N° d'ordre : 1237

UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

THÈSE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir le titre de

DOCTEUR DE 3ème CYCLE

MATHEMATIQUES APPLIQUEES

par

Moussa ELKIHEL

PROGRAMMATION DYNAMIQUE ET ROTATION DE CONTRAINTES POUR LES PROBLEMES D'OPTIMISATION ENTIERE

Thèse soutenue le 7 décembre 1984 devant la Commission d'Examen

MEMBRES DU JURY :

Président :	C. BREZINSKI
Rapporteur :	G. PLATEAU
Examinateurs :	C. CARREZ
	M. MINOUX

P R O F E S S E U R S CLASSE EXCEPTIONNELLE

M. CONSTANT Eugène	I.E.E.A.
M. FOURET René	Physique
M. GABILLARD Robert	I.E.E.A.
M. MONTREUIL Jean	Biologie
M. PARREAU Michel	Mathématiques
M. TRIDOT Gabriel	Chimie
M. VIVIER Emile	Biologie
M. WERTHEIMER Raymond	Physique

P R O F E S S E U R S 1ère classe

M. BACCHUS Pierre	Mathématiques
M. BEAUFILS Jean-Pierre (dét.)	Chimie
M. BIAYS Pierre	G.A.S.
M. BILLARD Jean (dét.)	Physique
M. BOILLY Bénoni	Biologie
M. BOIS Pierre	Mathématiques
M. BONNELLE Jean-Pierre	Chimie
M. BOUGHON Pierre	Mathématiques
M. BOURIQUET Robert	Biologie
M. BREZINSKI Claude	I.E.E.A.
M. CELET Paul	Sciences de la Terre
M. CHAMLEY Hervé	Biologie
M. COEURE Gérard	Mathématiques
M. CORDONNIER Vincent	I.E.E.A.
M. DEBOURSE Jean-Pierre	S.E.S.
M. DYMENT Arthur	Mathématiques

PROFESSEURS 1ère classe (suite)

M. ESCAIG Bertrand	Physique
M. FAURE Robert	Mathématiques
M. FOCT Jacques	Chimie
M. GRANELLE Jean-Jacques	S.E.S.
M. GRUSON Laurent	Mathématiques
M. GUILLAUME Jean	Biologie
M. HECTOR Joseph	Mathématiques
M. LABLACHE COMBIER Alain	Chimie
M. LACOSTE Louis	Biologie
M. LAVEINE Jean Pierre	Sciences de la Terre
M. LEHMANN Daniel	Mathématiques
Mme LENOBLE Jacqueline	Physique
M. LHOMME Jean	Chimie
M. LOMBARD Jacques	S.E.S.
M. LOUCHEUX Claude	Chimie
M. LUCQUIN Michel	Chimie
M. MIGEON Michel Recteur à Grenoble	E.U.D.I.L.
M. MIGNOT Fulbert (dét.)	Mathématiques
M. PAQUET Jacques	Sciences de la Terre
M. PROUVOST Jean	Sciences de la Terre
M. ROUSSEAU Jean-Paul	Biologie
M. SALMER Georges	I.E.E.A.
M. SEGUIER Guy	I.E.E.A.
M. SIMON Michel	S.E.S.
M. STANKIEWICZ François	S.E.S.
M. TILLIEU Jacques	Physique
M. VIDAL Pierre	I.E.E.A.
M. ZEYTOUNIAN Radyadour	Mathématiques

P R O F E S S E U R S 2ème classe

M. ANTOINE Philippe	Mathématiques (Calais)
M. BART André	Biologie
Mme BATTIAU Yvonne	Géographie
M. BEGUIN Paul	Mathématiques
M. BELLET Jean	Physique
M. BERZIN Robert	Mathématiques
M. BKOUCHE Rudolphe	Mathématiques
M. BODARD Marcel	Biologie
M. BOSQ Denis	Mathématiques
M. BRASSELET Jean-Paul	Mathématiques
M. BRUYELLE Pierre	Géographie
M. CAPURON Alfred	Biologie
M. CARREZ Christian	I.E.E.A.
M. CAYATTE Jean-Louis	S.E.S.
M. CHAPOTON Alain	C.U.E.E.P.
M. COQUERY Jean-Marie	Biologie
Mme CORSIN Paule	Sciences de la Terre
M. CORTOIS Jean	Physique
M. COUTURIER Daniel	Chimie
M. CROSNIER Yves	I.E.E.A.
M. CURGY Jean-Jacques	Biologie
Mle DACHARRY Monique	Géographie
M. DAUCHET Max	I.E.E.A.
M. DEBRABANT Pierre	E.U.D.I.L.
M. DEGAUQUE Pierre	I.E.E.A.
M. DELORME Pierre	Biologie
M. DELORME Robert	S.E.S.
M. DE MASSON D'AUTUME Antoine	S.E.S.
M. DEMUNTER Paul	C.U.E.E.P.

PROFESSEURS 2ème classe (Suite 1)

M. DENEL Jacques	I.E.E.A.
M. DE PARIS Jean-Claude	Mathématiques (Calais)
Mle DESSAUX Odile	Chimie
M. DEVRAINNE Pierre	Chimie
M. DHAINAUT André	Biologie
Mme DHAINAUT Nicole	Biologie
M. DORMARD Serge	S.E.S.
M. DOUKHAN Jean-Claude	E.U.D.I.L.
M. DUBOIS Henri	Physique
M. DUBRULLE Alain	Physique (Calais)
M. DUBUS Jean-Paul	I.E.E.A.
M. FAKIR Sabah	Mathématiques
M. FONTAINE Hubert	Physique
M. FOUQUART Yves	Physique
M. FRONTIER Serge	Biologie
M. GAMBLIN André	G.A.S.
M. GLORIEUX Pierre	Physique
M. GOBLOT Rémi	Mathématiques
M. GOSSELIN Gabriel (dét.)	S.E.S.
M. GOUDMAND Pierre	Chimie
M. GREGORY Pierre	I.P.A.
M. GREMY Jean-Paul	S.E.S.
M. GREVET Patrice	S.E.S.
M. GUILBAULT Pierre	Biologie
M. HENRY Jean-Pierre	E.U.D.I.L.
M. HERMAN Maurice	Physique
M. JACOB Gérard	I.E.E.A.
M. JACOB Pierre	Mathématiques
M. JEAN Raymond	Biologie
M. JOFFRE Patrick	I.P.A.

PROFESSEURS 2ème classe (suite 2)

M. JOURNEL Gérard	E.U.D.I.L.
M. KREMBEL Jean	Biologie
M. LANGRAND Claude	Mathématiques
M. LATTEUX Michel	I.E.E.A.
Mme LECLERCQ Ginette	Chimie
M. LEFEVRE Christian	Sciences de la Terre
Mle LEGRAND Denise	Mathématiques
Mle LEGRAND Solange	Mathématiques (Calais)
Mme LEHMANN Josiane	Mathématiques
M. LEMAIRE Jean	Physique
M. LHENAFF René	Géographie
M. LOCQUENEUX Robert	Physique
M. LOSFELD Joseph	C.U.E.E.P.
M. LOUAGE Francis(dét.)	E.U.D.I.L.
M. MACKE Bruno	Physique
M. MAIZIERES Christian	I.E.E.A.
M. MESSELYN Jean	Physique
M. MESSERLIN Patrick	S.E.S.
M. MONTEL Marc	Physique
Mme MOUNIER Yvonne	Biologie
M. PARSY Fernand	Mathématiques
Mle PAUPARDIN Colette	Biologie
M. PERROT Pierre	Chimie
M. PERTUZON Emile	Biologie
M. PONSOLLE Louis	Chimie
M. PORCHET Maurice	Biologie
M. POVY Lucien	E.U.D.I.L.
M. RACZY Ladislas	I.E.E.A.
M. RAOULT Jean François	Sciences de la Terre
M. RICHARD Alain	Biologie

PROFESSEURS 2ème Classe (suite 3)

M. RIETSCH François	E.U.D.I.L.
M. ROBINET Jean-Claude	E.U.D.I.L.
M. ROGALSKI Marc	Mathématiques
M. ROY Jean-Claude	Biologie
M. SCHAMPS Joël	Physique
Mme SCHWARZBACH Yvette	Mathématiques
M. SLIWA Henri	Chimie
M. SOMME Jean	G.A.S.
Mle SPIK Geneviève	Biologie
M. STAROSWIECKI Marcel	E.U.D.I.L.
M. STERBOUL François	E.U.D.I.L.
M. TAILLIEZ Roger	Institut Agricole
Mme TJOTTA Jacqueline (dét.)	Mathématiques
M. TOULOTTE Jean-Marc	I.E.E.A.
M. TURRELL Georges	Chimie
M. VANDORPE Bernard	E.U.D.I.L.
M. VAST Pierre	Chimie
M. VERBERT André	Biologie
M. VERNET Philippe	Biologie
M. WALLART Francis	Chimie
M. WARTEL Michel	Chimie
M. WATERLOT Michel	Sciences de la Terre
Mme ZINN JUSTIN Nicole	Mathématiques

CHARGES DE COURS

M. ADAM Michel	S.E.S.

CHARGES DE CONFERENCES

M. BAFCOP Joël	I.P.A.
M. DUVEAU Jacques	S.E.S.
M. HOFLACK Jean	I.P.A.
M. LATOUCHE Serge	S.E.S.
M. MALAUSSENA DE PERNO Jean-Louis	S.E.S.
M. NAVARRE Christian	I.P.A.
M. OPIGEZ Philippe	S.E.S.

REMERCIEMENTS

Je tiens à remercier Monsieur Claude BRESINSKI, qui m'a fait l'honneur d'accepter la présidence du Jury de cette Thèse.

Je suis très reconnaissant envers Monsieur Gérard PLATEAU, qui m'a proposé ce travail, m'a permis de le mener à bien par ses nombreux conseils, et d'en avoir constamment suivi l'évolution.

Je suis très honoré de la présence de Monsieur Christian CARREZ, Professeur à LILLE I, et de Monsieur Michel MINOUX, Ingénieur responsable du Département de Mathématiques Appliquées pour les Télécommunications et l'Informatique du CNET, qui ont bien voulu s'intéresser à ce travail et le juger.

Que soient remerciés également Madame Bénédicte VANDROEMME ainsi que Monsieur Henri GLANC, qui avec gentillesse et rapidité ont participé à la réalisation matérielle de cette Thèse.

Je remercie très vivement, tous les amis qui m'ont aidés et soutenus durant l'élaboration de ce travail.

TABLE DES MATIERES

IV.5. ALGORITHMES DE ROTATION DE CONTRAINTES

INTRODUCTION

ANNEXE : PROBLEMES DETAILLES DU CHAPITRE II (Tableau 4 bis) ET DU CHAPITRE III (Tableau 6)

INTRODUCTION

Ce travail se situe dans le cadre de l'optimisation en nombres entiers dont les applications sont répandues dans de nombreux domaines économiques et techniques (problèmes d'investissement, de planification économique, d'affectation, de gestion de stocks, de découpe industriel, etc...).

Il conserne la résolution des problèmes de l'optimisation d'une fonction linéaire cx ($c \in \mathbb{R}^n$, $x \in \mathbb{N}^n$) soumise à des contraintes linéaires exprimées en égalité ou en inégalité de la forme $ax \leq (=) b$($a \in Z^n$, $x \in \mathbb{N}^n$, $b \in Z$), par les méthodes de programmation dynamique, d'énumération implicite ou les deux approches combinées.

