuristique, j'ai proposé deux méthodes d'exploration approchées basées :
– sur un algorithme glouton et
– sur un algorithme de Branch & Bound limité en temps de calcul.
J'ai aussi proposé une méthode de résolution exacte en complétant la phase de programmation dynamique par une méthode d'exploration exacte de type Branch & Bound. Cette dernière approche nous permet de construire une méthode coopérative originale pour le MKP.

Résultats

Les approches heuristiques développées ont été testées pour différentes instances bien connues de la littérature ou engendrées aléatoirement. Les résultats ont été comparés avec ceux obtenus au moyen d'autres heuristiques de la littérature. J'ai ainsi montré que ces approches heuristiques permettent d'obtenir dans tous les cas une meilleure approximation de la valeur optimale, et j'ai pu dégager une heuristique ayant le meilleur compromis entre temps de calcul et qualité de la borne.

Les tests numériques effectués avec la méthode coopérative de résolution exacte sont assez prometteurs. Ils montrent que, pour certaines instances, les temps de résolution sont plus faibles que ceux obtenus avec une méthode classique de résolution telle que le Branch & Bound. De plus, sur des instances difficiles, la méthode coopérative semble converger plus rapidement vers la valeur de la borne optimale que le Branch & Bound classique.

Mon travail a aussi porté sur la question de la difficulté des instances, notamment dans le cas où

celles-ci sont engendrées aléatoirement. J'ai présenté deux nouvelles techniques pour engendrer aléatoirement des instances difficiles :
– instances dérivées de problèmes combinatoires en contrainte égalité : la difficulté de résolution du problème en contrainte inégalité résultant s'approche de celui en contrainte égalité qui est un problème plus difficile.
– instances engendrées à partir de l'analyse de la transformée en Z : la transformée en Z de (MKP) nous permet de faire des analogies avec le problème du sac à dos à une seule contrainte, pour lequel il existe des résultats bien connus pour engendrer des instances difficiles.

Les tests numériques montrent que ces techniques permettent d'obtenir des instances plus difficiles à la résolution comparées à des techniques classiques de construction de problèmes aléatoires. Ces deux générateurs d'instances difficiles permettent de comprendre un peu mieux la nature de la difficuté de résolution des problèmes (MKP) et d'évaluer l'efficacité des algorithmes de résolution approchées et exactes développés.

Ces différentes approches ainsi que les résultats obtenus ont été publiés dans, notamment, [4], [5], [8], [9] et [10].

3.3. Le problème du partage équitable

Le problème du partage équitable, ou Knapsack Sharing Problem (KSP), est un problème d'optimisation combinatoire NP-complet. Il se définit comme suit :

$$(KSP)\begin{cases} \max \min\limits_{i\in M}\left\{\sum\limits_{j\in N_i} p_{ij}.x_{ij}\right\} = z(KSP) \\ s.c. \sum\limits_{i\in M}\sum\limits_{j\in N_i} w_{ij}.x_{ij} \leq C, \\ \quad x_{ij} \in \{0,1\} \; pour \; i \in M \; et \; j \in N_i. \end{cases} \tag{3.1}$$

avec w_{ij} , p_{ij} pour $i \in \mathcal{M}$ et $j \in \mathcal{N}_i$, et C sont des entiers postifs.

On souhaite déterminer le sous ensemble d'objets à inclure dans le sac à dos afin de maximiser la plus petite valeur des profits. Ce problème intervient lorsque des ressources doivent être partagées de façon équitable entre différentes unités (Par exemple, le partage de munitions entre différentes unités militaires).

Travaux

Le KSP peut être décomposé en sous-problème du sac à dos comme suit :

$$(KP_i(C_i))\begin{cases} \max \sum\limits_{j\in\mathcal{N}_i} p_{ij}.x_{ij} = z(KP_i(C_i)) \\ s.c. \sum\limits_{j\in\mathcal{N}_i} w_{ij}.x_{ij} \leq C_i, \\ \quad x_{ij} \in \{0,1\} \quad j \in \mathcal{N}_i. \end{cases} \tag{3.2}$$

L'objectif est alors de trouver $(C_1^*, C_2^*, ..., C_m^*)$ tel que

$$\sum_{i \in \mathcal{M}} C_i^* \leq C \tag{3.3}$$

$$et \ \min_{i \in \mathcal{M}} \{z(KP_i(C_i^*))\} = z(KSP).$$

Afin de résoudre ces sous-problèmes, dont les capacités sont inconnues, j'ai proposé un algorithme de programmation dynamique, DKSP, utilisant le principe de dominance et qui au cours de la résolution essaie d'obtenir une bonne approximation des $(C_1^*, C_2^*, ..., C_m^*)$. Pour chaque problèmes $(KP_i(C_i))$ une liste d'états est générée qui est mise à jour à chaque étape de l'algorithme.

Les $(C_1, C_2, ..., C_m)$ sont initialisées par des bornes supérieures obtenues en résolvant des problèmes de sac à dos en continu. A chaque étape de la résolution par programmation dynamique, une borne inférieure du KSP est calculé en combinant les états de l'ensemble des listes ainsi qu'une évaluation par borne supérieure des $(C_1^*, C_2^*, ..., C_m^*)$. L'agorithme s'arrête lorsque l'équation 3.3 est sastifaite ou lorsque l'ensemble des étapes de la programmation dynamique ait été considéré.

Résultats

Hifi et al. ont présenté une méthode de résolution pour le KSP qui diffère de DKSP notamment par l'utilisation de la programmation dynamique dense. Mon approche, qui est elle basée sur la programmation dynamique creuse, vient compléter cette méthode existante car elle s'avère plus efficace sur certaines instances. En effet, la comparaison des temps de calcul montre que DKSP permet de résoudre plus rapidement les instances non-corrélées et d'une façon générale les intances présentant relativement peu ou beaucoup de classes (nombre des sous problèmes).

De plus, l'avantage majeure d'utiliser la programmation dynamique creuse réside dans l'occupation mémoire réduite comparée à la programmation dynamique dense. DKSP permet ainsi de résoudre des instances de tailles beaucoup plus grandes sans saturer la mémoire. Cette faible occupation mémoire, ainsi que les temps de calcul raisonnable permettent donc de valider cette approche. La méthode DKSP est présentée plus en détail dans [3].

3.4. Le projet SmartSurface

Le projet smart surface a pour objectif, la conception, le développement et le contrôle d'un système microrobotique distribué pour le convoyage, le positionnement et le tri de micropièces à l'échelle mésoscopique (μm au mm).

Le projet smart surface est financé par l'Agence Nationale de la Recherche et fédère cinq laboratoires de recherche français (femto-st, InESS, LAAS, LIFC, LIMMS).

Cette surface sera fondé sur une matrice de micromodules intelligents (plusieurs centaines). Chaque micromodule sera composé d'un micro-actionneur, d'un microcapteur et d'une unité de traitement. La coopération de ces nombreux micro-modules permettra de reconnaître les pièces et de commander les micro-actionneurs pour déplacer et positionner de façon précise ces pièces sur la smart surface.

Travaux

Mon travail sur ce projet est l'étude des communications distribuées au sein de la smartsurface afin d'assurer le traitement distribué des informations capteurs pour la reconnaissance des pièces. Comme on peut le voir sur la figure 3.4 la smart surface est constituée d'un réseau de cellules comportant chacun un capteur et une unité de traitement et chaque cellule ne peut communiquer qu'avec ces voisines.

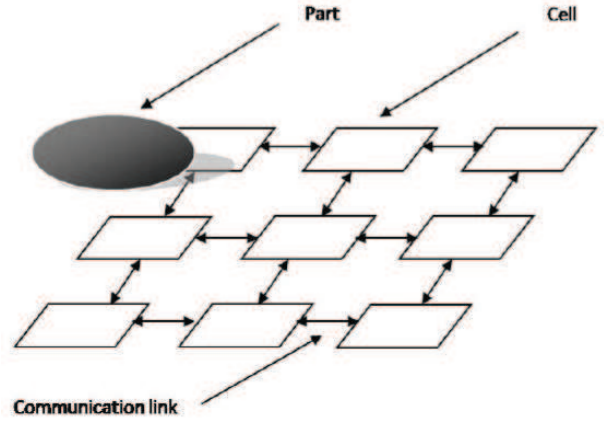


FIGURE 3.1 – Réseau de communication de la smart surface

Lorsqu'une pièce est posée sur cette surface, afin qu'une cellule puisse faire l'acquisition de cette dernière, elle doit connaitre l'état des capteurs de l'ensemble des cellules, et ceux qu'à travers des communications successives avec ces voisines. La problèmatique est alors de gérer cet ensemble de communication et surtout de savoir quand l'algorithme d'acquisition doit s'arrêter.

Pour celà, je me suis servi d'une modélisation par un problème de point fixe :

$$Trouver\ x^* \in E = \{0,1\}^{n^2}\ tel\ que\ x^*\ soit\ le\ plus\ petit\ vecteur\ satisfaisant\ :$$
$$x^* = F(x^*),$$

où n est le nombre de cellules et F une fonction de mapping de E dans E. x_i, $i \in E$, représente la vision qu'à la cellule i de la smart surface.

De cette modélisation, deux types d'algorithme ont été proposés :
- un algorithme synchrone : chaque cellule i se synchronise localement avec ces voisines avant de mettre à jour son vecteur x_i. L'acquisition distribuée se fait alors via des approximations succéssives :
$$x^{k+1} = F(x^k),\ k = 0, 1, ...$$
La distance de Manhattan permet de déterminer le nombre d'itération nécessaire pour trouver le point fixe x^* et donne ainsi un critère d'arrêt fiable.
- un algorithme asynchrone : chaque cellule i va à son propre rythme. Initialement, une cellule est active, le père, et va activer ces voisines , fils, par des envois d'informations qui elles même vont activer les leurs. Une cellule peut se désactiver, si $x_i = F(x_i)$ (aucune nouvelle information a été obtenue entre deux itérés) et que ses fils sont inactifs, et se réactiver si une nouvelle information est reçue. Le point fixe x^* est alors obtenu lorsque le père est désactivé.

Je me suis aussi intéressé à la partie reconnaissance des pièces en me basant sur les travaux de Boutoustous et al. du LIFC. Cette dernière a proposé une méthode de reconnaissance basée sur la comparaisons de critères. Cette approche devait reconnaitre à 100% la pièce (tout ou rien)

et présentait ses limites dés que des rotations étaient appliquées au pièces sur la smart surface. J'ai donc proposé une approche par calcul d'écart par rapport aux critères de référence, ce qui a permis d'améliorer la reconnaissance des pièces.

Résultats

Comme aucun prototype existe à l'heure actuelle, j'ai développé un logiciel de simulation multi-threadé sous JAVA. Cet outil reproduit le réseau de communication de la smart surface et permet donc de vérifier qu'aucun blocage (≪ deadlock ≫) ne se produit. Il permet de :
– créer des groupes de pièces pour la reconnaissance,
– positionner une pièce (translation et rotation),
– tester les algorithmes de communication (avec affichage du graphe d'activité),
– tester les méthodes de reconnaissance,
– simuler des fautes (cellules hors services).
Ce logiciel m'a ainsi permis de valider les algorithmes de communication synchrone et asynchrone pour la communication. Afin de parer aux fautes, un système de ≪ ping-pong ≫a été intégré, permettant ainsi à la smart surface de fonctionner même en mode dégradé.

De plus, j'ai testé les différentes méthodes de reconnaissance en faisant des tirages aléatoires de pièces (position et angle aléatoires également). Ceci a permis notamment de sélectionner les critères les plus pertinants et de montrer que l'approche par calcul d'écart était plus robuste et plus efficace qu'une approche par reconnaissance exacte.

Un effort particulier a été fourni afin de rendre accessible cet environnement de simulation pour simuler et tester d'autres algorithmes. Ce projet ANR, qui doit se terminer cette année, sera peut-être renouvellé par un autre projet ANR ≪ Smart Surface 2 ≫. Ces travaux ont été soumis dans [6] et pour plus d'informations sur ce projet un site internet a été créé :

http ://www.smartsurface.cnrs.fr

3.5. Programmation dynamique dense sur GPU

Depuis quelques années, les fondeurs de cartes graphiques développent des outils permettant d'utiliser leurs réalisations à des fins de calcul généraliste ; on parle alors de ¨General Purpose Graphic Processing Unit¨ ou GPGPU. Utiliser cette puissance de calcul pour résoudre des problèmes d'optimisation combinatoire représente un vrai challenge du fait des structures irrégulières résultant généralement des méthodes de résolution (comme par exemple le Branch & Bound).

Dans le cadre de ce travail, je me suis intéressé à la résolution du problème du sac à dos sur GPU. Le problème du sac à dos, est problème NP-complet et est un des problèmes d'optimisation discrète les plus étudiés car il apparait comme sous-problème de nombreux problèmes plus complexes (cf. par exemple le problème du partage équitable).

Travaux

Le problème du sac à dos, ou knapsack problem (KP), est défini comme suit :

$$(KP) \begin{cases} \max \sum_{j \in N} p_j.x_j = z(KP), \\ \text{s.c.} \sum_{j \in N} w_j.x_j \leq c, \\ x_j \in \{0,1\}, j \in N, \end{cases}$$

p_j, $w_{i,j}$ et c_i, $(i,j) \in N \times M$, étant des entiers positifs.