Cet exposé est scindé en quatre chapitres : l'exposé des techniques et des algorithmes de programmation dynamique dans le cas du knapsack général (max cx s.c $ax \leq b$, $x \leq u$, $x \in \mathbb{N}^n$ où $u \in \mathbb{N}^n$) (Chapitre I) utilisés dans les autres chapitres est suivi de l'étude de deux problèmes particuliers :

* Problème du knapsack 0-1 à contrainte parallèle à la fonction économique : (S) max ax s.c $ax \leq b$, $x \in \{0, 1\}^n$ (Chapitre II).

* Problème du knapsack 0-1 (Chapitre III) à contrainte d'égalité ou d'inégalité : (B) max cx s.c $ax \leq (=) b$, $x \in \{0, 1\}^n$. Le dernier chapitre est consacré au problème de rotation de contraintes : étant donné un problème d'optimisation en nombres entiers à plusieurs contraintes ; Kianfar [40] a montré qu'il est possible d'obtenir une formulation équivalente avec un domaine épousant mieux l'enveloppe convexe des points entiers en faisant subir une rotation à une ou plusieurs contraintes (soit celles du problème initiale, soit des contraintes additionnelles) de sorte qu'elles viennent s'appuyer sur de nouveaux points entiers.

Chacun des chapitres est précédé d'une introduction précisant les idées abordées.

Le Chapitre I est consacré à l'étude théorique de la programmation dynamique pour le problème du knapsack général. De ce point de vue, tout ce qui sera développé dans ce chapitre peut être généralisé de manière naturelle au problème à plusieurs contraintes. Une formulation du problème du knapsack général est donnée dans le cas d'une contrainte d'égalité (le cas d'une contrainte d'inégalité est classique). Les différentes techniques et équations récursives de la programmation dynamique ainsi que les algorithmes correspondants sont répertoriés : il s'agit d'abord du problème du knapsack général où les variables sont bornées explicitement, ensuite celui des variables bornées implicitement et enfin le cas particulier du knapsack 0-1.

Les conclusions tirées de cette synthèse sont générales : les complexités temporelle et spatiale des algorithmes de programmation dynamique dépendent fondamentalement de la taille du problème (n) et du second membre de la contrainte (b). C'est essentiellement le paramètre b qui constituera la limitation de l'application de la programmation dynamique d'un point de vue encombrement mémoire.

Le Chapitre II est consacré au problème particulier du knapsack 0-1 : (S) max ax s.c ax $\leq$ b, x $\in \{0, 1\}^n$. Dans le cas où très peu (voir aucune) des solutions réalisables permettant d'atteindre b, les schémas d'énumération implicite sont pratiquement inutilisables même si b est relativement petit ; par contre les algorithmes de la programmation dynamique sont efficaces pour ce type de problèmes. Les algorithmes rencontrés dans la littérature ; résumés au début de ce chapitre ne semblent être efficaces que pour des problèmes de données bien particulières (c'est-à-dire, selon n, b et le nombre de solutions permettant d'atteindre b on applique tel ou tel algorithme). La proposition d'une méthode - appelée IDS - généralisant les méthodes existantes dans le sens suivant : si un problème est résoluble par l'une des méthodes existantes alors il est résoluble par IDS avec des temps comparables (les résultats numériques donnés en fin du chapitre le prouvent) ; la réciproque étant fausse ; constitue la première contribution de cette thèse.

Le Chapitre III est consacré au problème du knapsack 0-1 général. Les études faites sur ce problème conduisent aux conclusions suivantes :

- Il y a des problèmes qui ne peuvent être résolus que par les algorithmes d'énumération implicite, d'autres uniquement par la programmation dynamique.

- La performance des algorithmes d'énumération implicite est liée à la nature des données : plus les données sont correlées plus le problème est difficile.

- Les algorithmes de la programmation dynamique sont moins affectés par cette correlation ; mais leur performance dépend de la valeur du second membre b de la contrainte.

La proposition d'une méthode générale et efficace - notée ID - combinant l'énumération implicite et la programmation dynamique constitue la deuxième contribution de cette thèse : elle permet de résoudre une vaste classe de problèmes bien plus grande que celle résoluble par les deux approches considérées séparément. Cette classe inclue le problème du knapsack 0-1 avec une contrainte d'égalité ; celui-ci est transformé en un problème ayant les mêmes solutions optimales avec une contrainte d'inégalité. La fin de ce chapitre regroupe les résultats expérimentaux des problèmes testés répertoriés des plus faciles aux plus difficiles.

Le Chapitre IV est consacré à la rotation de contraintes. Après un rappel de la formulation du problème et de son interprétation géométrique, nous introduisons le problème de la rotation d'une contrainte sous une contrainte additionnelle qui permet de serrer davantage le domaine. Cette idée a soulevé les problèmes suivants discutés dans ce chapitre : critère du choix de la contrainte additionnelle et celui de la contrainte éliminant une contrainte donnée. En plus des applications classiques de la rotation de contraintes (dans les méthodes algébriques, d'obtention de formulations équivalentes) nous montrerons comment les tests d'élimination de contraintes peuvent être améliorés.

Jusqu'à lors la rotation d'une contrainte de n variables et de second membre b nécessite une complexité temporelle $O(n^2b)$; la donnée d'un algorithme de complexité temporelle $O(nb)$ constitue la troisième contribution de cette thèse.

NOTATIONS

Les notations utilisées dans cet exposé sont :

$\lfloor\lambda\rfloor$: partie entière d'un réel λ ; $\lceil\lambda\rceil = -\lfloor-\lambda\rfloor$

étant donné un ensemble J :

$|J|$: cardinal de J

$J\backslash U$: complémentaire d'un sous-ensemble U de J

étant donnée A une matrice de format $(I \times J)$ où I et J sont des ensembles finis :

a_{ij} : élément $(i, j) \in I \times J$ de A

A_i : ligne $i \in I$ de A

étant donné y un vecteur (ligne ou colonne) indicé par un ensemble J :

y_j : élément $j \in J$ de y

$y_{J'}$: sous-vecteur de y composé des éléments y_j $j \in J' \subset J$

étant donné un problème d'optimisation (P) à n variables :

v(P) : valeur optimale de (P)

$\bar{v}(P)$: (resp. $\underline{v}(P)$) borne supérieure (resp. inférieure) de v(P)

F(P) : domaine de (P)

$(P \mid x_j = \varepsilon)$: lorsque (P) est résolu avec la condition supplémentaire

$x_j = \varepsilon \quad j \in \{1, \ldots, n\}$ et $\varepsilon \in \{0, 1\}$

x^* : solution optimale de (P)

$V = \{x \in \mathbb{R}^n \mid x_j = 0 \text{ ou } 1,\ j = 1, \ldots, n\}$

$(\bar{P})$: problème (P) dans lequel V est remplacé par son enveloppe convexe [V]

$\bar{x}$: solution optimale de $(\bar{P})$.

CHAPITRE 1

LA PROGRAMMATION DYNAMIQUE ET LE PROBLEME DU KNAPSACK

INTRODUCTION

La programmation dynamique est une méthode de résolution de problèmes d'optimisation. Il est rappelé que le terme "programmation" ne doit pas être compris dans le sens qu'on lui donne en informatique (écrire un algorithme dans un langage informatique) : il signifiera ici résolution de problème, mis sous une forme appelée "programme mathématique", qui à chaque décision possible est associée la valeur d'une variable. Les contraintes physiques ou économiques qui régissent le système étudié sont traduites en contraintes algébriques sur les variables de décision. On cherche une solution optimale vis-à-vis d'un critère appelé fonction économique.

Lorsque les contraintes et la fonction économique sont linéaires, on parle de programmation mathématique linéaire. Cette hypothèse de linéarité n'est aucunement nécessaire pour pouvoir appliquer la programmation dynamique. De nombreux problèmes non linéaires sont résolus par la programmation dynamique (par exemple : problèmes variationnels par Bellman [7, 8] et Dantzing [15] ; problèmes du knapsack non linéaires par Morin et Marsten [56]).

La programmation dynamique (où le terme "dynamique" signifie une résolution séquentielle de sous-problèmes qui s'obtiennent récursivement) peut apporter une solution effectivement calculable aux problèmes : de décisions séquentielles ; discrets ou continus ; aléatoires ou déterministes ; finis ou infinis (voir les ouvrages de Laurière [46] et Minoux [54]).

Ce chapitre est consacré à l'étude théorique de la programmation dynamique pour le problème du knapsack général ; souvent utilisé comme base de résolution de problèmes d'optimisation plus difficiles (voir la Thèse de Fayard et Plateau [18]). Du point de vue théorique, tout ce qui sera développé dans ce chapitre est généralisable de manière naturelle aux problèmes de plusieurs contraintes (Balinski [4], Shapiro et Wagner [67], Weingartner et Ness [72], Nemhauser et Ullman [59], Minoux [54], Shapiro [68], Wolsey [75]).

Une formulation du problème du knapsack général (Section I.1.1.) avec une contrainte d'égalité (le cas d'une contrainte d'inégalité est classique) précède une synthèse des différentes techniques et d'équations récursives de la programmation dynamique ainsi que les algorithmes correspondants.

Cette synthèse est présentée selon les trois cas particuliers du problème du knapsack général :

le problème du knapsack à variables bornées,

* explicitement (cas général) (Section I.1.2.)
* implicitement (un cas particulier du précédent ; (Section I.1.3.)
* par 1 (le cas du knapsack 0-1 ; Section I.2.) □

I.1. LE PROBLEME DU KNAPSACK GENERAL

I.1.1. Formulation

Les données positives du problème nécessaires aux équations dynamiques, nous ont amenées à donner les deux formulations :

a) Celle du problème du knapsack à contrainte d'égalité.

b) Celle du problème du knapsack à contrainte d'inégalité.

La première est plus difficile à réaliser que la deuxième.

a) Knapsack à contrainte d'égalité

Nous allons discuter dans quels cas, les données du problème

$$(P)\quad \begin{cases} \max \sum_{j=1}^{n} c_j x_j \\ \text{s.c} \sum_{j=1}^{n} a_j x_j = b \\ x_j \in \mathbb{N} \quad j = 1, \ldots, n \end{cases}$$

(où c_j, $a_j \in Z$ $j \in \{1, \ldots, n\}$ et $b \in \mathbb{N}$) sont des entiers positifs.

- Cas 1 : $a_j > 0$, $j = 1,\ldots, n$

En faisant une transformation adéquate du problème (P), on peut obtenir un problème équivalent avec tous les c_j ($j = 1,\ldots, n$) positifs.

Posons :

$$\alpha_j = - a_j \left\lfloor \min_{i=1,\ldots,n} (c_i/a_i) \right\rfloor \qquad j = 1,\ldots, n$$

$$c'_j = c_j + \alpha_j \qquad j = 1,\ldots, n$$

les c'_j ainsi définis sont des entiers positifs.

Le problème (P) est équivalent au problème (P') suivant :

$$(P') \begin{cases} \max \sum_{j=1}^{n} c'_j x_j \\ \text{s.c} \sum_{j=1}^{n} a_j x_j = b \\ x_j \in \mathbb{N} \quad j = 1,\ldots, n \end{cases}$$

puisque $\forall x \in F(P') = F(P)$

$$\sum_{j=1}^{n} c'_j x'_j = \sum_{j=1}^{n} c_j x_j + \sum_{j=1}^{n} \alpha_j x_j = \sum_{j=1}^{n} c_j x_j - \left\lfloor \min_{i=1,\ldots,n} (c_i/a_i) \right\rfloor \sum_{j=1}^{n} a_j x_j$$

$$= \sum_{j=1}^{n} c_j x_j - \left\lfloor \min_{i=1,\ldots,n} (c_i/a_i) \right\rfloor b \qquad \square$$

- Cas 2 : $c_j \geq 0$ $j = 1,\ldots, n$ et $\exists i \in \{1,\ldots, n\} : a_i \leq 0$

Ecrivons le problème (P) sous la forme suivante :

$$(P) \qquad \max_{x_i \in \mathbb{N}} \left\{ c_i x_i + \left[\begin{array}{l} \max \sum_{\substack{j=1 \\ j \neq i}}^{n} c_j x_j \\ \\ \text{s.c} \sum_{\substack{j=1 \\ j \neq i}}^{n} a_j x_j = b - a_i x_i \\ \\ x_j \in \mathbb{N} \ \forall j \neq i \end{array} \right] \right\}$$

Puisque a_i est négatif, la quantité $b - a_i x_i$ est positive et le problème (P) peut avoir une valeur infinie ou un domaine vide.

- Cas 3 : $\exists i \in \{1, \ldots, n\} : c_i \leq 0$ et $a_i \leq 0$

Dans ce cas le problème (P) peut avoir (suivant les exemples) une valeur finie ou infinie. Dans les deux exemples qui suivent, le premier illustre le cas où (P) a une valeur finie et le deuxième le cas où (P) a une valeur infinie.

Exemple 1 :

Etant donné le problème

$$\left[\begin{array}{l} \max -x_1 + x_2 \\ \text{s.c} -x_1 + 2x_2 = 2 \\ \quad x_1, x_2 \in \mathbb{N} \end{array} \right.$$

est équivalent à

$$\left[\begin{array}{l} \max x_2 \\ \text{s.c} \ 2x_2 = 2 + x_1 \\ \quad x_1, x_2 \in \mathbb{N} \end{array} \right.$$

dont toutes les solutions réalisables sont de la forme $x = \begin{pmatrix} 2\alpha \\ 1+\alpha \end{pmatrix}$ avec $\alpha \in \mathbb{N}$ et sa valeur optimale est infinie.

Remarque

Si le problème (P) est borné ; c'est-à-dire :

$$0 \leq x_j \leq u_j,\ j = 1,\ldots, n$$

(où u_j sont des entiers positifs).

Alors on peut toujours se ramener à un problème où tous les coefficients c_j et a_j sont des entiers positifs.

Pour cela, il suffit, dans le cas où il existe un i tel que ($a_i < 0$), de poser le changement de variable suivant :

$$x_i = u_i - x'_i$$

on se trouve alors dans les conditions du cas 1.

b) Knapsack à contrainte d'inégalité

Le problème du knapsack à contrainte d'inégalité lorsque sa valeur n'est pas infinie peut être formulé aisément (voir Fayard et Plateau [18]) :

$$(B)\quad \begin{cases} \max \sum_{j=1}^{n} c_j x_j \\ \text{s.c} \sum_{j=1}^{n} a_j x_j \leq b \\ 0 \leq x_j \leq u_j,\ j = 1,\ldots, n \\ x_j \in \mathbb{N},\ j = 1,\ldots, n \end{cases}$$

avec les hypothèses

$$(H)\quad \begin{cases} c_j,\ a_j \in \mathbb{N}^* \ (j = 1,\ldots, n) \text{ et } b \in \mathbb{N}^* \\ \max a_j \leq b \\ 1 \leq j \leq n \\ b < \sum_{j=1}^{n} a_j u_j \end{cases}$$

Dans la suite, les hypothèses (H) sont supposées être vérifiées. □

I.1.2. Le knapsack a variables bornées explicitement

La mise en équation dynamique du problème (B) nécessite

- en premier lieu de décomposer le problème en une suite de sous-problèmes de même nature,

- ensuite de relier les unes aux autres, par une équation récursive, les solutions optimales de ces sous-problèmes.

Pour un couple (k, y) (k = 1,..., n, y = 0, 1,..., b) considérons le sous-problème suivant :

$$(B_k(y)) \quad \begin{cases} \max \sum_{j=1}^{k} c_j x_j \\ \text{s.c} \sum_{j=1}^{k} a_j x_j \leq y \\ 0 \leq x_j \leq u_j \; ; \; x_j \in \mathbb{N} \quad j = 1,\ldots, k \end{cases}$$

(où $u_j \in \mathbb{N}^*$ $j = 1,\ldots, k$).

En définissant : $f_k(y) = v(B_k(y))$ pour $k \in \{1, 2,\ldots, n\}$ et $y \in \{0, 1,\ldots, b\}$ la valeur optimale de (B) est à l'évidence $f_n(b)$.

Dans le but de relier f_k et f_{k-1} par une équation récursive - appelée équation dynamique - , isolons $c_k x_k$ dans $(B_k(y))$:

$$\max_{0\le x_k\le \min(u_k,\lfloor y/a_k\rfloor)} \left\{ c_k\, x_k + \left[\max \sum_{j=1}^{k-1} c_j\, x_j \right.\right.$$

$$(B_k(y)) \qquad \left.\left. \text{s.c} \sum_{j=1}^{k-1} a_j\, x_j \le y - a_k\, x_k \right.\right.$$

$$\left.\left. 0 \le x_j \le u_j,\ x_j \in \mathbb{N}\ j = 1,\ldots,k-1 \right] \right\}$$

on déduit :

$$(1.1)\quad \begin{cases} f_k(y) = \max\{c_k\, x_k + f_{k-1}(y - a_k\, x_k)\} \\ \qquad x_k = 0,\ldots,\ \min(u_k,\ \lfloor y/a_k\rfloor) \\ k = 1,\ldots,\ n \text{ et } y = 0,\ 1,\ldots,\ b \end{cases}$$

sous l'hypothèse :

$$f_0(y) = 0 \qquad y = 0,\ 1,\ldots,\ b$$

L'algorithme dû à l'équation récursive (1.1) est à l'évidence de complexité temporelle $O(b \sum_{k=1}^{n} u_k)$ et spatiale $O(b)$.

Algorithme A1

But : Détermination de la valeur de (B) : $v(B) = f_n(b)$.

0 pour y de 0 à b faire $f_0(y) \leftarrow 0$ fait

1 pour k de 1 à n faire
  pour y de 0 à b faire
    $f_k(y) \leftarrow \max_{x_k=0,\ldots,\min(u_k,\lfloor y/a_k\rfloor)} (c_k\, x_k + f_{k-1}(y - a_k\, x_k))$
  fait
fait

Notes :

(i) Dans le cas où la contrainte du problème (B) est exprimée en égalité

(i.e. $\sum_{j=1}^{n} a_j x_j = b$), l'algorithme A1 reste valable en remplaçant l'étape 0 par 0'.

0'	$f_0(0) = 0$ pour y de 1 à b faire $f_0(y) \leftarrow -\infty$ fait

(ii) L'algorithme A1 est trivial ; il est moins performant que les algorithmes sophistiqués qui vont suivre, mais l'intérêt de sa présentation réside dans son aspect général qui le rend applicable aux problèmes non linéaires.

I.1.3. Equations relatives aux problèmes a variables non explicitement bornées

I.1.3.1. Equation due à Gilmore et Gomory : 1965 [23]

Dans le problème $(B_k(y))$ la variable x_k peut être nulle ou différente de zéro dans une solution optimale.

- si $x_k = 0$ alors $f_k(y) = f_{k-1}(y)$ (1.2)

- si $x_k > 0$ alors $f_k(y) = c_k + f_k(y - a_k)$ (1.3)

En combinant (1.2) et (1.3), on déduit l'équation récursive

$$f_k(y) = \begin{cases} f_{k-1}(y) & y = 0, \ldots, a_{k-1} \\ \max\{f_{k-1}(y),\ c_k + f_k(y - a_k)\} & y = a_k, \ldots, y \end{cases} \qquad (1.4)$$

sous l'hypothèse

$$f_0(y) = 0 \qquad y = 0, 1, \ldots, b$$

L'algorithme dû à l'équation récursive (1.4) est à l'évidence de complexité temporelle O(nb) et spatiale O(b).

Algorithme A2

But : Détermination de la valeur de (B) : $v(B) = f_n(b)$.

0 pour y de 0 à b faire $f_0(y) \leftarrow 0$ fait

1 pour k de 1 à n faire
  pour y de 0 à a_{k-1} faire
    $f_k(y) \leftarrow f_{k-1}(y)$
  fait
  pour y de a_k à b faire
    $f_k(y) \leftarrow \max\{f_{k-1}(y),\ c_k + f_k(y - a_k)\}$
  fait
fait

Notes :

(i) L'équation récursive (1.4) a été formulée par Dantzing en 1957 [15], dans le cas bivalent (i.e. $x_j = 0$ ou 1 $j = 1, \ldots, n$).

(ii) Dans le cas où la contrainte du problème (B) est exprimée en égalité, l'algorithme A2 reste valable en remplaçant l'étape 0 par 0'.

0' $f_0(0) = 0$
pour y de 1 à b faire $f_0(y) \leftarrow -\infty$
fait

(iii) La compléxité spatiale de l'algorithme A2 pour déterminer une solution optimale x^* du problème (B) est O(b) en utilisant l'indicateur I défini par :

$$I_0(y) = 0 \qquad y = 0,1,\ldots, b$$

$$I_k(y) = \begin{cases} I_{k-1}(y) & \text{si } f_k(y) = f_{k-1}(y) \\ k & \text{si } f_k(y) = c_k + f_k(y - a_k) \end{cases}$$

$$k = 1,\ldots, n \qquad y = 0, 1,\ldots, n$$

Le dernier vecteur I_n suffit pour retrouver une solution optimale.

(iii) On peut trouver l'algorithme A2 détaillé en [21].

Détermination de x^*

0 $y \leftarrow b$
pour j de 1 à n faire $x_j^* \leftarrow 0$ fait

1 Tant que $f_n(y) > 0$ faire

$$x^*_{I_n(y)} \leftarrow x^*_{I_n(y)} + 1 \ ;$$

$$y \leftarrow y - a_{I_n(y)} \ ;$$

fait

□

I.1.3.2. Equation due à Gilmore et Gomory : 1966 [24]

Considérons les sous-problèmes de (B)

$$(B(y)) \begin{cases} \max \sum_{j=1}^{n} c_j x_j \\ \text{s.c} \sum_{j=1}^{n} a_j x_j \leq y \\ x_j \in \mathbb{N} \qquad j = 1,\ldots, n \end{cases}$$

En définissant par $f(y) = v(B(y))$ $(y = 0, 1,\ldots, b)$, la valeur de (B) est $f(b)$.

Considérons pour $y \in \{0, 1, \ldots, b\}$ donné

$$S(y) = \{j \in \{1, \ldots, n\} \mid a_j \leq y\}$$

Il est clair que la valeur $f(y)$ de $(B(y))$ pour $y \in \{0, 1, \ldots, b\}$ se formule ainsi :

$$f(y) = \begin{cases} \max\{c_j + f(y - a_j) \mid j \in S(y)\} & \text{si } S(y) \neq \emptyset \\ 0 & \text{sinon} \end{cases}$$

Cette formulation reste valable, si on maximise sur un sous-ensemble de $S(y)$, à condition qu'il ait une intersection non vide avec l'ensemble D suivant,

$y \in \{0, 1, \ldots, b\}$ donné

$$D(y) = \{k \in \{1, \ldots, n\} \mid f(y) = c_k + f(y - a_k)\}$$

($k \in D(y)$ signifie qu'il existe une solution optimale de $(B(y))$ telle que la $k^{\text{ième}}$ composante est strictement positive) ce qui se traduit par

$$f(y) = \begin{cases} \max\ c_j + f(y - a_j) \mid j \in Q(y)\} & \text{si } Q(y) \neq \emptyset \\ 0 & \text{sinon} \end{cases}$$

où $Q(y) \subset S(y)$ et $Q(y) \cap D(y) \neq \emptyset$

Théorème 1

$\forall y \in \{0, \ldots, b\}$

En définissant par

$$d(y) = \begin{cases} \min\{j \in \{1, \ldots, n\} \mid j \in D(y)\} & \text{si } D(y) \neq \emptyset \\ n & \text{sinon} \end{cases}$$

$$Q(y) = \{j \in S(y) \mid j \leq d(y - a_j)\}$$

on a :

$$f(y) = \max\{(c_j + f(y - a_j),\ 0) \mid j \in Q(y)\}.$$

Démonstration

Puisque $Q(y) \subset S(y)$, il suffit de montrer que

$$Q(y) \cap D(y) \neq \emptyset.$$

Pour cela il suffit de remarquer que l'élément d(y) de D(y) appartient également à Q(y), puisque $\forall j \in D(y)$ on a

$D(y - a_j) \subset D(y)$; d'où $d(y) \leq d(y - a_j)$. □

Remarque : L'algorithme qui calcule f(b) (la valeur de (B)) est d'autant plus efficace que d(y) est petit ($y \in \{0, 1, \ldots, b\}$). C'est la raison pour laquelle les données sont supposées par la suite : $c_1/a_1 \geq c_2/a_2 \geq \ldots \geq c_n/a_n$.