La programmation dynamique dense est une méthode efficace pour la résolution de problèmes difficiles. Elle est, en effet, insensible au fait que les données du problème soient corrélées ou non. C'est une méthode récursive de complexité temporelle $\mathcal{O}(n.c)$ où à chaque étape $k \in \{1, ..., n\}$ on construit un tableau T^k de taille c tel que :

$$Pour\ j \in \{1, ..., c\} : T_j^k = \begin{cases} max\{T_j^{(k-1)}, T_{j-w_k}^{(k-1)} + p_k\},\ si\ j - w_k \geq 0, \\ T_j^{(k-1)},\ sinon. \end{cases}$$
$$avec\ T^0 = \{0, 0, ..., 0\}.$$

Ce type d'algorithme est adapté à la programmation sur GPU NVIDIA du fait de la connexité des adresses des données manipulées. J'ai donc associé à un thread le remplissage d'une case du tableau T^k. Au total, c threads seront excécutés. Sous CUDA, les threads sont regroupés dans des blocs qui seront distribués sur les multiprocesseurs (1 multiprocesseur ne peut prendre en charge qu'un bloc à la fois). Les c threads ont donc été classés par groupe de 512, ce qui correspond à la taille maximale d'un bloc.

Afin de reconstruire le vecteur solution associé à la borne optimale fournie par l'algorithme, on doit garder en mémoire l'ensemble des décisions prises au cours de la résolution, soit un tableau de taille $n.c$. Pour des problèmes de tailles importantes, l'occupation mémoire devient vite problématique et constitue le principal défaut de cette approche. De plus dans le cadre de la programmation sur GPU, les données de ce tableau doivent être communiquées à la CPU, ce qui résulte d'une perte de temps considérable. Afin de minimiser la quantité de données transmises, j'ai mis au point une technique de compression qui s'effectue directement sur le GPU, et c'est ce tableau compressé qui est alors transmis à la CPU.

Résultats

L'algorithme parallèle développé a été testé sur une carte GTX260 (24 Multiprocesseurs soit 192 coeurs, 1,4GHz) et les temps de calculs obtenus ont été comparés à ceux de l'algorithme séquentiel sur CPU (Intel Xeon 3,0GHz). Les instances considérés sont des instances aléatoires à données corrélées avec $n \in [10000, 100000]$.

Les temps de calculs ont montré une accélération maximale de 26 qui apparait avec les instances de grandes tailles ($n \geq 50000$) et qui reste stable. L'accélération moyenne observée sur l'ensemble des instances testé est de 22. Ces résultats sont d'autant plus encourageants que la fréquence d'horloge du CPU est deux fois plus grande que celle du GPU. Il est à noter que l'instance à 100000 variables n'a pas pu être résolues en séquentiel dans le temps imparti (2 heures).

Concernant la compression de données, la méthode que j'ai développé a montré que, dans le pire cas, la taille du tableau compressé représente $0,3\%$ de l'original. Le taux de compression décroit lorsque la taille des instances augmente, ce qui la rend donc efficace et robuste. Ceci permet

donc de valider cette approche et m'a permis de résoudre des instances de grandes tailles sans saturer la mémoire du CPU (résolution séquentielle) et du GPU (résolution parallèle).

4. Publications

1. V. Boyer, "Contribution à la programmation en nombre entier", Mémoire de thèse, Rapport LAAS n° 07804, 2008.

4.1. Revues internationnales avec comité de lecture

2. V. Boyer, D. El Baz & M. Elkihel, "Solving knapsack problems on GPU", soumis à Computers & Operations Research, 2010.
3. V. Boyer, D. El Baz & M. Elkihel, "A dynamic programming method with lists for the knapsack sharing problem", Computers & Industrial Engineering, 2010, à paraître.
4. V. Boyer, D. El Baz & M. Elkihel, "A heuristic for the 0-1 multidimensional knapsack problem", European Journal of Operational Research, Vol. 199, p. 658-664, 2009.
5. V. Boyer, D. El Baz & M. Elkihel, "Solution of multidimensionnal knapsack problems via cooperation of dynamic programming and branch and bound", European Journal of Industrial Engineering, 2009, à paraître.

4.2. Colloques internationales avec actes et comité de lecture

6. D. El-Baz, V. Boyer, J. Bourgeois, E. Dedu, K. Boutoustous, "Distributed discrete state acquisition and concurrent pattern recognition in a MEMS-based smart surface", soumis à DMEMS 2010, 2010.
7. V. Boyer, D. El Baz & M. Elkihel, "A dynamic programming method with dominance technique for the knapsack sharing problem", The 39th International Conference on Computers & Industrial Engineering, 2009.
8. V. Boyer, D. El Baz & M. Elkihel, "An exact cooperative method for solving the 0-1 multidimensional knapsack problem", MOSIM'08, Vol. 2, p. 927-934, 2008.

4.3. Colloques nationales avec actes et comité de lecture

9. V. Boyer, D. El Baz, M. Elkihel & J.B. Lasserre, "Générateur d'instances difficiles pour le sac à dos multidimensionnel en variable 0-1", Presses universitaire de l'université Blaise Pascal, article long selectionné pour paraître dans les actes de la conférence ROADEF, p. 33-43, 2008.
10. V. Boyer, D. El Baz & M. Elkihel, "Efficient heuristic for the 0-1 multidimensional knapsack problem", Presses Universitaires de Valencienne, article long selectionné pour paraître dans les actes de la conférence ROADEF, p. 95-106, 2006.

4.4. Conférences

11. V. Boyer, D. El Baz & M. Elkihel, "Programmation dynamique dense sur GPU", ROADEF'10, Toulouse (France), 2010.
12. V. Boyer, D. El Baz & M. Elkihel, "An exact method for the 0-1 multidimensional knapsack problem", ROADEF'07, Grenoble (France), 2007.
13. V. Boyer, D. El Baz & M. Elkihel, "A new heuristic for the multidimensional knapsack problem", 2nd International Workshop on Combinatorial Scientific Computing (CSC'05), Toulouse (France), 2005.

5. Activités d'enseignement

5.1. Mathématique

Niveau : Licence 2ème année
Responsable : Jean-Baptiste Hiriart-Urruty
Années : 2007-2009
Charge : TD
Nombre d'étudiants : 40 étudiants
Nombre d'heures : 24h (1 groupes de TD)

Description :

Analyse, Géométrie :

- Géométrie en 2D (rappels et compléments)
- Géométrie en 3D : Modes de repérage d'un point. Droites et plans. Distances. Angles. Sphères, cylindres, cônes (premiers cas simples). Annexe (rappels) : produits vectoriels, mixtes.
- Fonctions de plusieurs variables : Calcul différentiel et développement du 1er ordre. Calcul différentiel et développement du 2nd ordre. Application : conditions d'optimalité dans le problème de la minimisation d'une fonction de deux variables. Fonctions quadratiques (définition, propriétés de base).
- Introduction à l'optimisation : La méthode de plus profonde descente le long (de l'opposé) du gradient.
- Intégrales curvilignes : Compléments d'Analyse vectorielle. Intégrales curvilignes.
- Intégrales doubles : Techniques de calcul. Relations entre intégrales doubles et intégrales curvilignes : théorème de G. GREEN.
- Intégrales de surface. Théorème de la divergence. Les surfaces en 3D. Intégrales de surface. Théorème de la divergence.
- La formule de transformation de G. STOKES.

Algèbre linéaire :
C'est le point de vue "matriciel" qui est privilégié ici, et non le point de vue "application linéaire".

- Valeurs propres et vecteurs propres d'une matrice. Diagonalisation. Cas particulier important des matrices symétriques réelles ; lien avec les fonctions quadratiques.
- Représentations matricielles des principales transformations linéaires en 2D et 3D (symétries, rotation, ...)

5.2. Informatique (Maple)

Niveau :	Licence 2ème année, Préparation aux Concours Polytechniques
Responsables :	Marcel Mongeau et Youchun Qiu
Années :	2007-2009
Charge :	TD
Nombre d'étudiants :	45 étudiants répartis en 3 groupes
Nombre d'heures :	3*24h (3 groupes de TD)
Description :	TD d'initiation à la programmation et au logiciel Maple qui permet de faire des mathématiques en manipulant des expressions symboliques et numériques. Ce cours s'adresse aux étudiants préparant plus particulièrement le Concours des ENSI DEUG. Analyse : séries numériques, suites et séries de fonctions, intégrales généralisées, séries entières. Algèbre : diagonalisation, espaces préhilbertiens et euclidiens, groupe orthogonal.
Fonctionnement :	Contrôle continu : 2 épreuves de 1h, un examen de 2h. J'ai eu l'opportunité sur ces TD de participer activement à l'élaboration des sujets de TD ainsi que des examens et de prendre en charge une partie des corrections.

5.3. Informatique (langage C)

Niveau :	Licence 2ème année
Années :	2005-2006
Charge :	TD
Nombre d'étudiants :	$\approx$ 20 étudiants
Nombre d'heures :	18h (1 groupes de TD)
Description :	Il s'agit d'apprendre à utiliser un environnement de programmation (éditeur + compilateur) et d'utiliser l'outil pour mettre en oeuvre les algorithmes dont a besoin l'ingénieur. Initiation au langage C pour l'ingénieur : Structure générale d'un programme, principales instructions de contrôle de flux, variables simples et dimensionnées, structures, Création et utilisation de fonctions, fonctions d'entrée-sortie, opérateurs logiques.

5.4. Électrocinétique

Niveau :	Licence 1ère année
Responsable :	Vassant Sewraj
Années :	2004-2007
Charge :	TP
Nombre d'étudiants :	$\approx$ 48 étudiants répartis en 3 groupes
Nombre d'heures :	3*24h. (3 groupes de TD)
Description :	Initiation au montage de circuits éléctriques (lecture d'un schéma, branchements, Multimètre, caractéristique de dipôle, ...). Étude de différents montages électriques (filtres, redressement de fonctions, ...).

5.5. Physique

Niveau :	Licence 1ère année
Responsable :	Vassant Sewraj
Années :	2004-2007
Charge :	TP
Nombre d'étudiants :	$\approx$ 16 étudiants
Nombre d'heures :	12h.
Description :	TP sur l'effet Doppler. Matériel : un banc constitué d'un micro mobile et d'un haut-parleur fixe, et acquisition des points de mesure sur un PC.

6. Annexes

6.1. Extraits de publications

Dans cette section, j'ai choisi de mettre une copie de deux de mes publications qui permettent d'illustrer en partie mes activités de recherche :

– V. Boyer, D. El Baz & M. Elkihel, "A heuristic for the 0-1 multidimensional knapsack problem", EJOR, doi :10.1016/j.ejor.2007.06.068, 2008, à paraître, disponible en ligne.
– V. Boyer, D. El Baz & M. Elkihel, "An exact cooperative method for solving the 0-1 multidimensional knapsack problem", MOSIM'08, Vol. 2, P. 927-934, 2008.

European Journal of Operational Research xxx (2008) xxx–xxx

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Heuristics for the 0–1 multidimensional knapsack problem

V. Boyer, M. Elkihel *, D. El Baz

LAAS-CNRS, Université de Toulouse, 7 Avenue du Colonel Roche, 31077 Toulouse Cedex 4, France

ARTICLE INFO

Article history:
Received 9 September 2006
Accepted 8 June 2007
Available online xxxx

Keywords:
Multidimensional knapsack problem
Dynamic-programming
Branch-and-cut
Surrogate relaxation
Heuristics

ABSTRACT

Two heuristics for the 0–1 multidimensional knapsack problem (MKP) are presented. The first one uses surrogate relaxation, and the relaxed problem is solved via a modified dynamic-programming algorithm. The heuristics provides a feasible solution for (MKP). The second one combines a limited-branch-and-cut-procedure with the previous approach, and tries to improve the bound obtained by exploring some nodes that have been rejected by the modified dynamic-programming algorithm. Computational experiences show that our approaches give better results than the existing heuristics, and thus permit one to obtain a smaller gap between the solution provided and an optimal solution.

1. Introduction

The NP-hard multidimensional knapsack problem (MKP) (see [10,13,20]) arises in several practical problems such as capital budgeting, cargo loading, cutting stock problem and processors allocation in huge distributed systems. It can be defined as

$$(\text{MKP})\begin{cases}\max \sum_{j\in N} p_j \cdot x_j,\\ \text{subject to} \sum_{j\in N} w_{i,j}\cdot x_j \leqslant c_i, \quad \forall i \in M,\\ x_j \in \{0,1\}, \quad \forall j \in N,\end{cases} \tag{1}$$

where

- $N = \{1,2,\ldots,n\}$ and $M = \{1,2,\ldots,m\}$,
- n is the number of items,
- m is the number of constraints,
- $p_j \geqslant 0$ is the profit of the jth item,
- $w_{i,j} \geqslant 0$, for $i \in M$, are the weights of the jth item,
- and $c_i \geqslant 0$, for $i \in M$, are the capacities of the knapsack.

In the sequel, we shall use the following notation: given a problem (P), its optimal value will be denoted by $v(P)$.

To avoid any trivial solution, we assume that

- $\forall j \in N$ and $\forall i \in M$, $w_{i,j} \leqslant c_i$.
- $\forall i \in M$, $\sum_{j=1}^{n} w_{i,j} > c_i$.