Algorithme A3

But : Détermination de la valeur f(b) et de la solution optimale x^*.

```
0   pour y de 0 à b faire
                f(y) ← 0 ; d(y) ← n ;
    fait

1   pour y de min   a_i à b faire
           1≤i≤n
        pour j de 1 à n faire
          si a_j ≤ y et j ≤ d(y - a_j) alors
             v ← c_j + f(y - a_j) ;
             si v > f(y) alors ;
                f(y) ← v ; d(y) ← j ;
             fsi
          fsi
        fait
    fait
```

```
2   % Calcul de la solution optimale x* %
    y ← b
    pour j de 1 à n faire
        x*_j ← 0 ;
    fait
    Tant que f(y) > 0 faire
            x*_d(y) ← x*_d(y) + 1 ; y ← y - a_d(y) ;
    fait
```

Note :

L'algorithme A3 nécessite deux fois moins de places mémoire que l'algorithme A2. Deux listes seulement (de taille (b+1) chacune) sont utilisées ; Une pour $\{f(y)\}_{y=0}^{b}$ et l'autre pour $\{d(y)\}_{y=0}^{b}$.

L'exemple suivant illustre le déroulement de l'algorithme A3.

Exemple :

$$\begin{cases} \max\ 3x_1 + 4x_2 \\ \text{s.c}\ 2x_1 + 3x_2 \leq 9 \\ \quad x_1,\ x_2 \in \mathbb{N} \end{cases}$$

Les premières colonnes respectives de f(y) et d(y) représentent l'initialisation. La valeur optimale est f(y) = 13. La solution optimale est ($x_1^* = 3$, $x_2^* = 1$).

y	f(y)		d(y)	
0	0	0	2	2
1	0	0	2	2
2	0	3	2	1
3	0	4	2	2*
4	0	6	2	1
5	0	7	2	1*
6	0	9	2	1
7	0	10	2	1*
8	0	12	2	1
9	0	13	2	1*

Figure 1

Remarque : Dans cet exemple, on constate que d(y) = 1 pour y ≥ 4. Ceci est la conséquence de l'existence du phénomène de périodicité, que nous allons aborder dans le paragraphe suivant.

I.1.3.3. Périodicité et propriétés

I.1.3.3.1. Résultats préliminaires

L'hypothèse suivante : $c_1/a_1 \geq c_2/a_2 \geq \ldots \geq c_n/a_n$ est nécessaire pour la suite de l'exposé. Etant donnés $m \in \mathbb{N}$, $r \in \mathbb{N}$: $1 \leq m \leq r \leq n$.

Considérons les sous-problèmes suivants pour $y = 0, \ldots, b$.

$$(B_{m,r}(y)) \begin{cases} \max \sum_{j=m}^{r} c_j x_j \\ \text{s.c} \sum_{j=m}^{r} a_j x_j \leq y \\ x_j \in \mathbb{N} \quad j = m, \ldots, r \end{cases}$$

En définissant $f_m^r(y) = v(B_{m,r}(y))$ pour $y \in \{0, 1, \ldots, b\}$, il est évident que : $f(b) = f_1^n(b)$.

Considérons les coefficients $\alpha_i, \beta_i \in \mathbb{N}^*$ vérifiant

$$a_i \alpha_i = a_m \beta_i, \quad i = m+1, \ldots, n \tag{1.5}$$

posons : $y_0 = a_{m+1} \alpha_{m+1} + \ldots + a_n \alpha_n$ (1.6)

Théorème 2 : Gilmore et Gomory [24]

Etant donné $y \geq y_0$, *il existe* $y_1 \in \{0, 1, \ldots, y_0\}$

tel que :

$$f(y) = f_1^m(y - y_1) + f_{m+1}^n(y_1)$$

$\forall m \in \{1, 2, \ldots, n-1\}$.

Démonstration

Etant donné $y \geq y_0$

Soit y_1 la plus petite valeur telle que :

(1.7) $y_1 = \sum_{j=m+1}^{n} a_j x_j^*$ (Lorsque x^* décrit l'ensemble des solutions optimales).

(1.8) $f(y) = f_1^m(y - y_1) + f_{m+1}^n(y_1)$

Lorsque $y_1 > y_0$ (sinon le résultat est trivial)

i.e. $\sum_{j=m+1}^{n} a_j x_j^* > \sum_{j=m+1}^{n} a_j \alpha_j = y_0$ (y_0 étant définie en (1.6)),

il existe $i \in \{m+1, \ldots, n\}$ tel que : $x_i^* > \alpha_i$ ($\alpha_i \in \mathbb{N}^*$)

En posant $y_1' = y_1 - a_i \alpha_i$

il est clair que

$$0 \leq y_1' < y_1$$

Nous allons montrer que y_1' vérifie (1.8),

i.e. $f(y) = f_1^m(y - y_1') + f_{m+1}^n(y_1')$

ce qui est contradictoire avec la définition de y_1 :

$y_1' = y_1 - a_m \beta_i$ puisque $a_i \alpha_i = a_m \beta_i$ d'après (1.5)

d'où

$$\left.\begin{array}{l} f_1^m(y - y_1') = f_1^m(y - y_1 + a_m \beta_i) \geq f_1^m(y - y_1) + f_1^m(a_m \beta_i) \\ \text{or } f_1^m(a_m \beta_i) \geq c_m \beta_i \end{array}\right\} \Rightarrow$$

$$\ldots f_1^m(y - y_1') + f_{m+1}^n(y_1') \geq f_1^m(y - y_1) + c_m \beta_i + f_{m+1}^n(y_1') \qquad (1.9)$$

$$\left.\begin{array}{l} \dfrac{c_m}{a_m} \geq \dfrac{c_i}{a_i} \Rightarrow \dfrac{c_m\ \beta_i}{a_m\ \beta_i} \geq \dfrac{c_i\ \alpha_i}{a_i\ \alpha_i} \\ (1.5) \Rightarrow a_m\ \beta_i = a_i\ \alpha_i \end{array}\right\} \Rightarrow c_m\ \beta_i \geq c_i\ \alpha_i \qquad (1.10)$$

$$\left.\begin{array}{l} (1.9) \text{ et } (1.10) \Rightarrow f_1^m(y - y_1') + f_{m+1}^n(y_1') \geq f_1^m(y - y_1) + c_i\ \alpha_i + f_{m+1}^n(y_1 - a_i\ \alpha_i) \\ (1.8) \text{ et } x_i^* > \alpha_i > 0 \quad i \in \{m+1, \ldots, n\} \Rightarrow c_i\ \alpha_i + f_{m+1}^n(y_1 - a_i\ \alpha_i) = f_{m+1}^n(y_1) \end{array}\right\} \Rightarrow$$

$$\ldots f_1^m(y - y_1') + f_{m+1}^n(y_1') \geq f_1^m(y - y_1) + f_{m+1}^n(y_1) = f(y) \quad \text{(d'après (1.8))}$$

$$\text{or } f(y) \geq f_1^m(y - y_1') + f_{m+1}^n(y_1')$$

$$\text{d'où } f(y) = f_1^m(y - y_1') + f_{m+1}^n(y_1'). \qquad \square$$

Corollaire 1 : *Gilmore et Gomory* [24]

Si $m = 1$, *alors pour tout* $b \geq y_0$, *Il existe* $y_1 \leq y_0$:

$$f(b) = \left\lfloor (b - y_1)/a_1 \right\rfloor c_1 + f_2^n(y_1).$$

I.1.3.3.2. Périodicité

D'après le Corollaire 1, si b est suffisamment grand (par exemple $b \geq y_0 + a_1$) alors $\left\lfloor (b - y_1)/a_1 \right\rfloor \geq 1$, c'est-à-dire $x_1^* \geq 1$. f(b) peut s'écrire :

$$f(b) = f(b - a_1) + c_1$$

Considérons, la fonction ψ suivante :

$$\forall b \geq y_0 + a_1 : \psi(b) = f(b) - \left\lfloor b/a_1 \right\rfloor c_1 \qquad (1.11)$$

Le théorème suivant va permettre de conclure que la fonction ψ est périodique de période a_1.

Théorème 3 : [24]

$\forall b \geq y_0 + a_1$,

La fonction ψ définie en (1.11) est périodique de période a_1.

Démonstration

D'après ce qui précède, si $b \geq y_0 + a_1$ alors $f(b) = f(b - a_1) + c_1$

d'où : $\psi(b) = f(b - a_1) + c_1 - \lfloor b/a_1 \rfloor c_1$

$= f(b - a_1) - (\lfloor b/a_1 \rfloor - 1) c_1$

or $\lfloor b/a_1 \rfloor - 1 = \lfloor (b - a_1)/a_1 \rfloor$

d'où $\psi(b) = \psi(b - a_1)$.

Corollaire 2

Si $b = ka_1 + y$, $k \in \mathbb{N}$ *et* $y \geq y_0$

alors :

$\psi(b) = \psi(y)$.

Cette étude permet de donner la conclusion suivante :

Etant donné b (suffisamment grand), il existerait une valeur y^* (plus petite que b) pour laquelle il sera inutile de calculer

$f(y) \qquad y = y^* + 1, \ldots, b$;

pour calculer f(b), il suffirait de calculer

$f(y) \qquad y = 0, \ldots, y^*$

et de prendre

$$f(b) = \left\lfloor (b - y^*)/a_1 \right\rfloor c_1 + f(y^*)$$

y_0 (1.6) donnée par Gilmore et Gomory [24] étant une borne inférieure de y^*.

Le paragraphe suivant est consacré à d'autres bornes inférieures de y^*.

I.1.3.3.3. Conditions suffisantes

Le but est de donner une condition suffisante sur y^* tel que

$\forall y \in \mathbb{N}$

si $y \geq y^*$ alors $f(y) = c_1 + f(y - a_1)$.