A specific case of (MKP) is the classical knapsack problem with $m = 1$. The unique knapsack problem (UKP) has been given considerable attention in the literature though it is not, in fact, as difficult as (MKP), more precisely, it can be solved in a pseudo-polynomial time (see [2,3,6,11,12]). We have then tried to transform the original (MKP) into a (UKP) (see also [15,17]). In this purpose, we have used a relaxation technique, that is to say, surrogate relaxation. The surrogate relaxation of (MKP) can be defined as follows:

$$(S(u))\begin{cases}\max \sum_{j\in N} p_j \cdot x_j,\\ \text{subject to} \sum_{i\in M} u_i \cdot \sum_{j\in N} w_{i,j}\cdot x_j \leqslant \sum_{i\in M} u_i \cdot c_i,\\ x_j \in \{0,1\}, \quad \forall j \in N,\end{cases} \tag{2}$$

where $u^T = (u_1,\ldots,u_m) \geqslant 0$.

Since $(S(u))$ is a relaxation of (MKP), we have $v(S(u)) \geqslant v(\text{MKP})$, and the optimal multiplier vector, u^*, is defined as

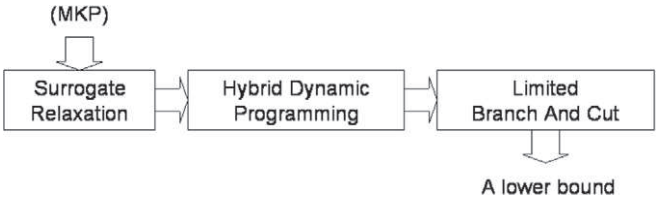
$$v(S(u^*)) = \min_{u\geqslant 0}\{v(S(u))\}. \tag{3}$$

Several heuristics have been proposed in order to find out good surrogate multipliers (see in particular [14,15,17]). In practice, it is not important to obtain the optimal multiplier vector, since in the general case we have no guarantee that $v(S(u^*)) = v(\text{MKP})$. Solving $(S(u))$ will give an upper bound of (MKP). In the sequel, we propose efficient algorithms based on dynamic-programming in order to find a good lower bound of (MKP) by solving $(S(u))$.

The basic algorithmic scheme can be presented as follow:

* Corresponding author.
E-mail addresses: vboyer@laas.fr (V. Boyer), elkihel@laas.fr (M. Elkihel), elbaz@laas.fr (D. El Baz).

0377-2217/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2007.06.068



We have studied first a heuristics based on dynamic-programming and surrogate relaxation of (MKP). We remarked that some of the states eliminated by the heuristics could be further explored in order to improve the bound. Thus, we designed a new heuristics which is based on a limited-branch-and-cut-procedure and that is added to first heuristics. In this last case, the following 2 phases can be considered:

– *Phase 1*: Surrogate relaxation + Hybrid-dynamic-programming.
– *Phase 2*: Limited-branch-and-cut.

The sole Phase 1 and the combination of the two phases represent the two heuristics proposed in this article. The first phase provides a lower bound of (MKP) and a list of states that will be treated in the second phase in order to improve the bound.

Solutions obtained with the above heuristics are compared with results given by heuristics from the literature such as:

– AGNES of Freville and Plateau [7].
– ADP-based heuristics approach of Bertsimas and Demir [8].
– Simple Multistage Algorithm (SMA) of Hanafi, Freville and El Abdellaoui [9].

Note that AGNES combines different greedy methods based on surrogate relaxation and the solutions provided are improved by a neighborhood search (a neighborhood is defined around a solution and is explored to find out a better solution). The ADP-based heuristics uses a diversification method, a taboo algorithm, completed too by a neighborhood search. The Simple Multistage Algorithm is an approximate dynamic-programming approach.

The first phase of our method can be seen as a diversification method which provides a family of feasible solutions for (MKP), and we try to improve the bound by exploring the neighborhood of these solutions with the second phase. This approach allows us to make a real cooperation between the two phases.

In the sequel, each steps of our algorithm is described.

The paper is organized as follows. In Section 2, we present the hybrid-dynamic-programming algorithm (HDP) and in Section 3 a cooperative method: the so-called limited-branch-and-cut method (LBC). Finally, in Section 4, we provide and analyze some computational results obtained with different instances from the literature and randomly generated instances. Our heuristics are also compared with other existing heuristics.

2. The surrogate relaxation

Solving (3) is not easy. As mentioned above, many heuristics exist and provide good approximations of u^*. A reasonable estimation can be calculated by dropping the integer restrictions in x. In other words, let

$$(LS(u))\begin{cases}\max \sum_{j\in N} p_j \cdot x_j,\\ \text{subject to} \sum_{i\in M} u_i \sum_{j\in N} w_{i,j}\cdot x_j \leqslant \sum_{i\in M} u_i \cdot c_i,\\ x_j \in [1], \quad \forall j \in N.\end{cases} \tag{4}$$

The best surrogate constraint is then generated by u^0, where

$$v(LS(u^0)) = \min_{u\geqslant 0} v(LS(u)). \tag{5}$$

In order to calculate the best surrogate constraint we consider the linear programing (LP) corresponding to (MKP)

$$(LP)\begin{cases}\max \sum_{j\in N} p_j \cdot x_j,\\ \text{subject to} \sum_{j\in N} w_{i,j}\cdot x_j \leqslant c_i, \quad \forall i \in M,\\ x_j \in [0,1], \quad \forall j \in N.\end{cases} \tag{6}$$

We denote by $\lambda^0 = (\lambda_1^0, \lambda_2^0, \ldots, \lambda_m^0) \geqslant 0$ the dual optimal variables corresponding to the constraints

$$\sum_{j\in N} w_{i,j}\cdot x_j \leqslant c_i, \quad i \in M. \tag{7}$$

We are now ready to show how to calculate the best surrogate constraint using the definition (5).

Theorem 1 (see [16, p. 132]). *The best surrogate constraint is generated by* $u^0 = \lambda^0$.

Then we have the following order relation (see [15,16, p. 130] and [18]):

$$v(LP) = v(LS(u^0)) \geqslant v(S(u^*)) \geqslant v(\text{MKP}). \tag{8}$$

Table 1 gives the bounds obtained with the surrogate relaxation for a set of instances from the literature.

3. Hybrid-dynamic-programming (HDP)

For simplicity of presentation, we will denote in the sequel $\sum_{i\in M} u_i.w_{i,j}$ by w_j and $\sum_{i\in M} u_i \cdot c_i$ by c. Then we have

$$(S(u))\begin{cases}\max \sum_{j\in N} p_j \cdot x_j,\\ \text{subject to} \sum_{j\in N} w_j \cdot x_j \leqslant c,\\ x_j \in \{0,1\}, \quad \forall j \in N.\end{cases} \tag{9}$$

We apply the dynamic-programming list algorithm to $(S(u^0))$ and we keep only the feasible solutions of (MKP). At each step, we update a list which is defined as follows:

$$\text{For } k \in N, \quad \mathscr{L}_k = \left\{(w,p) | w = \sum_{j=1}^{k} w_j \cdot x_j \leqslant c,\ p = \sum_{j=1}^{k} p_j \cdot x_j\right\} \tag{10}$$

The use of the concept of dominated states can permit one to reduce drastically the size of lists $\mathscr{L}_k$ since dominated states, according to Bellman's optimality principle, can be eliminated from the list:

Dominated state: Let (w,p) be a couple of weight and profit, i.e. a state of the problem. If $\exists(w',p')$ such that $w' \leqslant w$ and $p' \geqslant p$, then (w,p) is dominated by (w',p').

Note that dominated states must be saved in a secondary list denoted by $\mathscr{L}_{\text{sec}}$ since they can nevertheless give rise to an optimal solution of (MKP).

3.1. Dynamic-programming algorithm (DP)

List of states $\mathscr{L}_{k+1}$ are generated recursively by the dynamic-programming list algorithm (see [5] for more details and some examples). At stage $k+1$ the list $\mathscr{L}_{k+1}$ is constructed as follows:

The set of new states generated at stage $k+1$ is given by

$$\begin{aligned}\mathscr{L}'_{k+1} &= \mathscr{L}_k \oplus (w_{k+1}, p_{k+1})\\ &= \{(w+w_{k+1}, p+p_{k+1}) | (w,p) \in \mathscr{L}_k \quad \text{and} \quad w+w_{k+1} \leqslant c\},\end{aligned}$$

Table 1
Surrogate relaxation on some instances from the literature

Name of the instance	$n \times m$	v(MKP)	$v(LP)$	$v(LS(u^0))$	$v(S(u^0))$
Petersen 1	6×10	3800	4134.07	4134.07	3800
Petersen 2	10×10	87,061	92,977.70	92,977.70	91,779
Petersen 3	15×10	4015	4127.89	4127.89	4105
Petersen 4	20×10	6120	6155.33	6155.33	6120
Petersen 5	28×10	12,400	12,462.10	12462.10	12,440
Petersen 6	39×5	10,618	10,672.35	10,672.35	10,662
Petersen 7	50×5	16,537	16,612.82	16,612.82	16,599
Hansen and Plateau 1	28×4	3418	3472.35	3472.35	3462
Hansen and Plateau 2	35×4	3186	3261.82	3261.82	3248
Weingartner 1	28×2	141,278	14,2019.00	142,019.00	141,548
Weingartner 2	28×2	130,083	13,1637.47	131,637.47	130,883
Weingartner 3	28×2	95,677	99,647.08	99,647.08	97,906
Weingartner 4	28×2	119,337	122,505.25	122,505.25	121,087
Weingartner 5	28×2	98,796	100,433.16	100,433.16	98,796
Weingartner 6	28×2	130,623	131,335.00	131,335.00	130,733
Weingartner 7	105×2	1,095,445	1,095,721.25	1,095,721.25	1,095,591
Weingartner 8	105×2	624,319	628,773.69	628,773.69	627,976

and we have: $\mathscr{L}_{k+1} := \mathscr{L}_k \cup \mathscr{L}'_{k+1} - \mathscr{D}_{k+1}$, where $\mathscr{D}_{k+1}$, the set of dominated pairs at stage $k+1$, is defined as follow:

$$\mathscr{D}_{k+1} = \{(w,p) | (w,p) \in \mathscr{L}_k \cup \mathscr{L}'_{k+1} \quad \text{and} \quad \exists (w',p') \in \mathscr{L}_k \cup \mathscr{L}'_{k+1} \text{ with } w' \leqslant w,\ p \leqslant p' \quad \text{and} \quad (w',p') \neq (w,p)\}.$$

Initially, we have $\mathscr{L}_0 = \{(0,0)\}$.

Let (w,p) be a state generated at stage k, we define the subproblem associated with (w,p) by

$$(S(u))_{(w,p)} \begin{cases} \max \sum_{j=k+1}^{n} p_j \cdot x_j + p, \\ \text{subject to } \sum_{j=k+1}^{n} w_j \cdot x_j \leqslant c - w, \\ x_j \in \{0,1\}, \quad j \in \{k+1, \ldots, n\}. \end{cases} \tag{11}$$

An upper bound, $\bar{v}_{(w,p)}$, of the above problem, is obtained by solving the linear relaxation of $(S(u))_{(w,p)}$, i.e. $(LS(u))_{(w,p)}$, with the Martello and Toth algorithm (see [4]) and a lower bound, $\underline{v}_{(w,p)}$, is obtained with a greedy algorithm on $(S(u))_{(w,p)}$.

In a list, all the states are sorted by their decreasing upper bound. As mentioned above, our algorithm consists in applying dynamic-programming (DP) to solve $S(u^0)$. At each stage of DP, the following points are checked when a new state (w,p) is generated:

– Is the state feasible for (MKP) (this will permit one to eliminate the unfeasible solutions)? Then, we try to improve the lower bound of (MKP), $\underline{v}$(MKP), with the value of p.
– Is the state dominated? In this case the state is saved in the secondary list $\mathscr{L}_{\text{sec}}$.
– Is the upper bound associated with the state (w,p) smaller than the current lower bound of $S(u^0)$? Then the state is saved too in the secondary list $\mathscr{L}_{\text{sec}}$.

For each state (w,p) which has not been eliminated or saved in the secondary list after these tests, we try to improve the lower bound of $(S(u^0))$, i.e. $\underline{v}(S(u^0))$, by computing a lower bound of the state with a greedy algorithm.

DP algorithm is described below:

Dynamic-programming algorithm (DP):
Initialisation:
$\mathscr{L}_0 = \{(0,0)\}$, $\mathscr{L}_{\text{sec}} = \emptyset$
$\underline{v}(S(u^0)) = \underline{v}$(MKP) (where $\underline{v}$(MKP) is a lower bound of (MKP) given by a greedy algorithm).
Computing the lists:
For j := 1 to n
$\mathscr{L}'_j := \mathscr{L}_{j-1} \oplus (w_j, p_j)$
Remove all states $(w,p) \in \mathscr{L}'_j$ that are unfeasible for (MKP)
$\mathscr{L}_j := \text{MergeLists}(\mathscr{L}_{j-1}, \mathscr{L}'_j)$
For each state $(w,p) \in \mathscr{L}_j$ Compute $\bar{v}_{(w,p)}$ and $\underline{v}_{(w,p)}$
Updating the bounds:
$p_{max} := max\{p | (w,p) \in \mathscr{L}_j\}$ and $v_{max} := max\{\underline{v}_{(w,p)} | (w,p) \in \mathscr{L}_j\}$
$\underline{v}(\text{MKP}) := max\{\underline{v}(\text{MKP}), p_{max}\}$
$\underline{v}(S(u^0)) := max\{\underline{v}(S(u^0)), v_{max}\}$
Updating $\mathscr{L}_{\text{sec}}$:
$\mathscr{D}_j = \{(w,p) | (w,p)$ is dominated or $\bar{v}_{(w,p)} \leqslant \underline{v}(S(u^0))\}$
$\mathscr{L}_{\text{sec}} := \mathscr{L}_{\text{sec}} \cup \mathscr{D}_j$ and $\mathscr{L}_j := \mathscr{L}_j - \mathscr{D}_j$
End for.

In the end of the algorithm, we obtain a lower bound of (MKP). In order to improve the lower bound and the efficiency of DP algorithm, we add to the algorithm a reducing-variable procedure, which is defined as follow:

Reducing-variables rule 1: Let $\underline{v}$ be a lower bound of (MKP) and v_j^0, v_j^1, respectively, be the upper bounds of (MKP) with $x_j = 0$, $x_j = 1$, respectively. If $\underline{v} > v_j^k$ with $k = 0$ or 1, then we can definitively fix $x_j = 1 - k$.

These upper bounds are obtained with the Martello and Toth algorithm on $(S(u^0))$. We use this reducing-variables rule whenever we improve $\underline{v}$(MKP) during the dynamic-programming phase. When a variable is fixed, we have to update all the states of the active list and to eliminate all the states which do not match the fixed variables or are unfeasible. The results of DP are presented in Table 2.

3.2. Improvement of the lower bound (ILB)

We present now a procedure that allows us to improve significantly the lower bound given by DP algorithm. More precisely, we try to obtain better lower bounds for the states saved in the secondary list. Before calculating these bounds, we eliminate all the states that have become unfeasible or which are uncompatible with the variables that have been yet reduced or that have an upper bound smaller than the current lower bound of (MKP), i.e. $\underline{v}$(MKP).

For a state (w,p), let J be the index set of free variables and $I = N - J$ the set of fixed variables. If the states have been generated at the kth stage of DP Algorithm, $J = \{k+1, \ldots, n\}$, $w = \sum_k^{j=1} w_j \cdot x_j$ and $p = \sum_{j=1}^{k} p_j \cdot x_j$, where x_j, $j \in I$ denote here the values of the component of vector x which have been already compute during the k first step of the dynamic-programming list method. Then we define the new subproblem:

Please cite this article in press as: V. Boyer et al., Heuristics for the 0–1 multidimensional knapsack problem, European Journal of Operational Research (2008), doi:10.1016/j.ejor.2007.06.068

$$(\mathrm{MKP})_{(w,p)} \begin{cases} \max \sum_{j \in J} p_j \cdot x_j + p, \\ \text{subject to } \sum_{j \in J} w_{i,j} \cdot x_j \leqslant \bar{c}_i, \quad \forall i \in M, \\ x_j \in \{0,1\}, \quad \forall j \in J, \end{cases} \tag{12}$$

where $\bar{c}_i = c_i - \sum_{j \in I} w_{i,j} \cdot x_j$, $\forall i \in M$.

Two methods are used in order to evaluate the lower bound of the above problem:

– A greedy algorithm.
– An enumerative method when the number $n' = n - k$ of variables of the subproblem is sufficiently small (given by the parameter α: $n' \leqslant \alpha$).

When all the states have been treated the process stops. The detail of the algorithm is given in what follows:

Procedure ILB:

Assign to $\underline{v}(\mathrm{MKP})$ the value of the lower bound returned by DP algorithm.
For each state $(w,p) \in \mathscr{L}_{\mathrm{sec}}$
Compute $\underline{v}_{(w,p)}$ a lower bound of $(\mathrm{MKP})_{(w,p)}$
Endfor.

$v_{\max} := \max\{\underline{v}_{(w,p)} | (w,p) \in \mathscr{L}_{\mathrm{sec}}\}$.
$\underline{v}(\mathrm{MKP}) := \max\{\underline{v}(\mathrm{MKP}), v_{\max}\}$.

The improvement given by ILB (with low cost in term of processing time) can be clearly seen from the comparison of Tables 2 and 3. Empirically, the value $\alpha = 10$ has given the better results.

Table 2
Lower bounds given by DP

Name of the instance	$n \times m$	v(MKP)	$\underline{v}$(MKP)	Gap (%)
Petersen 1	6×10	3800	3700	2.63
Petersen 2	10×10	87,061	83,369	4.24
Petersen 3	15×10	4015	3245	19.18
Petersen 4	20×10	6120	6010	1.80
Petersen 5	28×10	12,400	11,930	3.79
Petersen 6	39×5	10,618	10,313	2.87
Petersen 7	50×5	16,537	16,449	0.53
Hansen and Plateau 1	28×4	3418	3347	2.08
Hansen and Plateau 2	35×4	3186	3098	2.76
Weingartner 1	28×2	141,278	140,477	0.57
Weingartner 2	28×2	130,083	130,723	0.12
Weingartner 3	28×2	95,677	95,627	0.05
Weingartner 4	28×2	119,337	104,799	12.18
Weingartner 5	28×2	98,796	98,796	0.00
Weingartner 6	28×2	130,623	130,233	0.30
Weingartner 7	105×2	1,095,445	1,094,757	0.06
Weingartner 8	105×2	624,319	619,101	0.84

Table 3
Lower bounds with ILB, $\alpha = 10$

Name of the instance	$n \times m$	v(MKP)	$\underline{v}$(MKP)	Gap (%)
Petersen 1	6×10	3800	3800	0.00
Petersen 2	10×10	87,061	87,061	0.00
Petersen 3	15×10	4015	4015	0.00
Petersen 4	20×10	6120	6120	0.00
Petersen 5	28×10	12,400	12,400	0.00
Petersen 6	39×5	10,618	10,618	0.00
Petersen 7	50×5	16,537	16,508	0.18
Hansen and Plateau 1	28×4	3418	3418	0.00
Hansen and Plateau 2	35×4	3186	3148	1.19
Weingartner 1	28×2	141,278	141,278	0.00
Weingartner 2	28×2	130,083	130,083	0.00
Weingartner 3	28×2	95,677	95,677	0.00
Weingartner 4	28×2	119,337	119,337	0.00
Weingartner 5	28×2	98,796	98,796	0.00
Weingartner 6	28×2	130,623	130,623	0.00
Weingartner 7	105×2	1,095,445	1,095,445	0.00
Weingartner 8	105×2	624,319	624,319	0.00

4. Limited-branch-and-cut (LBC)

In this section we present the last part of our algorithm, it permits one to improve the lower bound provided by the (ILB) procedure. As mentioned above, the states in the secondary list $\mathscr{L}_{\mathrm{sec}}$ can give rise to better results for (MKP). We propose an algorithm based on a branch-and-cut method in order to explore a neighborhood of the states in $\mathscr{L}_{\mathrm{sec}}$.

4.1. Classic branch-and-cut

Let (w,p) be the first state of $\mathscr{L}_{\mathrm{sec}}$ (the states are sorted according to their decreasing upper bounds).

An upper bound of $(\mathrm{MKP})_{(w,p)}$, $\bar{v}_{(w,p)}$, is obtained by solving its linear relaxation, using a simplex algorithm, and a lower bound, $\underline{v}_{(w,p)}$, is obtained with a greedy algorithm on $(\mathrm{MKP})_{(w,p)}$.

We propose the following branching strategy:

Branching rule: Let (w,p) be a state of the problem (MKP), J the index of the free variables and $\widetilde{X_J} = \{\widetilde{x_j} | j \in J\}$ an optimal solution of the linear relaxation of $(\mathrm{MKP})_{(w,p)}$. Then the branching variable x_k, $k \in J$, is such that $k = \arg\min_{j \in J}\{|\widetilde{x_j} - 0.5|\}$.

Whenever we evaluate an upper bound, we use the following reducing-variable method:

Reducing-variables rule 2 (see [19]): Let $\underline{v}$ be a lower bound of (MKP). Let $\widetilde{v}$ be the optimal bound and $\widetilde{X} = \{\widetilde{x_j} | j \in N\}$ an optimal solution of the linear relaxation of (MKP). Then we denote by $\widetilde{P} = \{\widetilde{p_j} | j \in N\}$, the reduced profits. For $j \in N$, if $\widetilde{x}_j = 0$, $\widetilde{x}_j = 1$, respectively, and $\widetilde{v} - |\widetilde{p}_j| \leqslant \underline{v}$, then there exists an optimal solution for (MKP) with $x_j = 0$, $x_j = 1$, respectively.

This last rule permits one to reduce significantly the processing time by reducing the number of states to explore.

4.2. Limited-branch-and-cut (LBC)

We propose a method, based on the branch-and-cut technique described above, to explore quickly the states saved in the secondary list. Indeed, at each step of the algorithm, we enforce the value of the variables in order to limit the processing time. We use the following heuristics to fix variables:

Reducing-variables rule 3: Let $\tilde{v}$ be the optimal bound and $\widetilde{X} = \{\widetilde{x_j} | j \in N\}$ an optimal solution of the linear relaxation of (MKP). For $j \in N$, if $\tilde{x}_j = 0$, $\tilde{x}_j = 1$, respectively, then x_j is fixed to 0, 1, respectively.

In order to limit the exploration, we consider only 50% of the secondary list, that is to say, we decide to consider only half the states in $\mathscr{L}_{\mathrm{sec}}$ with the best upper bounds.

Procedure LBC:

Assign to $\underline{v}(\mathrm{MKP})$ the value of the lower bound returned by ILB algorithm.
While $\mathscr{L}_{\mathrm{sec}} \neq \emptyset$
  Let (w,p) be the first state in $\mathscr{L}_{\mathrm{sec}}$
  $\mathscr{L}_{\mathrm{sec}} := \mathscr{L}_{\mathrm{sec}} - (w,p)$
  Compute $\bar{v}_{(w,p)}$ an upper bound of $(\mathrm{MKP})_{(w,p)}$
  If $\bar{v}_{(w,p)} > \underline{v}(\mathrm{MKP})$
    Fix variables according to reducing-variables rule 3 and update p
    Compute $\underline{v}_{(w,p)}$ a lower bound of $(\mathrm{MKP})_{(w,p)}$
    If $\underline{v}_{(w,p)} > \underline{v}(\mathrm{MKP})$ then $\underline{v}(\mathrm{MKP}) := \underline{v}_{(w,p)}$ Endif
    Chose the branching variable and branch on it
    Insert the two resulting states in $\mathscr{L}_{\mathrm{sec}}$ if they are feasible
  Endif
Endwhile.

Please cite this article in press as: V. Boyer et al., Heuristics for the 0–1 multidimensional knapsack problem, European Journal of Operational Research (2008), doi:10.1016/j.ejor.2007.06.068

Table 4
Small instances from the literature

Name of the instance	Number of instances	Average gap to optimality (%)			
		HDP	SMA	ADP	AGNES
Petersen	7	0.02	8.24	1.62	1.05
Hansen and Plateau	2	0.45	8.34	7.28	1.44
Weingartner	8	0.01	4.67	4.05	3.37
Freville and Plateau	6	0.44	12.85	6.86	1.91
Fleisher	1	0.00	12.16	0.00	3.60
Sent	2	0.00	1.65	0.35	0.20

5. Computational experiences

Our heuristics were programmed in C and compiled with GNU's GCC. Computational experiences were carried out using an Intel Pentium M Processor 725. We compare our heuristics to the following heuristics of the literature:

– AGNES of Freville and Plateau [7].
– ADP-based heuristic approach of Bertsimas and Demir [8].
– Simple Multistage Algorithm (SMA) of Hanafi, Freville and El Abdellaoui [9].

Our tests were made on the following instances:

– small instances from the literature of Petersen, Weingartner, Hansen and Plateau, Freville and Plateau, Fleisher and Sent ([1]).
– Large instances from the literature of Chu and Beasley ([1]).
– Randomly generated instances:

$$(\mathrm{MKP})\begin{cases}\max \sum_{j\in N} p_j \cdot x_j, \\ \text{subject to} \sum_{j\in N} w_{i,j}\cdot x_j \leqslant \epsilon . \sum_{j\in N} w_{i,j}, \quad \forall i \in M, \\ x_j \in \{0,1\}, \quad \forall j \in N,\end{cases} \tag{13}$$

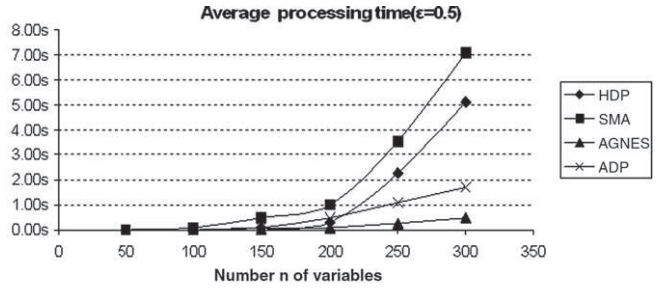
where $\mathrm{N} = \{1,2,\ldots,n\}$, $\mathrm{M} = \{1,2,\ldots,]\frac{n}{2}[\}$, $p_j \in [0,1000]$ for all $j \in N$, $w_{i,j} \in [0,1000]$, for all $i \in M$, $j \in N$ and $\epsilon \in]0,1[$.

Note that, for the instances randomly generated and from Chu and Beasley, the bounds provided by the heuristics are compared with the optimal solution of the corresponding linear relaxation (see (6)).

5.1. HDP heuristics

5.1.1. Instances from the literature

From Tables 4 and 5, we note that the lower bound given by HDP is better than the others. It is difficult to compare processing time with the small instances since it takes less than 1 second to solve all at once. In Table 5, we could remark that for the seven first sets we have a competiting processing time compared with other heuristics.