Théorème 4 : HU 1966 [32]

Si $c_1/a_1 > c_2/a_2$ *alors pour tout* $y \geq y_h = \left\lceil \frac{c_1}{\frac{c_1}{a_1} - \frac{c_2}{a_2}} \right\rceil$

On a :

$$f(y) = c_1 + f(y - a_1).$$

Démonstration

Etant donnés, un minorant de $(B(y))$ $(\underline{v}(B(y)) = c_1 \left\lfloor \frac{y}{a_1} \right\rfloor)$ et un majorant de $(B(y) \mid x_1 = 0)$ $(\overline{v(B(y) \mid x_1 = 0)} = c_2 \frac{y}{a_2})$ une condition suffisante pour que $x_1^* \geq 1$ est :

$$\overline{v(B(y) \mid x_1 = 0)} \leq \underline{v}(B(y))$$

Montrons que sous les hypothèses du théorème, cette dernière est vérifiée :

$$\left.\begin{array}{l} c_1/a_1 > c_2/a_2 \\ y \geq \left\lceil \dfrac{c_1}{\dfrac{c_1}{a_1} - \dfrac{c_2}{a_2}} \right\rceil \end{array}\right\} \Rightarrow y \geq \frac{c_1}{\frac{c_1}{a_1} - \frac{c_2}{a_2}}$$

$$\Rightarrow y \frac{c_1}{a_1} - y \frac{c_2}{a_2} \geq c_1 \Rightarrow \ldots$$

$$\left.\begin{array}{l} \Rightarrow c_1 \dfrac{y}{a_1} - c_1 \geq c_2 \dfrac{y}{a_2} \\ \text{or } c_1 \dfrac{y}{a_1} - c_1 \leq c_1 \left\lfloor \dfrac{y}{a_1} \right\rfloor \end{array}\right\} \Rightarrow c_1 \left\lfloor \frac{y}{a_1} \right\rfloor \geq c_2 \frac{y}{a_2} . \qquad \square$$

Dans le cas où $c_1/a_1 = c_2/a_2$, une autre condition suffisante est donnée par le théorème suivant.

Théorème 5 : *Salkin* [66]

Sous l'hypothèse $c_1/a_1 = c_2/a_2$, *si* $y > y_s = a_{max}(a_1 - 1)$ *avec*

$a_{max} = \max_{2 \leq j \leq n} (a_j)$ *alors*

$$f(y) = c_1 + f(y - a_1)$$

Démonstration

Soit x_e la variable d'écart : $\sum_{j=1}^{n} a_j x_j + x_e = y$ l'indice de base est $i = 1$ et on a :

$$x_1 = \frac{1}{a_1} (y - \sum_{j=2}^{n} a_j x_j - x_e) \geq 0 \text{ et } x_1 \text{ entier.}$$

$$x_1 = \frac{1}{a_1} (y - \sum_{j=2}^{n} a_j x_j - x_e) \geq \frac{1}{a_1} (y - a_{max}(\sum_{j=2}^{n} x_j + x_e)) \qquad (i)$$

or pour toute solution entière obtenue à partir d'une solution de base, les variables hors base doivent vérifier :

$$\sum_{j=2}^{n} x_j + x_e \leq a_1 - 1 \qquad \text{(voir [66])}$$

(i) devient :

$$x_1 \geq \frac{1}{a_1}(y - a_{max}(a_1 - 1))$$

Pour que $x_1 > 0$ il suffit que

$$y > a_{max}(a_1 - 1)$$

d'où $y > a_{max}(a_1 - 1) \Rightarrow x_1 \geq 1 \Rightarrow f(y) = c_1 + f(y - a_1)$ □

Remarque : Les bornes inférieures y_0, y_h et y_s sont souvent éloignées de la valeur théorique de y^*.

I.1.3.3.4. Application de la périodicité dans l'algorithme A3

Le test déterminant le seuil y^* dans l'algorithme A3 peut être réalisé comme suit :

soient $a_k = \max_{j=1,\ldots,n} (a_j)$ et y' appartenant à $\mathbb{N}$.

Supposons avoir calculé f(y) (par l'algorithme A3) pour y = 0, 1,..., y' et qu'à partir de y', on constate que d(y) = 1 dans l'intervalle $[y', y' + a_k[$, il est inutile de continuer l'exécution de l'algorithme puisque, dans ce cas, d(y) = 1 dans tout l'intervalle [y', b]. En effet, si d(y) = 1 dans l'intervalle $[y', y' + a_k[$ alors $d(y' + a_k - a_j) = 1$ pour tout j ; puisque $f(y) = \max_{1 \leq j \leq n} c_j + f(y - a_j)$ on déduit que $d(y' + a_k) = 1$. Ainsi, de proche en proche, on déduit que d(y) = 1 pour tout y appartenant à [y', b].

Si en plus $d(y' - 1) > 1$ alors on peut conclure que le seul théorique y^* est exactement y'.

Dans l'exemple précédent (Figure 1) $y_0 = 6$, $y_h = 18$ et le seuil théorique était $y^* = 4$. L'algorithme A3 peut s'arrêter à $y' = 4 + 3 - 1 = 6$.

Pour plus de signification, considérons un autre exemple :

$$\begin{cases} \max\ 11x_1 + 7x_2 + 5x_3 + x_4 \\ \text{s.c}\ 6x_1 + 4x_2 + 3x_3 + x_4 \leq 25 \\ \qquad x_1, x_2, x_3, x_4 \in \mathbb{N} \end{cases}$$

Dans cet exemple $y_0 = 24$ et $y_h = 132$ alors que l'algorithme A3 s'achève dès que $y' = 14$, c'est-à-dire $y^* = 14 - 6 + 1 = 9$.

I.2. PROBLEME DU KNAPSACK EN VARIABLES 0-1

I.2.1. Formulation

Le problème du knapsack en variable 0-1 - appelé aussi problème du knapsack 0-1 ou problème du knapsack bivalent - est un cas particulier du problème (B) formulé dans le paragraphe I.1.1.b) et s'écrit :

$$(B) \quad \begin{cases} \max \sum_{j=1}^{n} c_j x_j \\ \text{s.c} \sum_{j=1}^{n} a_j x_j \leq b \\ \qquad x_j = 0 \text{ ou } 1 \qquad j = 1, \ldots, n \end{cases}$$

avec les hypothèses (voir les hypothèses (H) I.1.1.b)) :

$$\begin{cases} c_j, a_j \in \mathbb{N}^* \ (j = 1, \ldots, n) \text{ et } b \in \mathbb{N}^* \\ \max a_j \leq b \\ 1 \leq j \leq n \\ b < \sum_{j=1}^{n} a_j \end{cases}$$

La particularité du problème (B) envisage différentes techniques en programmation dynamique étudiées dans les paragraphes qui suivent.

I.2.2. Equation récursive et algorithme de programmation dynamique

Pour un couple (k, y) (k = 1,..., n ; y = 0, 1,..., b) considérons le sous-problème suivant de (B) :

$$(B_k(y)) \begin{cases} \max \sum_{j=1}^{k} c_j x_j \\ \text{s.c} \sum_{j=1}^{k} a_j x_j \leq y \\ x_j = 0 \text{ ou } 1 \quad j = 1,\ldots, k \end{cases}$$

En définissant

$$f_k(y) = v(B_k(y)) \qquad k = 1,\ldots, n \qquad y = 0, 1,\ldots, b$$

la valeur optimale de (B) est à l'évidence $f_n(b)$ et l'équation récursive (1.1) donnée en I.1.2 devient dans ce cas

$$f_0(y) = 0 \qquad y = 0, 1,\ldots, b$$

$$f_k(y) = \begin{cases} f_{k-1}(y) & y = 0,\ldots, a_{k-1} \\ \max\{f_{k-1}(y) ; c_k + f_{k-1}(y - a_k)\} & y = a_k,\ldots, \bar{b} \\ f_k(\bar{b}) & y = \bar{b}+1,\ldots, b \end{cases} \tag{1.20}$$

où $\bar{b} = \min\{\sum_{j=1}^{k} a_j, b\}$

L'algorithme A1 donné en I.1.2 est de compléxité temporelle O(nb) et spatiale O(b) dans le cas du problème du knapsack 0-1 et s'écrit en détail comme suit (voir aussi Toth [70]).

Algorithme A4

But : Calcul $F_b = v(B) = f_n(b)$ où $F_y = f_k(y)$

$\forall y \in \{0, 1, \ldots, b\}$ et $\forall k \in \{0, 1, \ldots, n\}$.

0 $\bar{b} \leftarrow 0$, $F_0 \leftarrow 0$

1 Pour k de 1 à n faire
si $\bar{b} < b$ alors $u \leftarrow \bar{b}$; $\bar{b} \leftarrow \min\{u + a_k, b\}$; $F_{\bar{b}} \leftarrow F_u$;
$y \leftarrow \bar{b} - 1$; Tant que $y > u$ faire $F_y \leftarrow F_u$; $y \leftarrow y - 1$;
fait
fsi
$y \leftarrow \bar{b}$;
Tant que $y \geq a_k$ faire $z \leftarrow y - a_k$; $F \leftarrow c_k + F_z$;
si $F_y < F$ alors $F_y \leftarrow F$;
fsi
$y \leftarrow y - 1$;
fait
fait

I.2.3. Détermination d'une solution optimale d'un algorithme de programmation dynamique

Comme on peut le constater, l'algorithme A4 ne donne pas une solution optimale x^* mais uniquement la valeur du problème (B).

La détermination de x^* peut se faire par différentes techniques ; celles-ci sont applicables à tous les algorithmes de la programmation dynamique.

Les méthodes de détermination d'une solution optimale x^*, que nous avons pû rencontrer dans la littérature sont au nombre de trois :

1° La première méthode est basée sur une idée simple et classique : par exemple, à chaque étape k = 1,..., n de l'algorithme A4, il suffit de gérer parallèlement le vecteur I_k défini par :

$\forall y \in \{0, 1, \ldots, b\}$, $\forall k \in \{1, \ldots, n\}$
$I_0(y) = 0$

$$I_k(y) = \begin{cases} 0 & \text{si } f_k(y) = f_{k-1}(y) \\ 1 & \text{si } f_k(y) = c_k + f_{k-1}(y - a_k) \end{cases}$$

(f_k étant définie en (1.20) (cf. I.2.2)), et de déterminer x^* à la fin de l'algorithme A4 par :

Détermination de x^* :

```
0  y ← b

1  pour k de n à 1 pas -1 faire
       x*_k ← I_k(y) ; y ← y − a_k x*_k
   fait
```

Cette méthode nécessite le stockage des vecteurs $I_k (k = 1, \ldots, n)$ de dimension $(b + 1)$ chacun. Elle est donc de compléxité spatiale $O(nb)$.

2° Dans le cas de l'utilisation d'un calculateur ne permettant pas une telle place mémoire, Magazine et Oguz [48] proposent un algorithme, moins performant en temps d'exécution, de compléxités temporelle $O(n \log_2 nb)$ et spatiale $O(b)$. Pour plus de détails sur cette méthode, on peut consulter la thèse de Fayard et Plateau [18] (page 80).

3° Dans le cas où le nombre de variables n du problème (B) est petit ($n \simeq 30$), Horowitz et Sahni [31] proposent un moyen de gérer x^* de compléxité spatiale $O(b)$, sans bien sûr augmenter la compléxité temporelle de l'algorithme de programmation dynamique : par exemple à chaque étape k de l'algorithme A4, il suffit de gérer parallèlement le vecteur X_k défini par :

$$\forall y \in \{0, 1, \ldots, b\}, \forall k \in \{1, \ldots, n\}$$
$$X_0(y) = 0$$

$$X_k(y) = \begin{cases} X_{k-1}(y) & \text{si } f_k(y) = f_{k-1}(y) \\ X_{k-1}(y - a_k) + 2^{k-1} & \text{si } f_k(y) = f_{k-1}(y - a_k) + c_k \end{cases}$$

et de déterminer x^* à la fin de l'algorithme A4 en décomposant $X_n(b) = \sum_{k=1}^{n} x_k^* \, 2^{k-1}$.

Par exemple si $X_6(b) = 2^0 + 2^3 + 2^5$ alors $x^* = (1, 0, 0, 1, 0, 1)$.

I.2.4. Fonctions knapsacks et algorithme

L'algorithme donné plus loin (I.2.4.3) est basé sur l'étude des fonctions knapsacks définies en (1.20) (cf. I.2.2). Cette étude montrera qu'il n'est pas nécessaire de calculer les f_k en tous points de l'ensemble $\{0, 1,..., b\}$, mais seulement en des points particuliers de celui-ci.