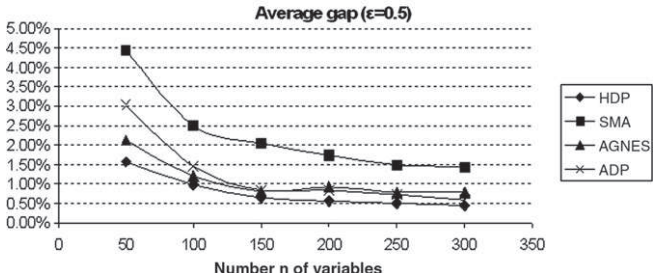
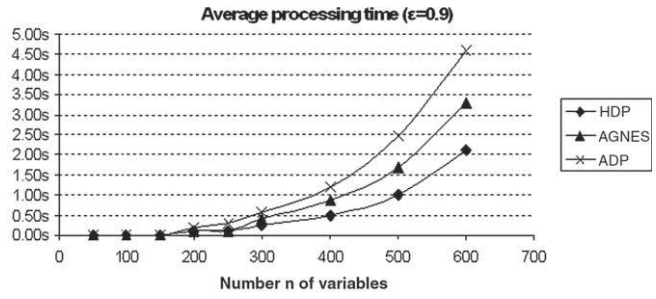


Fig. 1. Average performances with $\epsilon = 0.5$.



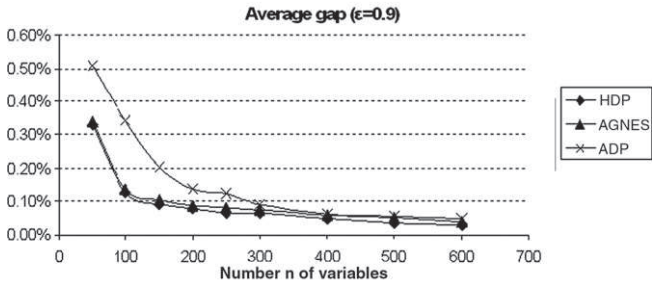


Fig. 2. Average performances with $\epsilon = 0.9$.

Table 5
Large instances from the literature

Name of the instance	Number of instances	Size $n \times m$	Average gap (%)				Average processing time (s)			
			HDP	SMA	ADP	AGNES	HDP	SMA	ADP	AGNES
Chu and Beasley 1	30	100×5	0.69	2.68	1.72	0.88	<0.001	0.03	0.03	<0.001
Chu and Beasley 2	30	250×5	0.22	1.17	0.58	0.29	0.10	0.57	0.07	<0.001
Chu and Beasley 3	30	500×5	0.08	0.59	0.26	0.12	0.53	4.57	0.30	0.10
Chu and Beasley 4	30	100×10	1.22	3.60	1.97	1.54	0.03	0.03	0.03	<0.001
Chu and Beasley 5	30	250×10	0.46	1.60	0.76	0.57	0.17	0.70	0.10	0.03
Chu and Beasley 6	30	500×10	0.21	0.80	0.38	0.26	0.83	5.70	0.43	0.13
Chu and Beasley 7	30	100×30	2.04	5.13	2.70	3.22	1.67	0.10	0.07	0.03
Chu and Beasley 8	30	250×30	0.89	2.60	1.18	1.41	9.87	1.40	0.37	0.10
Chu and Beasley 9	30	500×30	0.48	1.45	0.58	0.72	26.37	11.93	1.20	0.30

Please cite this article in press as: V. Boyer et al., Heuristics for the 0–1 multidimensional knapsack problem, European Journal of Operational Research (2008), doi:10.1016/j.ejor.2007.06.068

Table 6
Small instances from the literature

Name of the instance	Number of instances	Average gap to optimality (%)	
		HDP	HDP + LBC
Petersen	7	0.02	0.02
Hansen and Plateau	2	0.45	0.45
Weingartner	8	0.01	0.00
Freville and Plateau	6	0.44	0.20
Fleisher	1	0.00	0.00
Sent	2	0.00	0.00

5.1.2. Randomly generated instances

We have compared processing time in function of the number n of variables of (MKP) and ϵ. For a given number n of variables we have generated randomly 25 instances in order to compute average performance of our heuristics.

The average performance is represented in Figs. 1 and 2, respectively for $\epsilon = 0.5$ and $\epsilon = 0.9$. Computational experiences show that we have obtained better bounds than other heuristics, but with $\epsilon = 0.5$, processing time tends to be important. That is not surprising as HDP is constructed with a dynamic-programming algorithm.

5.2. HDP + LBC heuristic

Adding LBC procedure leads to greater computing time but permits one to have a better approximation of the optimal bound. In this section we will compare the bounds provided by HDP + LBC with those given by HDP alone.

5.2.1. Instances from the literature

Table 6 shows that for the instances of Freville and Plateau, LBC improves significantly the lower bounds. For the other instances, as the bound provided by HDP is very close to the optimal one, it is hard to improve it. Processing time to solve all these instances at once is about 1 seconds. For the large instances presented in Table 7, HDP + LBC improves in all cases the bound provided by HDP. Note that, in order to limit the processing time, we stop LBC if the time spend in this procedure exceeds the one of HDP.

5.2.2. Randomly generated instances

According to the results presented in Figs. 3 and 4, we can see that the improvement of the lower bounds is better for instances with a small capacity. Then, when ϵ is close to 0.9, we cannot use LBC procedure, but sometimes it permits one to reduce the gap with the optimal solution. As the bound provided by HDP is close to the optimal one, it is not easy to improve it, then, in some cases, the LBC procedure could not give a better approximation. Although processing time seems to be important, we have on average 3 times less time than an exact method with a good approximation of the optimal solution.

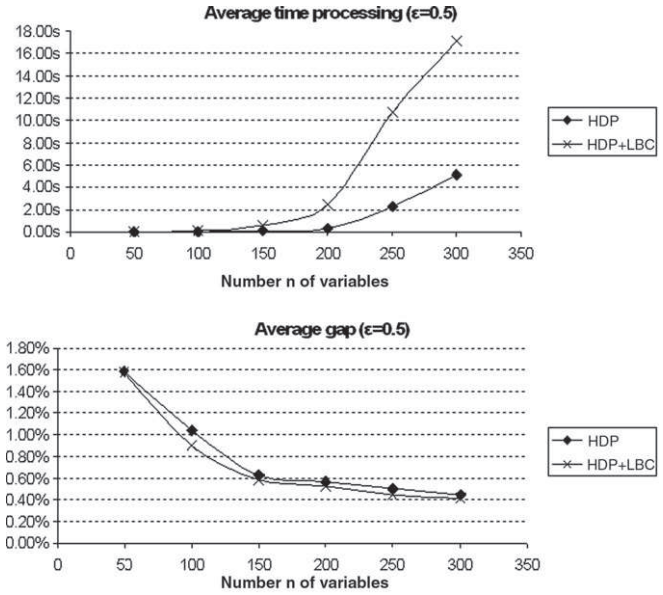


Fig. 3. Average performances with $\epsilon = 0.5$.

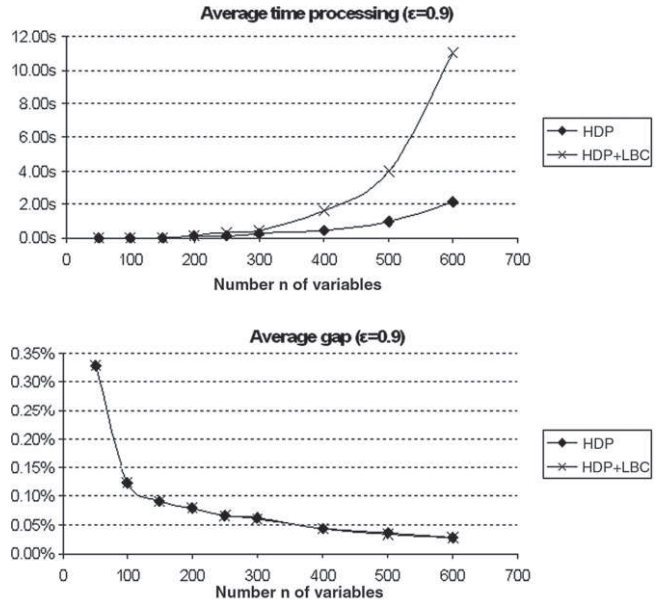


Fig. 4. Average performances with $\epsilon = 0.9$.

Table 7
Large instances from the literature

Name of the instance	Number of instances	Size $n \times m$	Average gap (%)		Average processing time (s)	
			HDP	HDP + LBC	HDP	HDP + LBC
Chu and Beasley 1	30	100×5	0.69	0.57	<0.001	0.60
Chu and Beasley 2	30	250×5	0.22	0.16	0.10	1.00
Chu and Beasley 3	30	500×5	0.08	0.07	0.53	1.13
Chu and Beasley 4	30	100×10	1.22	0.95	0.03	1.00
Chu and Beasley 5	30	250×10	0.46	0.32	0.17	0.97
Chu and Beasley 6	30	500×10	0.21	0.16	0.83	4.03
Chu and Beasley 7	30	100×30	2.04	1.81	1.67	3.97
Chu and Beasley 8	30	250×30	0.89	0.77	9.87	21.20
Chu and Beasley 9	30	500×30	0.48	0.42	26.37	93.37

Please cite this article in press as: V. Boyer et al., Heuristics for the 0–1 multidimensional knapsack problem, European Journal of Operational Research (2008), doi:10.1016/j.ejor.2007.06.068

6. Conclusion

The main idea of HDP is to obtain a processing time similar to the one of dynamic-programming algorithm applied to a classical (UKP) while having good performance in terms of gap. Removing the improvement procedure could sometimes diminish the processing time by two but deteriorates significantly the lower bound. Nevertheless HDP seems to be a good heuristics as it gives a better solution than the existing ones with a quite good processing time.

Using a procedure like LBC improves the lower bound obtained by HDP with a smaller computing time than an exact method. Computing results with instances from the literature show that we increase the occurrence of obtaining the optimal solution from 76% up to 84%. Then using HDP + LBC permits one to have a good approximation of the optimal bound.

A work on the robustness of the algorithm has to be done, by studying a good preprocessing procedure for example in order to reduce the processing time.

References

[1] OR-Library, J.E. Beasley, <http://www.people.brunel.ac.uk/mastjjb/jeb/orlib/mknapinfo.html>.
[2] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, Springer, 2004.
[3] S. Martello, D. Pisinger, P. Toth, New trends in exact algorithms for the 0–1 knapsack problem, European Journal of Operational Research 123 (2000) 325–332.
[4] S. Martello, P. Toth, Knapsack Problems – Algorithms and Computer Implementations, Wiley and Sons, 1990.
[5] D. El Baz, M. Elkihel, Load balancing methods and parallel dynamic programming algorithm using dominance technique applied to the 0–1 knapsack problem, Journal of Parallel and Distributed Computing 65 (2005) 74–84.
[6] G. Plateau, M. Elkihel, A hybrid method for the 0–1 knapsack problem, Methods of Operations Research 49 (1985) 277–293.
[7] A. Freville, G. Plateau, An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem, Discrete Applied Mathematics 49 (1994) 189–212.
[8] D. Bertsimas, R. Demir, An approximate dynamic-programming approach to multi-dimensional knapsack problem, Management Science 4 (2002) 550–565.
[9] S. Hanafi, A. Freville, A. El Abdellaoui, Comparison of heuristics for the 0–1 multidimensional knapsack problem, Meta-Heuristics: Theory and Application, Kluwer Academic, 1996. pp. 446–465.
[10] V. Boyer, Méthodes et/ou mixte pour la programmation linéaire en variables 0–1, DEA report, LAAS-CNRS Toulouse (2004).
[11] M. Elkihel, Programmation dynamique et rotations de contraintes pour les problèmes d'optimisation entière, Ph.D Thesis, Université des Sciences et Techniques de Lille, 1984.
[12] G. Plateau, Contribution à la résolution des programmes mathématiques en nombres entiers, Thèse d'Etat de l'Université des Sciences et Techniques de Lille (1979).
[13] A. Freville, The multidimensional 0–1 knapsack problem: An overview, European Journal of Operational Research 155 (2004) 1–21.
[14] A. Freville, G. Plateau, An exact search for the solution of the surrogate dual of the 0–1 bidimensional knapsack problem, European Journal of Operational Research 68 (1993) 413–421.
[15] B. Gavish, H. Pirkul, Efficient algorithms for solving multiconstraint 0–1 knapsack problems to optimality, Mathematical Programming 31 (1985) 78–205.
[16] S. Garfinkel, L. Nemhauser, Integer Programming, Wiley Interscience, 1972.
[17] F. Glover, Surrogate constraints, Operations Research 16 (1968) 741–749.
[18] M. Osario, F. Glover, P. Hammer, Cutting and surrogate analysis for improved multidimensional knapsack problem, Annal of Operations Research 117 (2002) 71–93.
[19] L. Nemhauser, A. Wolsey, Integer and Combinational Optimization, Wiley Interscience Publication, 1988.
[20] D. Sherali, J. Driscoll, Evolution and state-of-the-art in integer programing, Journal of Computational and Applied Mathematics 124 (2000) 319–340.

Please cite this article in press as: V. Boyer et al., Heuristics for the 0–1 multidimensional knapsack problem, European Journal of Operational Research (2008), doi:10.1016/j.ejor.2007.06.068

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

AN EXACT COOPERATIVE METHOD FOR SOLVING THE 0-1 MULTIDIMENSIONAL KNAPSACK PROBLEM

V. BOYER, Didier EL BAZ, Moussa ELKIHEL

LAAS-CNRS, Université de Toulouse, 7, Avenue du Colonel Roche - 31077 Toulouse Cedex 4
vboyer@laas.fr, elbaz@laas.fr, elkihel@laas.fr

ABSTRACT: *This article presents an exact cooperative method for solving the Multidimensional Knapsack Problem (MKP) which combines dynamic programming and branch and bound. The first step of our algorithm tries to find out a good feasible solution of the (MKP) using surrogate relaxation. For this purpose, we have developed a modified dynamic programming algorithm. The second step is based on a branch and bound procedure. Our algorithm was tested for several randomly generated test sets and problems in the literature. Solutions obtained with the first step are compared with results provided by other existing heuristics, finally our method is compared with a branch and bound algorithm.*

KEY WORDS: *Multidimensional Knapsack Problems, Dynamic Programming, Branch and Bound, Surrogate Relaxation, Cooperative Method.*

1. INTRODUCTION

The NP-hard multidimensional knapsack problem (MKP) arises in several practical contexts such as the capital budgeting, cargo loading, cutting stock problems and processors allocation in huge distributed systems.

A multidimensional knapsack is defined by its capacities $(c_1, ..., c_m)$, $m \in \mathbb{N}$, and n items have to be placed in. To an item $j \in N = \{1, 2, ..., n\}$, the following variables and vectors are associated:

- the decision variable $x_j \in \{0, 1\}$ ($x_j = 1$ if the item j is placed in the knapsack, and $x_j = 0$ otherwise),
- the profit $p_j \geq 0$ and
- the weights $w_{i,j} \geq 0$, $i \in M = \{1, ..., m\}$.

Then, the multidimensional knapsack problem can be written as follows:

$$(MKP) \begin{cases} \max \sum_{j \in N} p_j.x_j, \\ \text{s.t.} \sum_{j \in N} w_{i,j}.x_j \leq c_i, \ \forall i \in M, \\ x_j \in \{0, 1\}, \ \forall j \in N. \end{cases} \quad (1)$$

In the sequel, we shall use the following notation: given a problem (P), its optimal value will be denoted by $v(P)$. $\overline{v}(P)$ and $\underline{v}(P)$ will represent, respectively, the value of an upper and a lower bound for $v(P)$.

To avoid any trivial solutions, we assume that:

- $\forall j \in N$ and $\forall\ i \in M,\ w_{i,j} \leq c_i$.
- $\forall i \in M,\ \sum_{j=1}^{n} w_{i,j} > c_i$.

A special case of (MKP) is the classical knapsack problem (with m=1). The Knapsack Problem (KP) has been given a lot of attention in the literature though it is not, in fact, as difficult as (MKP), more precisely, it can be solved in a pseudo-polynomial time (see (Kellerer & al. 2004) and (Plateau & Elkihel 1985)). Due to the intrinsic difficulty that is NP-hardness of (MKP), we have tried to transform the original (MKP) into a (KP) (see also (Gavish & Pirkul 1985) and (Glover 1968)). To this purpose, we have used a relaxation technique, that is to say, surrogate relaxation.

In the sequel, we propose an efficient algorithm based on dynamic programming in order to find out a good lower bound of (MKP) by solving a surrogate relaxation, and we show how to complete this heuristics with a branch and bound procedure in order to construct an exact method for solving (MKP).

The main steps of our algorithm can be presented as follows:

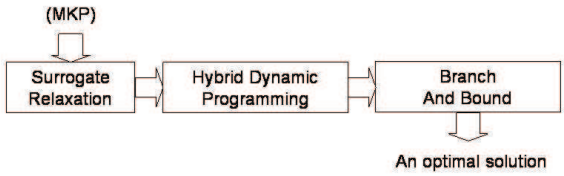


Figure 1. Computational scheme.

Section 2 deals with the construction of the surrogate constraint. In Section 3, we present the hybrid dynamic programming algorithm (HDP). Section 4 deals with the the exact cooperative method. Finally, in section 5, we display and analyze some computational results obtained for different problems from the literature and randomly generated problems.

2. THE SURROGATE RELAXATION

The surrogate relaxation of (MKP) can be defined as follows:

$$(S(u)) \begin{cases} \max \sum_{j\in N} p_j.x_j, \\ \text{s.t. } \sum_{i\in M} u_i.\sum_{j\in N} w_{i,j}.x_j \leq \sum_{i\in M} u_i.c_i, \\ x_j \in \{0,1\},\ \forall j \in N, \end{cases} \quad (2)$$

where $u^T = (u_1, ..., u_m) \geq 0$.

Since $(S(u))$ is a relaxation of (MKP), we have $v(S(u)) \geq v(MKP)$, and the optimal multiplier vector, u^*, is defined by:

$$v(S(u^*)) = \min_{u\geq 0}\{v(S(u))\}. \quad (3)$$

Since solving (3) is a NP_hard problem, several heuristics have been proposed in order to find good surrogate multipliers (see in particular (Freville & Plateau 1993), (Gavish & Pirkul 1985) and (Glover 1968)). In practice, it is not important to obtain the optimal multiplier vector, since in the general case we have no guarantee that $v(S(u^*)) = v(MKP)$. A reasonable estimation can be computed by dropping the integrality restrictions in x. In other words, let

$$(LS(u)) \begin{cases} \max \sum_{j\in N} p_j.x_j, \\ \text{s.t. } \sum_{i\in M} u_i \sum_{j\in N} w_{i,j}.x_j \leq \sum_{i\in M} u_i.c_i, \\ x_j \in [0,1],\ \forall j \in N. \end{cases} \quad (4)$$

be the continuous surrogate relaxation.

The optimal continuous surrogate multipliers are derived from u^0, where:

$$v(LS(u^0)) = \min_{u\geq 0}\ v(LS(u)). \quad (5)$$

In order to compute u^0, we consider the linear programing problem (LP) corresponding to (MKP):

$$(LP) \begin{cases} \max \sum_{j\in N} p_j.x_j, \\ \text{s.t.} \sum_{j\in N} w_{i,j}.x_j \leq c_i,\ \forall i \in M, \\ x_j \in [0,1],\ \forall j \in N. \end{cases} \quad (6)$$

We denote by $\lambda^0 = (\lambda_1^0, \lambda_2^0, ..., \lambda_m^0) \geq 0$ the dual optimal variables associated with the constraints

$$\sum_{j\in N} w_{i,j}.x_j \leq c_i,\ i \in M. \quad (7)$$

Then, the optimal continuous surrogate multipliers can be obtained as follows using the equation (5) (see (Garfinkel & Nemhauser 1972) p. 132).

Theorem: The optimal continuous surrogate multiplier vector is generated by $u^0 = \lambda^0$.

Then we have the following order relation $v(LP) = v(LS(u^0)) \geq v(S(u^*)) \geq v(MKP)$ (see (Gavish & Pirkul 1985), (Garfinkel & Nemhauser 1972) p. 130 and (Osario & al. 2002)).

The reader is referred to (Boyer 2004), (Boyer & al. 2006) and (Boyer & al. 2007) for computational studies related to bounds obtained with surrogate relaxation.

3. HYBRID DYNAMIC PROGRAMMING (HDP)

For simplicity of presentation, we will denote in the sequel $\sum_{i\in M} u_i.w_{i,j}$ by w_j and $\sum_{i\in M} u_i.c_i$ by c. Then we have:

$$(S(u)) \begin{cases} \max \sum_{j\in N} p_j.x_j, \\ \text{s.t. } \sum_{j\in N} w_j.x_j \leq c, \\ x_j \in \{0,1\},\ \forall j \in N. \end{cases} \quad (8)$$

We apply the dynamic programming algorithm to $(S(u^0))$ and we keep only the feasible solutions of (MKP). At each steps, $k \in N$, we update a list which is defined as follows:

$$\mathcal{L}_k = \left\{ (w,p) \mid w = \sum_{j=1}^{k} w_j.x_j \leq c,\ p = \sum_{j=1}^{k} p_j.x_j \right\} \quad (9)$$

The use of the concept of dominated states permits one to reduce drastically the size of lists $\mathcal{L}_k$ since dominated states can be eliminated from the list:

Dominated state: Let (w,p) be a couple of weight and profit, i.e. a state of the problem. If $\exists(w',p')$ such that $w' \leq w$ and $p' \geq p$, then (w,p) is dominated by (w',p').

Note that dominated states are saved in a secondary list denoted by $\mathcal{L}_{sec}$ since they can give rise to an optimal solution for (MKP). The states are sorted in $\mathcal{L}_{sec}$ according to their associated upper bound.

Let (w,p) be the state generated at stage k, we define the sub-problem associated with (w,p) by:

$$(S(u))_{(w,p)} \begin{cases} \max \sum_{j=k+1}^{n} p_j.x_j + p, \\ \text{s.t.} \sum_{j=k+1}^{n} w_j.x_j \leq c - w, \\ x_j \in \{0,1\}, j \in \{k+1, ..., n\}. \end{cases} \quad (10)$$

Given a state (w,p), an upper bound, $\overline{v}_{(w,p)}$, is obtained by solving the linear relaxation of $(S(u))_{(w,p)}$, i.e. $(LS(u))_{(w,p)}$, with the Martello And Toth algorithm (see (Martello & Toth 1990)) and a lower bound, $\underline{v}_{(w,p)}$, is obtained with a greedy algorithm on $(S(u))_{(w,p)}$.

In a list, all the states are ordered according to their decreasing upper bound. As mentioned above, our algorithm consists in applying dynamic programming (DP) to $S(u^0)$. At each stage of dynamic programming, we check the following points at the creation of a new state (w,p):

- Is the state feasible for (MKP) (this will permit one to eliminate the unfeasible solutions)? Then, we try to improve the lower bound of (MKP), $\underline{v}(MKP)$, with the value of p.
- Is the state dominated? In this case the state is saved in the secondary list $\mathcal{L}_{sec}$.
- Is the upper bound of the state (w,p) smaller than the current lower bound of $S(u^0)$? Then the state is saved too in the secondary list $\mathcal{L}_{sec}$.

For each state (w,p) which has not been eliminated or saved in the secondary list after these tests, we try to improve the lower bound of $(S(u^0))$, i.e. $\underline{v}(S(u^0))$, by computing a lower bound of the state with a greedy algorithm.

Dynamic programming algorithm is described below:

Dynamic Programming Algorithm (DP):

Initialisation:

$\mathcal{L}_0 = \{(0,0)\}$, $\mathcal{L}_{sec} = \emptyset$

$\underline{v}(S(u^0)) = \underline{v}(MKP)$ *(where* $\underline{v}(MKP)$ *is a lower bound of* (MKP) *given by a greedy algorithm)*

Computing the lists:

For j:=1 to n

$\mathcal{L}'_{j-1}$:=$\{(w+w_j, p+p_j) \mid (w,p) \in \mathcal{L}_{j-1}\}$;
Remove all states $(w,p) \in \mathcal{L}'_{j-1}$ *which are unfeasible for* (MKP);
$\mathcal{L}_j$:=*MergeLists(*$\mathcal{L}_{j-1}$, $\mathcal{L}'_{j-1}$*);*

For each state $(w,p) \in \mathcal{L}_j$
Compute $\overline{v}_{(w,p)}$ *and* $\underline{v}_{(w,p)}$;
End For;

Updating the bounds:
p_{max}:=*max* $\{p \mid (w,p) \in \mathcal{L}_j\}$ *and*
v_{max}:=*max* $\{\underline{v}_{(w,p)} \mid (w,p) \in \mathcal{L}_j\}$;
$\underline{v}(MKP)$:=*max* $\{\underline{v}(MKP),\ p_{max}\}$;
$\underline{v}(S(u^0))$:=*max* $\{\underline{v}(S(u^0)),\ v_{max}\}$;

Updating $\mathcal{L}_{sec}$*:*
$\mathcal{D}$:=$\{(w,p) \in \mathcal{L}_j \mid (w,p)$ *is dominated or* $\overline{v}_{(w,p)} \leq \underline{v}(S(u^0))\}$;
$\mathcal{L}_{sec}$:=$\mathcal{L}_{sec} \cup \mathcal{D}$ *and* $\mathcal{L}_j$:=$\mathcal{L}_j - \mathcal{D}$;

End for.

At the end of the algorithm, we obtain a lower bound of (MKP), i.e. $\underline{v}(MKP)$. In order to improve this lower bound and the efficiency of DP algorithm, we add to the algorithm a reducing variable process, which is defined as follow:

Reducing variables rule 1: Let $\underline{v}$ be a lower bound of (MKP) and v_j^0, v_j^1, respectively, be the upper bounds of (MKP) with $x_j = 0$, $x_j = 1$,

respectively. If $\underline{v} > v_j^k$ with $k = 0$ or 1, then we can definitively fix $x_j = 1 - k$.

The upper bounds, v_j^0 and v_j^1, $j \in N$, are obtained via the Martello and Toth algorithm on $(S(u^0))$. We use this reducing variables rule whenever we improve $\underline{v}(MKP)$ during the Dynamic Programming Phase. When a variable is fixed, we have to update all the states of the active list and to eliminate all the states which do not match the fixed variables or which become unfeasible.

We present now a procedure that allows us to improve significantly the lower bound given by DP algorithm. More precisely, we try to obtain better lower bounds for the states saved in the secondary list. Before calculating these bounds, we eliminate all the states that have become unfeasible or which are incompatible with the variables that have been yet reduced or that have an upper bound smaller than the current lower bound of (MKP), i.e. $\underline{v}(MKP)$.