I.2.4.1. Etude des fonctions knapsack

Etant donné $k \in \mathbb{N}^*$, soit f_k la fonction knapsack définie par :

$$f_k(y) = \max\left\{ \sum_{j=1}^{k} c_j x_j \mid \sum_{j=1}^{k} a_j x_j \leq y,\ x_j = 0 \text{ ou } 1,\ j = 1,..., k \right\}$$

pour $y = 0, 1,...$

En définissant par $w_i(k)$ les sommes d'une partie des a_j où $j \in \{1,..., k\}$ telles que :

$$(1.21)\quad \begin{cases} w_0(k) = 0 \\ w_1(k) = \min\{ \sum_{j=1}^{k} a_j x_j \mid w_0(k) < \sum_{j=1}^{k} a_j x_j,\ x_j = 0 \text{ ou } 1,\ j = 1,..., k\} \\ w_2(k) = \min\{ \sum_{j=1}^{k} a_j x_j \mid w_1(k) < \sum_{j=1}^{k} a_j x_j,\ x_j = 0 \text{ ou } 1,\ j = 1,..., k\} \\ \quad \vdots \\ w_i(k) = \min\{ \sum_{j=1}^{k} a_j x_j \mid w_{i-1}(k) < \sum_{j=1}^{k} a_j x_j,\ x_j = 0 \text{ ou } 1,\ j = 1,..., k\} \\ \quad \vdots \\ w_{\bar{J}_k}(k) = \sum_{j=1}^{k} a_j \end{cases}$$

la fonction knapsack f_k vérifie la propriété suivante :

Propriété 1

Etant donnés $k \in \mathbb{N}^*$ *et la liste* $\left\{w_j(k)\right\}_{j=0}^{\bar{J}_k}$, *alors :*

(i) $f_k(w_j(k)) \leq f_k(w_{j+1}(k))$, $j = 0, \ldots, \bar{J}_k$

$$w_{\bar{J}_k+1}(k) = +\infty$$

(ii) *Pour tout* $y \in \mathbb{N}$ *vérifiant* $w_j(k) \leq y < w_{j+1}(k)$, *on a :*

$$f_k(y) = f_k(w_j(k)),$$

c'est-à-dire : les fonctions f_k *sont constantes par morceaux (voir figure 2).*

La démonstration de (i) et (ii) est immédiate.

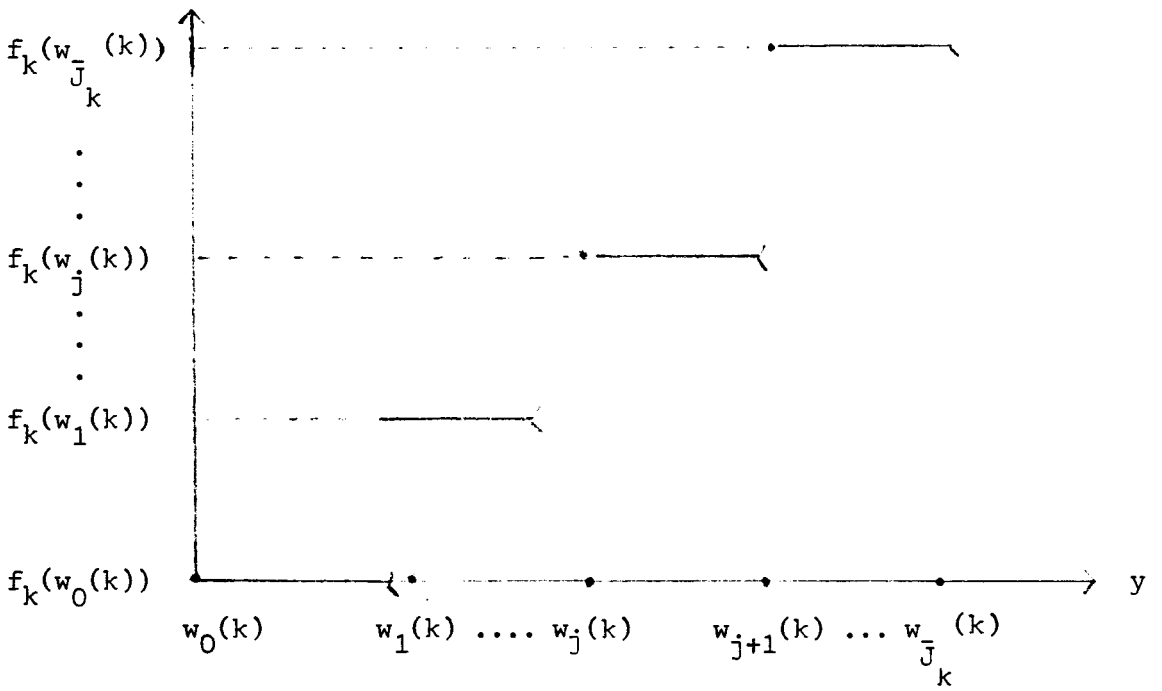


Figure 2

Propriété 2

Etant donnés $k \in \mathbb{N}^*$ *et les listes* $\left\{w_j(k)\right\}_{j=0}^{\bar{J}_k}$, $\left\{w_j(k+1)\right\}_{j=0}^{\bar{J}_{k+1}}$, *alors :*

(i) $\{w_j(k+1)\}_{j=0}^{\bar{J}_{k+1}} = \{w_j(k)\}_{j=0}^{\bar{J}_k} \cup \{w_j(k) + a_{k+1} \quad j = 0, 1, \ldots, \bar{J}_k\}$

(ii) $k \leq \bar{J}_k \leq 2^k - 1$.

Démonstration

(i) : Etant donné $m \in \{0, 1, \ldots, \bar{J}_{k+1}\}$

$$w_m(k+1) = \min\left\{\sum_{j=1}^{k+1} a_j x_j \mid w_{m-1}(k+1) < \sum_{j=1}^{k+1} a_j x_j,\ x_j = 0 \text{ ou } 1,\ j = 1, \ldots, k+1\right\}$$

$$= \sum_{j=1}^{k+1} a_j x_j^*$$

si $x_{k+1}^* = 0$ alors $w_m(k+1)$ est un élément de la liste $\left\{w_i(k)\right\}_{i=0}^{\bar{J}_k}$ sinon $x_{k+1}^* = 1$ et $w_m(k+1)$ est de la forme $a_{k+1} + w_i(k)$ avec $i \in \{0, 1, \ldots, \bar{J}_k\}$.

(ii) : D'après (i), on a $1 + \bar{J}_k \leq \bar{J}_{k+1} \leq 2\,\bar{J}_k + 1$ on déduit à l'aide d'une démonstration par récurrence sur k que $k \leq \bar{J}_k \leq 2^k - 1$.

I.2.4.2. Application au problème du knapsack 0-1

En désignant par J_k $k \in \{0, 1, \ldots, n\}$ l'indice de la somme d'une partie de coefficients a_j vérifiant :

$$w_{J_k}(k) \leq b < w_{J_k+1}(k)$$

la propriété 1 - (ii) entraîne que

$$v(B) = f_n(b) = f_n(w_{J_n}(n))$$

Contrairement à l'algorithme A4 (cf. I.2.2) où les fonctions f_k sont calculées en tous points $y \in \{0, 1, \ldots, b\}$; celles-ci seront calculées uniquement aux points critiques $w_i(k)$ $i \in \{0, 1, \ldots, J_k\}$ avec J_k en général beaucoup plus petit que b.

En faisant correspondre à tout point critique $w_i(k)$ $i \in \{0, 1, \ldots, J_k\}$ la valeur de la fonction f_k ; c'est-à-dire :

$$v_i(k) = f_k(w_i(k))$$

les listes $\left\{(w_i(k), v_i(k))\right\}_{i=0}^{J_k}$ pour tout $k \in \{1, 2, \ldots, n\}$ seront construites récursivement en n étapes.

Il est à noter que par construction, on a :

1° $0 = w_0(k) < w_1(k) < \ldots < w_{J_k}(k) \leq b$

2° $0 = v_0(k) \leq v_1(k) \leq \ldots \leq v_{J_k}(k) \leq v(B)$

3° $v(B) = v_{J_n}(n)$

En introduisant la notion de dominance entre les paires (Ahrens et Finke [1], Horowitz et Sahni [31] et Toth [70]) comme suit :

La paire (W, V) est dominée par (W', V') si

$W' \leq W$ et $V' \geq V$

les inégalités larges dans 2° sont remplacées par des inégalités strictes.

I.2.4.3. Algorithme A5

Connaissant la $k^{\text{ième}}$ liste $\left\{(w_j(k), v_j(k)\right\}_{j=0}^{\bar{J}_k}$, le principe pour construire la $(k+1)^{\text{ième}}$ liste $\left\{(w_j(k+1), v_j(k+1)\right\}_{j=0}^{J_{k+1}}$ est le même que celui donné dans la propriété 2 - (ii) c'est-à-dire :

$$\left\{(w_j(k+1), v_j(k+1)\right\}_{j=0}^{J_{k+1}} \subset \left\{(w_j(k), v_j(k)\right\}_{j=0}^{J_k} \cup \left\{(w_j(k) + a_{k+1}, v_j(k) + c_{k+1}) \mid w_j(k) + a_{k+1} \leq b \quad j = 0, 1, \ldots, J_k\right\}$$

Cette relation récursive entre les listes permet de résoudre le problème (B) par un algorithme à n étapes.

Cet algorithme est similaire à celui de Toth [70], mais avec une implémentation améliorée permettant de réduire la place mémoire de moitié :

à chaque étape de l'algorithme, le principe est le suivant :

- utiliser une seule pile pour les listes $\left\{w_j(k)\right\}_{j=0}^{J_k}$ et $\left\{w_j(k+1)\right\}_{j=0}^{J_{k+1}}$,
- utiliser une seule pile pour les listes $\left\{v_j(k)\right\}_{j=0}^{J_k}$ et $\left\{v_j(k+1)\right\}_{j=0}^{J_{k+1}}$,

de manière à ce qu'il n'y ait pas de destruction d'informations utiles. Par exemple, pour fixer les idées, si les éléments de la liste $\left\{w_j(k)\right\}_{j=0}^{J_k}$ sont rangés à partir du bas de la pile alors les éléments de la liste $\left\{w_j(k+1)\right\}_{j=0}^{J_{k+1}}$ seront rangés à partir du haut de la même pile, ainsi de suite.

Pour illustrer l'implémentation de l'algorithme suivant considérons, l'exemple ci-dessous (figure 3) :

Exemple : $n = 4$, $b = 19$

$(c_i) = (10, 9, 8, 7)$

$(a_i) = (8, 11, 7, 8)$

7			19	19			19	19
6			8	10			15	18
5			0	0	19	19	8	10
4					15	18	7	8
3					8	10	0	0
2	8	10			7	8		
1	0	0			0	0		
j	w_j	v_j	w_j	v_j	w_j	v_j	w_j	v_j
	k = 1		k = 2		k = 3		k = 4	

Figure 3

La solution optimale $x^* = (1, 1, 0, 0)$ de valeur $v(B) = 19$.

Remarque : (i) Il est à noter dans l'exemple ci-dessus que le nombre total des couples (w_j, v_j) des quatre étapes est 15, alors que le nombre des différentes sommes w_j inférieures ou égales à b est 24. Cette réduction est due à l'utilisation de la dominance entre les couples.

Algorithme détaillé : Algorithme A5

But : Calcul de la valeur v(B).