For a state (w, p), let J be the index of free variables. If the states has been generated at the k-th stage of DP Algorithm, $J = \{k+1, ..., n\}$, $w = \sum_{j=1}^{k} w_j.x_j$ and $p = \sum_{j=1}^{k} p_j.x_j$. Then we defined the new subproblem:

$$(MKP)_{(w,p)} \begin{cases} \max \sum_{j \in J} p_j.x_j + p, \\ \text{s.t.} \sum_{j \in J} w_{i,j}.x_j \leq \overline{c}_i, \ \forall i \in M, \\ x_j \in \{0,1\}, \ \forall j \in J, \end{cases} \quad (11)$$

where $\overline{c}_i = c_i - \sum_{j=1}^{k} w_{i,j}.x_j, \ \forall i \in M$.

Two methods are used in order to evaluate the lower bound of a state using the subproblem defined above according to the reduced variables:

- a greedy algorithm;
- an enumerative method when the number $n' = n-k$ of variables of the subproblem is sufficiently small (given by the parameter α: $n' \leq \alpha$).

When all the states are treated the process stops. The detail of the algorithm is given in what follows:

Procedure ILB:

Assign to $\underline{v}(MKP)$ the value of the lower bound returned by DP algorithm;

For each state $(w, p) \in \mathcal{L}_{sec}$
Compute $\underline{v}_{(w,p)}$ a lower bound of $(MKP)_{(w,p)}$;
End For;

v_{max}:=*max* $\{\underline{v}_{(w,p)} \mid (w, p) \in \mathcal{L}_{sec}\}$;

$\underline{v}(MKP)$:=*max* $\{\underline{v}(MKP), v_{max}\}$.

The combination of the ILB procedure with the DP algorithm gives the so-called HDP heuristics.

4. COOPERATIVE METHOD (CM)

As mentioned above, the secondary list $\mathcal{L}_{sec}$ can contain an optimal solution of (MKP). We propose an algorithm based on a branch and bound method in order to explore the list $\mathcal{L}_{sec}$.

4.1. Principle

Let (w, p) be the first state of $\mathcal{L}_{sec}$ (the first state corresponds to the largest upper bound). An upper bound, $\overline{v}_{(w,p)}$, is obtained by solving the linear relaxation of $(MKP)_{(w,p)}$, using a simplex algorithm. A lower bound, $\underline{v}_{(w,p)}$, is obtained with a greedy algorithm on $(MKP)_{(w,p)}$.
We propose the following branching strategy:

Branching rule: Let (w, p) be a state of the problem (MKP), J the index of the free variables (the variables that have not been already fixed by the branch and bound) and $\widetilde{X_J} = \{\widetilde{x_j} \mid j \in J\}$ an optimal solution of the linear relaxation of $(MKP)_{(w,p)}$. Then, the branching variable x_k, $k \in J$, is such that $k = \arg\min_{j \in J}\{|\widetilde{x_j} - 0.5|\}$.

Whenever we evaluate an upper bound, we use the following reducing variables rule (see (Nemhauser & Wolsey 1988)):

Reducing variables rule 2: Let $\underline{v}$ be a lower bound of (MKP). Let $\widetilde{v}$ and $\widetilde{x} = \{\widetilde{x_j} \mid j \in N\}$ be respectively the optimal value and an optimal solution of the linear relaxation of (MKP). Then we denote by $\widetilde{p} = \{\widetilde{p_j} \mid j \in N\}$, the reduced profits. For $j \in N$, if $\widetilde{x_j} = 0$, $\widetilde{x_j} = 1$, respectively, and $\widetilde{v} - |\widetilde{p_j}| \leq \underline{v}$ then there exists an optimal solution of (MKP) with $x_j = 0$, $x_j = 1$, respectively.

This last rule permits one to reduce significantly the processing time by reducing the number of states to explore.

4.2. Details of the algorithm

The branch and bound method described above is used in order to explore the states saved in the secondary list $\mathcal{L}_{sec}$ since this list can contain an optimal solution of (MKP).

Procedure BB:

Let $\underline{v}$ be the value of a lower bound of (MKP), and $\mathcal{L}$ a list of states.

While $\mathcal{L} \neq \emptyset$

Let (w, p) be the first state in $\mathcal{L}$;

$\mathcal{L} := \mathcal{L} - \{(w, p)\}$;

Compute $\overline{v}_{(w,p)}$ an upper bound of $(MKP)_{(w,p)}$;

If $\overline{v}_{(w,p)} > \underline{v}$

Fix variables according to reducing variables rule 2 and update the state (w, p);

Compute $\underline{v}_{(w,p)}$ a lower bound of $(MKP)_{(w,p)}$;

If $\underline{v}_{(w,p)} > \underline{v}$, $\underline{v} := \underline{v}_{(w,p)}$ Endif;

Chose the branching variable and branch on it;

Insert the two resulting states in $\mathcal{L}$ if they are feasible;

Endif;

Endwhile.

The combination of HDP with the procedure BB permits one to obtain an exact solution; it corresponds to the so-called cooperative method (CM).

Procedure CM:

Step 1:
Compute $\mathcal{L}_{sec}$ and $\underline{v}(MKP)$ using HDP heuristics.

Step 2:
Use procedure BB with $\underline{v} = \underline{v}(MKP)$ and $\mathcal{L} = \mathcal{L}_{sec}$.

The last value of $\underline{v}$ returned by BB is the optimal value of (MKP).

5. COMPUTATIONAL EXPERIENCES

Our algorithm was written in C and compiled with GNU's GCC. Computational experiences were carried out using a Sun Blade 100 (500 MHz). We compare first our heuristics HDP to the following heuristics of the literature:

- AGNES of Fréville and Plateau (Freville & Plateau 1994);
- ADP-based heuristics approach of Bertsimas and Demir (Bertsimas & Demir 2002);
- Simple Multistage Algorithm (SMA) of Hanafi, Fréville and El Abdellaoui (Hanafi & al. 1996).

Our tests were made on the following problems:

- Various problems from the literature of Chu and Beasley (see (Beasley 1990)) composed of 9 instances of 30 problems with different sizes (100x5, 250x5, 500x5, 100x10, 250x10, 500x10, 100x30, 250x30 and 500x30), numbered respectively from 1 to 9;
- Randomly generated problems with:
  - uncorrelated data: the value of the profits and the weights are distributed independently and uniformly over $[1, 1000]$,
  - correlated data: the value of the weights are distributed uniformly over $[1, 1000]$ and the profits are taken as follows:

$$\forall j \in N,\ p_j = \frac{\sum_{k=1}^{m} w_{k,j}}{m} + 100.$$

The capacity c of the knapsack is generated as follows: $\forall i \in M,\ c_i = 0.5. \sum_{j \in N} w_{i,j}$.

5.1. HDP heuristics

The computational results for the HDP heuristics are presented in:

- tables 1 and 2, for the instance of Chu & Beasley,
- tables 3 and 4, for randomly generated instance.

Some results for the DP heuristics are presented in tables 1 and 2.

Inst.	heuristics				
	DP	HDP	SMA	ADP	AGNES
1	1.96	0.69	2.68	1.72	0.88
2	0.58	0.21	1.17	0.58	0.29
3	0.27	0.07	0.59	0.26	0.12
4	2.87	1.25	3.6	1.97	1.54
5	1.03	0.47	1.6	0.76	0.57
6	0.54	0.21	0.8	0.38	0.26
7	4.23	2.05	5.13	2.7	3.22
8	1.7	0.9	2.6	1.18	1.41
9	1.39	0.49	1.45	0.58	0.72

Table 1: Heuristics: problems of Chu and Beasley (gap to optimal value (%)).

Inst.	heuristics				
	DP	HDP	SMA	ADP	AGNES
1	0.03	0.07	0.15	0.12	0.10
2	0.27	0.52	1.94	0.24	0.10
3	1.50	2.07	15.63	1.03	0.34
4	0.05	0.12	0.17	0.15	0.10
5	0.45	0.94	2.39	0.34	0.10
6	2.36	3.81	19.49	1.47	0.44
7	2.49	5.36	0.39	0.24	0.10
8	22.26	36.66	4.79	1.27	0.34
9	81.31	88.07	40.80	4.10	1.03

Table 2: Heuristics: problems of Chu and Beasley (computational time (s)).

From Tables 1 and 3, we note that the lower bound given by HDP is better than the one obtained with other methods. According to tables 2 and 4 the bounds provided by HDP are obtained at the price of reasonable computational time.

5.2. Exact methods

In this section, we compare computational results obtained with CM with the one obtained by using the branch and bound method (BB). Note that if computational time exceeds 10 minutes, then the methods stop and return the best value of lower bound they have obtained. In order to compare these bounds, the

Inst.	size	heuristics			
	nxm	HDP	SMA	ADP	AGNES
UD	50x25	1.81	5.13	3.31	4.46
UD	100x50	1.19	3.29	1.53	2.86
UD	150x75	0.72	2.15	1.05	1.90
UD	200x100	0.56	1.77	0.78	1.50
UD	250x125	0.52	1.64	0.71	1.53
UD	300x150	0.50	1.48	0.55	1.34
UD	400x200	0.45	1.20	0.48	0.94
UD	500x250	0.36	1.08	0.44	0.87
CD	50x5	1.75	6.43	3.48	4.12
CD	100x10	1.07	3.51	1.50	2.55
CD	150x15	1.15	2.28	1.44	2.85
CD	200x20	0.99	2.05	1.07	2.18
CD	250x25	0.97	1.64	0.98	1.92

UD: Instance with Uncorrelated Data
CD: Instance with Correlated Data

Table 3: Heuristics: Randomly generated problems (gap to optimal value (%)).

Inst.	size	heuristics			
	nxm	HDP	SMA	ADP	AGNES
UD	50x25	0.03	0.07	0.10	0.02
UD	100x50	0.22	0.78	0.53	0.10
UD	150x75	0.50	3.90	1.09	0.30
UD	200x100	3.06	11.39	3.51	0.65
UD	250x125	9.77	25.45	7.35	1.33
UD	300x150	84.46	57.27	14.85	2.44
UD	400x200	227.96	171.74	41.93	5.91
UD	500x250	519.10	1110.55	80.52	12.42
CD	50x5	0.64	0.04	0.06	0.03
CD	100x10	23.26	0.36	0.41	0.18
CD	150x15	30.05	1.41	1.41	0.62
CD	200x20	42.48	3.53	3.53	1.46
CD	250x25	80.90	7.15	7.15	2.81

UD: Instance with Uncorrelated Data
CD: Instance with Correlated Data

Table 4: Heuristics: Randomly generated problems (computational time (s)).

Inst.	Gap (%)	t_BB (s)	t_CM(s)
1	0,00	166,60	160,06
2	0,0038	501,61	493,56
3	-0,0146	600,00	600,00
4	0,00	533.95	600,00
5	-0,0157	600,00	600,00
6	-0,0525	600,00	600,00
7	-0,0602	600,00	600,00
8	0,0158	600,00	600,00
9	-0.0223	600,00	600,00

Gap: gap between CM & BB
t_BB: BB computational time
t_CM: CM computational time

Table 5: CM exact method: problems of Chu and Beasley

gaps displayed is defined as follows:

$$Gap = \frac{v_{BB} - v_{CM}}{v_{BB}}, \tag{12}$$

where v_{BB} and v_{CM} are the value of the bound delivered by, respectively, BB and CM. Of course, when the computational times are under 10 minutes, $v_{BB} = v_{CM} = v(MKP)$, the optimal value, and $Gap = 0$.

We present first preliminary results for problems in the literature and randomly generated problems.

Table 5 and Table 6 show that the computational times for BB and CM are similar when they do not exceed 10 minutes. Concerning the gap, we note that it is, in most cases, negative, that is to say, when we stop the process when it exceed 10 minutes, CM deliver a better bound than BB. According to these results, CM seems to converge more rapidly toward the optimal value than BB.

6. CONCLUSION

The main advantage of the HDP heuristics is to obtain a processing time similar to the one of dynamic programming algorithm applied to a classical (KP) while having good performance in terms of gap. HDP seems to be a good heuristics since it gives better solutions than the one obtained with other heuristics with a quite good processing time.

Inst.	nxm	Gap (%)	t_BB (s)	t_CM (s)
UD	50x25	0.00	0,35	0,33
UD	100x50	0.00	0,81	0,99
UD	150x75	0.00	328,73	329,60
UD	200x100	-0.01	600,00	600,00
UD	300x150	-0.01	600,00	600,00
UD	400x200	0.00	600,00	600,00
UD	500x250	-0.01	600,00	600,00
CD	50x5	-0.02	600,00	600,00
CD	100x10	-0.10	600,00	600,00
CD	150x15	-0.05	600,00	600,00

UD: Instance with Uncorrelated Data
CD: Instance with Correlated Data
Gap: gap between CM & BB
t_BB: BB computational time
t_CM: CM computational time

Table 6: CM exact method: randomly generated problems.

Combining a procedure like BB (Branch and Bound) with HDP permits one to obtain an exact method. Computing experimentation on problems from the literature shows that the combination of HDP and BB gives the same processing times similar to the one of a classical branch and bound. However, this cooperative method seems to improve the convergence toward the optimal value.

HDP could be combined easily with other methods, like a Taboo search for example, in order to improve its performances to explore the neighborhood of the states saved in the secondary list. That solution could be an alternative to limit the processing time.