% Initialisation %

$w_1 \leftarrow 0$; $v_1 \leftarrow 0$; $w_2 \leftarrow a_1$; $v_2 \leftarrow c_1$; $j \leftarrow 2$; $w_{h+1} \leftarrow b+1$

% h étant la hauteur maximale allouée aux piles %

% j (resp. m) décrit l'indice des éléments de la nouvelle liste (resp. ancienne liste) %

Pour k de 2 à n faire

$m \leftarrow j$; $i \leftarrow j$; $j \leftarrow h+1$;

Tant que $w_i + a_k > b$ faire $i \leftarrow i-1$ fait

% Comparer $(w_i + a_k, v_i + c_k)$ et (w_m, v_m) ; les classer dans l'ordre croissant %

Tant que $i \neq 0$ et $m \neq 0$ faire $w \leftarrow w_i + a_k$; $v \leftarrow v_i + c_k$;

% trois cas possibles : $w > w_m$; $w = w_m$; $w < w_m$ %

* $w > w_m$: si $w \geq w_j$ alors $i \leftarrow i-1$; sinon $j \leftarrow j-1$;
si $v_m \geq v$ alors $w_j \leftarrow w_m$; $v_j \leftarrow v_m$; $i \leftarrow i-1$; $m \leftarrow m-1$;
sinon $w_j \leftarrow w$; $v_j \leftarrow v$; $i \leftarrow i-1$ fsi
fsi

* $w = w_m$: si $w \geq w_j$ alors $m \leftarrow m-1$; $i \leftarrow i-1$; sinon $j \leftarrow j-1$; $w_j \leftarrow w$;
si $v_m < v$ alors $v_j \leftarrow v$; $m \leftarrow m-1$; $i \leftarrow i-1$;
sinon $v_j \leftarrow v_m$; $m \leftarrow m-1$; $i \leftarrow i-1$ fsi
fsi

```
* w < w_m : tant que w < w_m faire
              si w_m ≥ w_j alors m ← m-1
                 sinon j ← j - 1  ; si v_m ≤ v alors w_j ← w ;
                                       v_j ← v ; w ← w_m ; m ← m-1 ; i ← i-1 ;
                                    sinon w_j ← w_m ; v_j ← v_m ; m ← m-1   fsi
              fsi
           fait
  fait

Tant que m ≠ 0 faire j ← j-1 ; w_j ← w_m ; v_j ← v_m ; m ← m-1 fait
si k = n stop
k ← k+1
m ← j ; i ← j ; j ← 0 ;

Tant que a_k < w_m faire j ← j+1 ; w_j ← w_m ; v_j ← v_m ; m ← m+1  fait

% Comparer (w_i + a_k, v_i + c_k) et (w_m, v_m) et les classer dans l'ordre
croissant %

Tant que i ≠ h et m ≠ h faire w ← w_i + a_k ; v ← v_i + c_k ;


% trois cas possibles : w > w_m ; w = w_m ; w < w_m %

* w > w_m : Tant que w_m < w faire
           si v_m ≤ v_j alors m ← m+1 ;
              sinon j ← j+1 ; w_j ← w_m ; v_j ← w_m ;
                 si v_j ≥ v alors i ← i+1 ; v ← v_m ; m ← m+1 ;
                    sinon m ← m+1 fsi
           fsi
           fait

* w = w_m : si v_m ≤ v_j alors m ← m+1 ; sinon j ← j+1 ; w_j ← w ;
              v_j ← v_m ; si v_j < v alors v_j ← v ; i ← i+1 ;
                        sinon m ← m+1, i ← i+1  fsi
           fsi
```

```
* w < w_m : si v < v_j alors i ← i+1 ; sinon j ← j+1 ; w_j ← w ; v_j ← v ;
                si v_m ≤ v alors m ← m+1 ; i ← i+1 ; sinon i ← i+1 fsi
           fsi
  fait
      Tant que i ≠ h et w ≤ b faire v ← v_i + c_k
              si v ≤ v_j alors i ← i+1 ; w ← w_i + a_k ;
                 sinon j ← j+1 ; w_j ← w ; v_j ← v ; i ← i+1, w ← w_i + a_k fsi
      fait
  fait
```

Complexité de l'algorithme A5

A chaque étape $k \in \{1,\ldots, n\}$, l'algorithme A5 construit J_k couple (w_j, v_j) $(j = 0, 1,\ldots, J_k)$. Ce nombre J_k vérifie :

- $k \leq J_k \leq 2^k - 1$ (cf. I.2.4.1. Propriété 2 - (ii))

- $J_k \leq b$

d'où $k \leq J_k \leq \min(2^k - 1, b)$

On déduit que l'algorithme A5 a les complexités temporelle $O(\min(nb, 2^n))$ et spatiale $O(\min(b, 2^n))$.

I.2.5. Algorithme basé sur la fusion et la division de problèmes : Algorithme A6

Dans le cas de l'utilisation d'un calculateur ne permettant pas l'encombrement en place mémoire $O(\min(2^n, b))$ donné par l'algorithme précédent A5, Horowitz et Sahni [31] proposent une méthode de compléxité temporelle $O(\min(2^{n/2}, nb))$ et spatiale $O(\min(2^{n/2}, b))$.

Le principe de la méthode est :

(1) Diviser le problème d'origine (B) en deux sous-problèmes (B1) et B2) de $q = \lfloor n/2 \rfloor$ (respectivement $r = n-q$) variables et ayant tous les deux le même second membre b que celui de (B).

② Résoudre les sous-problèmes (B1) et (B2) par l'algorithme A5 (cf. I.2.4.3). Désignons par L1 (resp. L2) la liste $\{w1_j, v1_j)\}_{j=0}^{\ell_1}$ (resp. $\{(w2_j, v2_j)\}_{j=0}^{\ell_2}$) obtenue à la fin de la résolution de (B1) (resp. (B2)) par l'algorithme A5.

③ Fusionner les listes L1 et L2 pour trouver la valeur de (B) (voir figure 4) en utilisant l'algorithme suivant :

Algorithme de fusion : Algorithme A7

But : Calcul la valeur de (B) (v = v(B)) en fusionnant L1 et L2

% Initialisation %

$k \leftarrow 1$; $j \leftarrow \ell_1$; $v \leftarrow 0$; $w2_{\ell_2+1} \leftarrow b+1$; $w \leftarrow w1_j + w2_k$

Tant que ($w \leq b$ ou $j \neq 0$) faire

tant que $w \leq b$ faire $k \leftarrow k+1$; $w \leftarrow w1_j + w2_k$ fait
$p \leftarrow v1_j + v2_{k-1}$
si $p > v$ alors $v \leftarrow p$ fsi
tant que ($w > b$ et $j \neq 1$) faire $j \leftarrow j-1$; $w \leftarrow w1_j + w2_k$ fait
si $k \neq \ell_2+1$ alors $k \leftarrow k+1$; $w \leftarrow w1_j + w2_k$ fsi

fait

Exemple : Considérons le problème (B) donné précédemment (cf. I.2.4.3.) où ses données sont :

$n = 4$; $b = 19$
$(c_i) = (10, 9, 8, 7)$; $(a_i) = (8, 11, 7, 8)$

Les trois étapes de la méthode sont :

* Division du problème (B) en deux sous-problèmes (B1) et (B2) suivants :

$$(B1)\ \begin{cases} \max\ 10x_1 + 9x_2 \\ s.c\ 8x_1 + 11x_2 \le 19 \\ x_j = 0 \text{ ou } 1 \quad j = 1, 2 \end{cases} \quad , \quad (B2)\ \begin{cases} \max\ 8x_3 + 7x_4 \\ s.c\ 7x_3 + 8x_4 \le 19 \\ x_j = 0 \text{ ou } 1 \quad j = 3, 4 \end{cases}$$

* Résolution de (B1) et (B2) par l'algorithme A5, soient L1 et L2 leurs listes respectives (figure 4) :

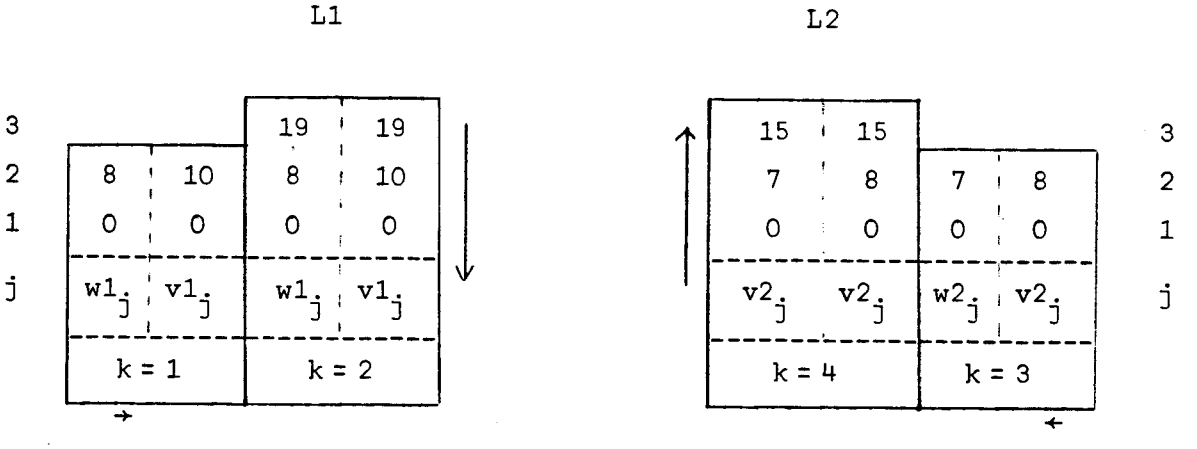


Figure 4

* Fusion de L1 et L2 en parcourant les listes dans le sens des flèches (figure 4) par l'algorithme A7 :

(19, 19) + (0, 0) = (19, 19) ; v ← 19
(8, 10) + (7, 8) = (15, 18) pas d'amélioration de v
(0, 0) + (15, 15) = (15, 15) pas d'amélioration de v
v(B) = 19 □

Complexité de la méthode

Dans la phase (2) de la méthode, la résolution de chaque sous-problème est de compléxité temporelle $O(\min(2^{n/2}, bn/2))$ (voir l'algorithme A5 - I.2.4.3), on déduit une compléxité temporelle $O(\min(2^{n/2}, nb))$ pour les deux sous-problèmes.

Dans la phase (3) l'algorithme de fusion à une compléxité temporelle $O(\ell_1 + \ell_2)$ ($\ell_1 + 1 = |L1|$ et $\ell_2 + 1 = |L2|$ et $\ell_2, \ell_1 \le \min(2^{n/2}, b)$). On déduit alors que l'algorithme A6 est de compléxité temporelle $O(\min(2^{n/2}, nb))$ et spatiale $O(\min(2^{n/2}, b))$.

Remarque : L'amélioration apportée par Horowitz et Sahni, peut-elle l'être davantage si on divise le problème d'origine en k sous-problèmes ?

La réponse à cette question n'est pas tout-à-fait négative. La résolution des k sous-problèmes est de compléxités temporelle et spatiale $O(k2^{n/k})$.

Cependant, il n'est pas possible de fusionner les k sous-problèmes dans un temps inférieur à $O(2^{n/2})$.

Dans le cas où k = 4, Ahrens et Finke [1] donnę un algorithme de compléxité temporelle $O(2^{n/2})$ et spatiale $O(2^{n/4})$.

Dans le cas k = 6, Karnin [38] donne un algorithme de compléxités temporelle $O(2^{n/2})$ et spatiale $O(2^{n/6})$, mais en utilisant plusieurs processeurs.

I.2.6. Stratégie "Branch and Bound" dans la programmation dynamique

Les méthodes d'énumération implicite, dites aussi arborescentes par séparation et évaluation (branch and bound) sont considérablement améliorées par l'utilisation d'évaluations par défaut (minorants) et par excès (majorants) permettant ainsi une réduction importante que se soit au niveau de contraintes, de variables ou de sous-problèmes générés. L'utilisation de cette technique dans la programmation dynamique (voir Proschan et Bray [63], Weingartner et Ness [72] Morin et Marsten [55, 56, 57] Toth [70] et Minoux [54]) évite - par exemple dans le cas du problème (B) - de stocker un nombre important de combinaisons inutiles à la détermination de la valeur v(B).