REFERENCES

Beasley, J. E. (1990). Or-library: http://people.brunel.ac.uk/˜mastjjb/jeb/orlib/mknapinfo.html.

Bertsimas, D. & Demir, R. (2002). An approximate dynamic-programming approach to multidimensional knapsack problem, *Management Science* **4**: 550–565.

Boyer, V. (2004). Méthodes et/ou mixte pour la programmation linéaire en variables 0-1, DEA report. LAAS-CNRS Toulouse (France).

Boyer, V. & al. (2006). An efficient heuristics for the multidimensional knapsack problem, *ROADEF'06, Presses Universitaires de Valenciennes* pp. 95–106.

Boyer, V. & al. (2007). Heuristics for the 0-1 multidimensional knapsack problem, European Journal of Operational Research. to appear.

Elkihel, M. (1984). Programmation dynamique et rotations de contraintes pour les problèmes d'optimisation entière, Thèse de Doctorat. Université des Sciences et Techniques de Lille (France).

Freville, A. & Plateau, G. (1993). An exact search for the solution of the surrogate dual of the 0-1 bidimensional knapsack problem, *European Journal of Operational Research* **68**: 413–421.

Freville, A. & Plateau, G. (1994). An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem, *Discrete Applied Mathematics* **49**: 189–212.

Fréville, A. (2004). The multidimensional 0-1 knapsack problem: An overview, *European Journal of Operational Research* **155**: 1–21.

Garey, M. R. & Jonhson, D. S. (1979). Computer and intractability. a guide to the theory of np-completeness, ISBN 0-7167-1044-7.

Garfinkel, S. & Nemhauser, L. (1972). *Integer Programming*, Wiley Interscience.

Gavish, B. & Pirkul, H. (1985). Efficient algorithms for solving multiconstraint 0-1 knapsack problems to optimality, *Mathematical Programming* **31**: 78–205.

Glover, F. (1968). Surrogate constraints, *Operations Research* **16**: 741–749.

Hanafi, S. & al. (1996). *Meta-Heuristics: Theory and Application*, Kluwer Academic, chapter Comparaison of heuristics for the 0-1 multidimensional knapsack problem, pp. 446–465.

Kellerer, H. & al. (2004). *Knapsack Problems*, Springer.

Martello, S. & al. (2000). New trends in exact algorithms for the 0-1 knapsack problem, *European Journal of Operational Research* **123**: 325–332.

Martello, S. & Toth, P. (1990). *Knapsack Problems - Algorithms and Computer Implementations*, Wiley & Sons.

Nemhauser, L. & Wolsey, A. (1988). *Integer and combinational optimization*, Wiley Interscience.

Osario, M. & al. (2002). Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions, *Annals of Operations Research* **117**: 71–93.

Plateau, G. (1979). Contribution à la résolution des programmes mathématiques en nombres entiers, Thèse de Doctorat. Université des Sciences et Techniques de Lille.

Plateau, G. & Elkihel, M. (1985). A hybrid method for the 0-1 knapsack problem, *Methods of Operations Research* **49**: 277–293.

Poirriez, V. & Andonov, R. (1998). Unbounded knapsack problem: new results, *Algorithms and Experiments*, pp. 103–111.

Sherali, D. & Driscoll, J. (2000). Evolution and state-of-the-art in integer programing, *Journal of Computationnal and Applied Mathematics* **124**: 319–340.

6.2. Copie de diplômes et rapport de soutenance

– Copie du diplôme de doctorat
– Rapport de soutenance
– Copie du diplôme d'ingénieur
– Copie du diplôme de DEA

R É P U B L I Q U E F R A N Ç A I S E

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE

INSTITUT NATIONAL DES SCIENCES APPLIQUEES DE TOULOUSE

DOCTORAT

Vu le code de l'éducation, notamment son article L. 612-7 ;

Vu le code de la recherche, notamment son article L. 412-1 ;

Vu le décret n° 2002-481 du 8 avril 2002 relatif aux grades et titres universitaires et aux diplômes nationaux ;

Vu l'arrêté du 3 septembre 1998 relatif à la charte des thèses ;

Vu l'arrêté du 27 juin 1985 modifié fixant la liste des établissements autorisés à délivrer, seuls, le doctorat ;

Vu l'arrêté du 6 janvier 2005 relatif à la cotutelle internationale de thèse ;

Vu l'arrêté du 7 août 2006 relatif à la formation doctorale ;

Vu les pièces justificatives produites par M. VINCENT BOYER, né le 21 juillet 1980 à SAINT-DENIS (974), en vue de son inscription au doctorat ;

Vu le procès-verbal du jury attestant que l'intéressé a soutenu, le 14 décembre 2007 une thèse portant sur le sujet suivant : CONTRIBUTION A LA PROGRAMMATION EN NOMBRE ENTIER, préparée au sein de l'école doctorale Systèmes, devant un jury présidé par SAID HANAFI, Professeur des universités et composé de DIDIER EL BAZ, Chargé de recherche, MOUSSA ELKIHEL, Maître de conférence, MHAND HIFI, Professeur des universités, JEAN-BERNARD LASSERRE, Directeur de recherche, GERARD PLATEAU, Professeur des universités ;

Vu la délibération du jury ;

Le **DIPLÔME DE DOCTEUR** EN SYSTEMES AUTOMATIQUES

est délivré à **M. VINCENT BOYER**

et confère **le grade de docteur**,

pour en jouir avec les droits et prérogatives qui y sont attachés.

Fait à Toulouse, le 12 mars 2009

Le titulaire

Le Directeur

Louis CASTEX

Le Recteur d'Académie,
Chancelier des universités

Olivier DUGRIP

N° INSATOU 5857859
/2009200700588

Université de Toulouse

INSA TOULOUSE

RAPPORT SUR LA SOUTENANCE DU DIPLOME DE DOCTORAT DE L'UNIVERSITE DE TOULOUSE DELIVRE PAR L'INSA DE TOULOUSE

Spécialité : Systèmes Automatiques

de M. Vincent Boyer

soutenue le 14 Décembre 2007

Université de Toulouse

Monsieur Vincent Boyer a fait une présentation claire et précise de sa contribution aux méthodes coopératives pour la résolution de problèmes de sac à dos multi dimensionnel en variables 0-1. D'une part, il a proposé diverses variantes des méthodes coopératives entre la programmation dynamique et la méthode par séparation et évaluation. D'autre part, il a proposé deux nouveaux générateurs d'instances difficiles pour le problème du sac à dos multidimensionnel en variables 0-1.
Monsieur Vincent Boyer a montré qu'il maîtrise les différents aspects de son travail. Le Jury a bien apprécié l'importance du travail d'implémentation et d'expérimentation effectué. Par ailleurs, le candidat a bien répondu aux nombreuses questions du Jury.
Pour toutes ces raisons le Jury décerne à Monsieur Vincent Boyer le grade de Docteur de l'Université de Toulouse délivré par l'INSA de Toulouse spécialité Systèmes Automatiques.

Institut National des Sciences Appliquées * TOULOUSE *

Institut National des Sciences Appliquées * TOULOUSE *

Toulouse, le 14/12/2007

Les Membres du Jury : Visas :

Président : S. Hanafi

Membres : Didier El Baz

M. ELKIHEL

G. PLATEAU

M. HIFI

R É P U B L I Q U E F R A N Ç A I S E

MINISTÈRE DE L'ÉDUCATION NATIONALE , DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE

INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE

ÉCOLE NATIONALE SUPÉRIEURE D'ÉLECTROTECHNIQUE D'ÉLECTRONIQUE D'INFORMATIQUE D'HYDRAULIQUE ET DES TÉLÉCOMMUNICATIONS

DIPLÔME D'INGÉNIEUR

GRADE DE MASTER - MASTER'S DEGREE

Vu le code de l'éducation et notamment son article L.642-1,

Vu le décret n° 62-35 du 16 janvier 1962 modifié, ensemble l'arrêté du 3 octobre 1991 portant délégation d'atribution aux recteurs d'académie,

Vu le décret n° 99-747 du 30 août 1999 modifié relatif à la création du grade de master, notamment son article 2, alinéa 3

Vu l'arrêté du 16 juin 2003 fixant la liste des écoles habilitées à délivrer un titre d'ingénieur diplômé

Vu les procès verbaux du jury attestant que M. VINCENT BOYER né le 21 juillet 1980 à SAINT-DENIS (974) a satisfait à l'ensemble des obligations prévues pour la délivrance du diplôme d'ingénieur,

le titre d'ingénieur DIPLÔMÉ DE L'ÉCOLE NATIONALE SUPÉRIEURE D'ÉLECTROTECHNIQUE, D'ÉLECTRONIQUE, D'INFORMATIQUE, D'HYDRAULIQUE ET DES TÉLÉCOMMUNICATIONS, Spécialité GÉNIE ÉLECTRIQUE ET AUTOMATIQUE

est délivré, au titre de l'année universitaire 2003-2004, à **M. VINCENT BOYER**

à qui est conféré le grade de master.

Fait à Toulouse, le 16 novembre 2004

Le titulaire	*Le Directeur*	*Le Président*	*Le Recteur d'Académie, Chancelier des universités*
	François RODRIGUEZ	Roland MORANCHO	Nicole BELLOUBET-FRIER

N°

/2004200300322

Espace européen de l'enseignement supérieur - European higher education area - *Espace européen de l'enseignement supérieur*

R É P U B L I Q U E F R A N Ç A I S E

MINISTÈRE DE L'ÉDUCATION NATIONALE , DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE

INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE

DIPLÔME D'ÉTUDES APPROFONDIES

GRADE DE MASTER - MASTER'S DEGREE

Vu la loi n° 84-52 du 26 janvier 1984 modifiée sur l'enseignement supérieur, notamment ses articles 5, 16, 17 et 43 ;

Vu le décret n° 84-573 du 5 juillet 1984 modifié relatif aux diplômes nationaux de l'enseignement supérieur,

Vu le décret n° 2002-604 du 25 avril 2002 modifiant le décret n° 99-747 du 30 août 1999 relatif à la création du grade de master

Vu l'arrêté ministériel du 18 février 2000 relatif aux habilitations de l'INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE, de l'ENSICA TOULOUSE, de l'INSA TOULOUSE, de l'UNIVERSITE TOULOUSE 3, de l'ENSAE TOULOUSE, de l'ENI TARBES à délivrer des diplômes d'études approfondies

Vu les pièces justificatives produites par M. VINCENT BOYER, né le 21 juillet 1980 à SAINT-DENIS (974), en vue de son inscription au Diplôme d'Etudes Approfondies SYSTÈMES AUTOMATIQUES

Vu les procès-verbaux du jury attestant que l'intéressé a satisfait au contrôle des connaissances et des aptitudes prévu par les textes réglementaires

le **DIPLÔME D'ÉTUDES APPROFONDIES** SYSTÈMES AUTOMATIQUES, *mention bien*

est décerné à **M. VINCENT BOYER**

à qui est conféré le grade de master

au titre de l'année universitaire 2003-2004.

Fait à Toulouse, le 23 décembre 2004

Le titulaire

Le Président

Roland MORANCHO

Le Recteur d'Académie,
Chancelier des universités

Nicole BELLOUBET-FRIER

N° INPTOU 3190686
/2004200301065

European higher education area - *Espace européen de l'enseignement supérieur* - European higher education area

6.3. Attestations d'enseignements

– Attestation de M. Marcel Mongeau
– Attestation de M. Jean-Baptiste Hiriart-Urruty

INSTITUT de MATHEMATIQUES de TOULOUSE

Marcel Mongeau
Institut de Mathématiques de Toulouse
Université Paul-Sabatier
31062 Toulouse cedex 9
tel. : +33 5 61 55 84 82
mongeau@math.univ-toulouse.fr
www.math.univ-toulouse.fr/~mongeau

Rapporteurs CNU-27
Dossier de demande de qualification de Vincent Boyer

Le 19 novembre 2008

Chers Collègues

Je vous écris pour soutenir la demande de qualification de Vincent Boyer en section 27.

Tout d'abord, j'atteste formellement que Vincent Boyer, ATER à l'Université Paul Sabatier de Toulouse, a enseigné des TD devant machine avec le logiciel de calcul formel et numérique MAPLE dans notre filière *Préparation aux Concours Polytechniques* (PCP) en L2 durant l'année universitaire 2008-2009 et également en 2007-2008, pour un volume de 72 heures chaque année. J'étais responsable de ce module et j'y ai également enseigné en parallèle avec lui. Vincent Boyer a de plus participé activement à l'élaboration des sujets de TD, des contrôles continus, ainsi qu'à leurs corrections.

Vincent Boyer est un enseignant fiable, dynamique et apprécié par ses étudiants. Je recommande chaleureusement sa qualification en section 27.

Cordialement

Marcel Mongeau

INSTITUT
de MATHEMATIQUES
de TOULOUSE

Je, soussigné J.-B. HIRIART-URRUTY (Professeur de mathématiques), atteste que Monsieur Vincent BOYER, ATER, effectue des TD associés à mon Cours de mathématiques en L2 « Sciences de l'Ingénieur », première période de l'année universitaire 2007-2008 et également 2008-2009, pour un volume de 24 heures chaque année.

Toulouse, le 13 octobre 2008

Jean-Baptiste HIRIART-URRUTY
Mathématiques UFR MIG
Université Paul Sabatier (Toulouse III)
118, route de Narbonne
31062 TOULOUSE Cedex 4 - France