Plaçons-nous à une étape $k \in \{1, 2, \ldots, n\}$ de l'algorithme A5 (cf. I.2.4.3) tout couple $(w_j(k), v_j(k))$ $j \in \{1, \ldots, J_k\}$ vérifiera l'un des tests a), b) ou c) ci-dessous, sera écarté une fois pour toute dans les étapes qui suivent (voir Toth [70]).

a) $w_j(k) < A_k = \max\{b - \sum_{i=k+1}^{n} a_i, 1\}$ et $w_{j+1} \leq A_k$

b) $b - \min_{i=k+1,\ldots,n} \{a_i\} < w_j(k) < b$ et $j < J_k$

c) Soit $\underline{v}_k(B)$ un minorant de la valeur du problème $(B_k(b))$ (défini en I.2.2) tel que :

$$\underline{v}_0(B) = 0$$
$$\underline{v}_k(B) = \max\{v_{J_k}(k) + \underline{v},\ \underline{v}_{k-1}(B)\}$$

où $\underline{v}$ est un minorant de la valeur du sous-problème :

$$\max \sum_{i=k+1}^{n} c_i x_i \text{ s.c. } \sum_{i=k+1}^{n} a_i x_i \leq b - w_{J_k}(k),\ x_i = 0 \text{ ou } 1,\ i = k+1,\ldots, n$$

et $\bar{v}_{k,j}$ est un majorant de la valeur du sous-problème :

$$\max \sum_{i=k+1}^{n} c_i x_i \text{ s.c. } \sum_{i=k+1}^{n} a_i x_i \leq b - w_j(k),\ x_i = 0 \text{ ou } 1,\ i = k+1,\ldots, n$$

Test d'élimination

$$v_j(k) + \bar{v}_{k,j} \leq \underline{v}_k(B). \qquad \square$$

CHAPITRE II

KNAPSACK 0-1 A CONTRAINTE PARALLELE A LA FONCTION ECONOMIQUE

INTRODUCTION

Les études faites, sur le problème du knapsack 0-1 (B), montrent que les algorithmes d'énumération implicite, ont une mauvaise performance lorsqu'il existe une forte corrélation entre les coefficients de la contrainte et ceux de la fonction économique. Dans ce cas, les évaluations par défaut ou par excès de v(B) n'apportent pratiquement aucune amélioration (voir Martello et Toth [50]).

Le problème du knapsack 0-1, où $c_j = a_j$ pour tout j ; appelé "knapsack à contrainte parallèle à la fonction économique" noté (S) - dans la littérature, on lui donne le nom du "Subset-Sum problem" (Martello et Toth [51]), ou "value-Independent problem" (Faaland [16], Balas et Zemel [3]), ou "Stickstacking-problem" (Ahrens et Finke [1]) - est un cas extrême de la correlation où tous les majorants de v(S) sont égaux à la valeur triviale b.

Cependant, le problème (S) peut être résolu par trois types d'algorithmes :

- Algorithmes de programmation dynamique pure (Ahrens et Finke [1], Faaland [16], Toth [70]).

- Algorithmes d'énumération implicite pure pour le knapsack général (B) adaptés au problème (S).

- Algorithmes hybrides combinant les deux types précédents (Martello et Toth [51]) (cf. II.1).

L'analyse des résultats de ces méthodes conduit aux conclusions suivantes :

. Les premiers sont inutilisables lorsque b est trop grand.

. Les temps de calcul des seconds algorithmes augmentent exponentiellement lorsque très peu de solutions réalisables (voir aucune) atteignent le second membre b de la contrainte ; ces problèmes sont dits "difficiles".

. Les algorithmes de la troisième classe sont efficaces pour des problèmes "difficiles" avec n petit ($n \leq 40$) et des problèmes "faciles" (c'est-à-dire de nombreuses solutions réalisables atteignent le second membre b) avec n grand ($n \geq 1000$).

Mais leur désavantage est l'utilisation systématique de l'énumération implicite même lorsque l'encombrement en place mémoire pour la programmation dynamique n'est pas excessif ; ce principe aboutit à leur inéfficacité pour les problèmes "difficiles" avec n grand et b petit en regard de la capacité en place mémoire du calculateur.

L'objet de ce chapitre est de donner un algorithme général noté I.D.S (cf. II.2) capable de résoudre une vaste classe de problèmes tout en restant meilleur en temps de calcul par rapport aux méthodes précédentes (cf. II.3).

L'algorithme I.D.S se déroule dans l'ordre suivant :

- Dans un premier temps, une procédure de la programmation dynamique est utilisée ; elle peut être arrêtée dès qu'une solution de valeur b est trouvée.

- Le schéma classique d'énumération n'est utilisé que lorsque la place mémoire pour la procédure dynamique devient excessive ; chaque sous-vecteur produit par l'énumération est complété par la partie dynamique à l'aide de l'algorithme A7 de fusion (cf. I.2.5).

II.1. LES METHODES EXISTANTES

II.1.1. Position du problème

Etant données n valeurs a_i ($i = 1, n$), entières positives et une capacité b entière positive, le problème du knapsack 0-1 dont la contrainte est parallèle à la fonction économique est de sélectionner parmi les n valeurs a_i $i \in \{1,\ldots, n\}$ celles qui ont une somme maximale inférieure ou égale à b. Le problème se traduit mathématiquement par :

$$(S)\begin{cases}\max \sum_{j=1}^{n} a_j x_j \\ \text{s.c} \sum_{j=1}^{n} a_j x_j \leq b \\ x_j = 0 \text{ ou } 1 \quad j = 1,\ldots, n\end{cases}$$

On rappelle (cf. I.1.1. b)) que, sans perdre de généralité, les données satisfont les hypothèses suivantes :

$$\begin{cases}0 < a_j \leq b & j = 1,\ldots, b \\ \sum_{j=1}^{n} a_j > b\end{cases}$$

On note x^* une solution optimale de (S) ($v(S) = ax^*$).

II.1.2. Méthode de Faaland [16] (1973)

La méthode de Faaland est basée sur la gestion du nombre de solutions de l'équation (2.1) définie ci-dessous et non sur les valeurs des solutions de celle-ci, comme c'est le cas des équations classiques étudiées au Chapitre I. Cette méthode améliore la complexité temporelle expérimentale de la programmation dynamique, de l'ordre de 50 % dans le cas du problème (S) :

Etant donnés $k \in \{1,\ldots, n\}$ et $y \in \{0, 1,\ldots, n\}$, en définissant par $g_k(y)$ le nombre de solutions de l'équation diophantienne :

$$\sum_{j=1}^{k} a_j x_j = y \tag{2.1}$$

alors,

$$g_1(y) = \begin{cases}1 & \text{si } y = 0 \text{ ou } a_1 \\ 0 & \text{sinon}\end{cases} \tag{2.2}$$

$$g_k(y) = \begin{cases}g_{k-1}(y), & \forall y \in \{0, 1,\ldots, a_k-1\} \\ g_{k-1}(y) + g_{k-1}(y - a_k), & \forall y \in \{a_k,\ldots, \bar{b}\} \\ 0 & \forall y \in \{\bar{b}+1,\ldots, b\}\end{cases} \tag{2.3}$$

où

$$\bar{b} = \min\{\sum_{j=1}^{k} a_j, b\}$$

et la valeur optimale de (S) est :

$$v(S) = \max\{y \mid y \leq b \text{ et } g_n(y) > 0\} \qquad (2.4)$$

La solution optimale x^* est donnée par :

```
% v(S) étant calculée par (2.4) %
  y ← v(S)
pour k de n à 2 pas -1 faire
      si g_{k-1}(y - a_k) > 0 alors x*_k ← 1 ; y ← y - a_k
                              sinon x*_k ← 0
      fsi
fait
x*_1 ← y/a_1
```

Remarques :

(i) En rappelant que l'équation (1.20) (cf. I.2.2) s'écrit dans le cas du problème (S) :

$$f_0(y) = 0 \qquad y = 0, 1, \ldots, b$$

$$(2.5) \qquad f_k(y) = \begin{cases} f_{k-1}(y) & y = 1, \ldots, a_k - 1 \\ \max\{f_{k-1}(y) ; a_k + f_{k-1}(y - a_k)\} & y = a_k, \ldots, \bar{b} \\ \sum_{j=1}^{k} a_j & y = \bar{b} + 1, \ldots, b \end{cases}$$

(où $f_k(y) = \max \sum_{j=1}^{k} a_j x_j$ s.c $\sum_{j=1}^{k} a_j x_j \leq y$, $x_j = 0$ ou $1 \quad j = 1, \ldots, k$).

il est clair qu'en (2.5), on fait une comparaison de plus par rapport à (2.3) entre a_k et $\bar{b}$. On déduit qu'un algorithme basé sur (2.3) pourrai avoir un gain de calcul de 50 % par rapport à celui basé sur (2.5).

(ii) Contrairement à (2.5), (2.3) est généralisable (Faaland [16]) au problème :

$$\begin{cases} \max \sum_{j=1}^{n} a_j x_j \\ \text{s.c} \sum a_j x_j \leq b \\ \qquad 0 \leq x_j \leq u_j \qquad j = 1, \ldots, n \\ \qquad\quad x_j \text{ entier} \end{cases}$$

où $u_j \in \mathbb{N}$ $j \in \{1, \ldots, n\}$.

II.1.3. Méthode d'Ahrens et Finke : Algorithme A8 [1]

Cette méthode consiste, d'une part à diviser le problème (S) en quatre sous-problèmes ayant le même nombre de variables ; d'autre part à faire une fusion (cf. I.2.5) des quatre sous-problèmes pour trouver la valeur du problème (S).

Pour simplifier l'exposé, on pose n = 4m. En considérant les sous-problèmes de (S) suivants :

$$(SA) \quad \max \sum_{j \in A} a_j x_j \text{ s.c } \sum_{j \in A} a_j x_j \leq b,\ x_j = 0 \text{ ou } 1 \quad \forall j \in A$$

(où A désigne un ensemble égal tour à tour à :

H = {1,..., m} ; I = {m+1,..., 2m} ; K = {2m+1,..., 3m} ;

M = {3m+1,..., n})

L'algorithme A8 d'Ahrens et Finke comporte deux phases principales :

PHASE 1 : Construire par la programmation dynamique les listes $L.A = \left\{WA_j\right\}_{j=0}^{J_A}$ (où WA_j désignent les sommes partielles des a_j, $j \in A$, cf. I.2.4) correspondant aux problèmes (SA) (A étant égal tour à tour à H, I, K, M).

PHASE 2 : Fusionner les quatre listes (même principe que pour deux listes (cf. I.2.5) voir l'exemple ci-dessous) pour trouver v(S) :

$$v(S) = \max\{WH_i + WI_j + WK_k + WM_\ell = W \mid W \leq b\}$$

$i = 0,\ldots, J_H$; $j = 0,\ldots, J_I$; $k = 0,\ldots, J_K$; $\ell = 0,\ldots, J_M$ □

Exemple : Considérons l'exemple suivant à 8 variables :

$$(S)\begin{cases}\max\ 2x_1 + 4x_2 + 6x_3 + 8x_4 + 10x_5 + 12x_6 + 14x_7 + 16x_8 \\ \text{s.c}\ 2x_1 + 4x_2 + 6x_3 + 8x_4 + 10x_5 + 12x_6 + 14x_7 + 16x_8 \leq 17 \\ \quad x_j = 0 \text{ ou } 1 \qquad j = 1,\ldots, 8\end{cases}$$

Phase 1 :

$H = \{1, 2\}$, $I = \{3, 4\}$, $K = \{5, 6\}$, $M = \{7, 8\}$

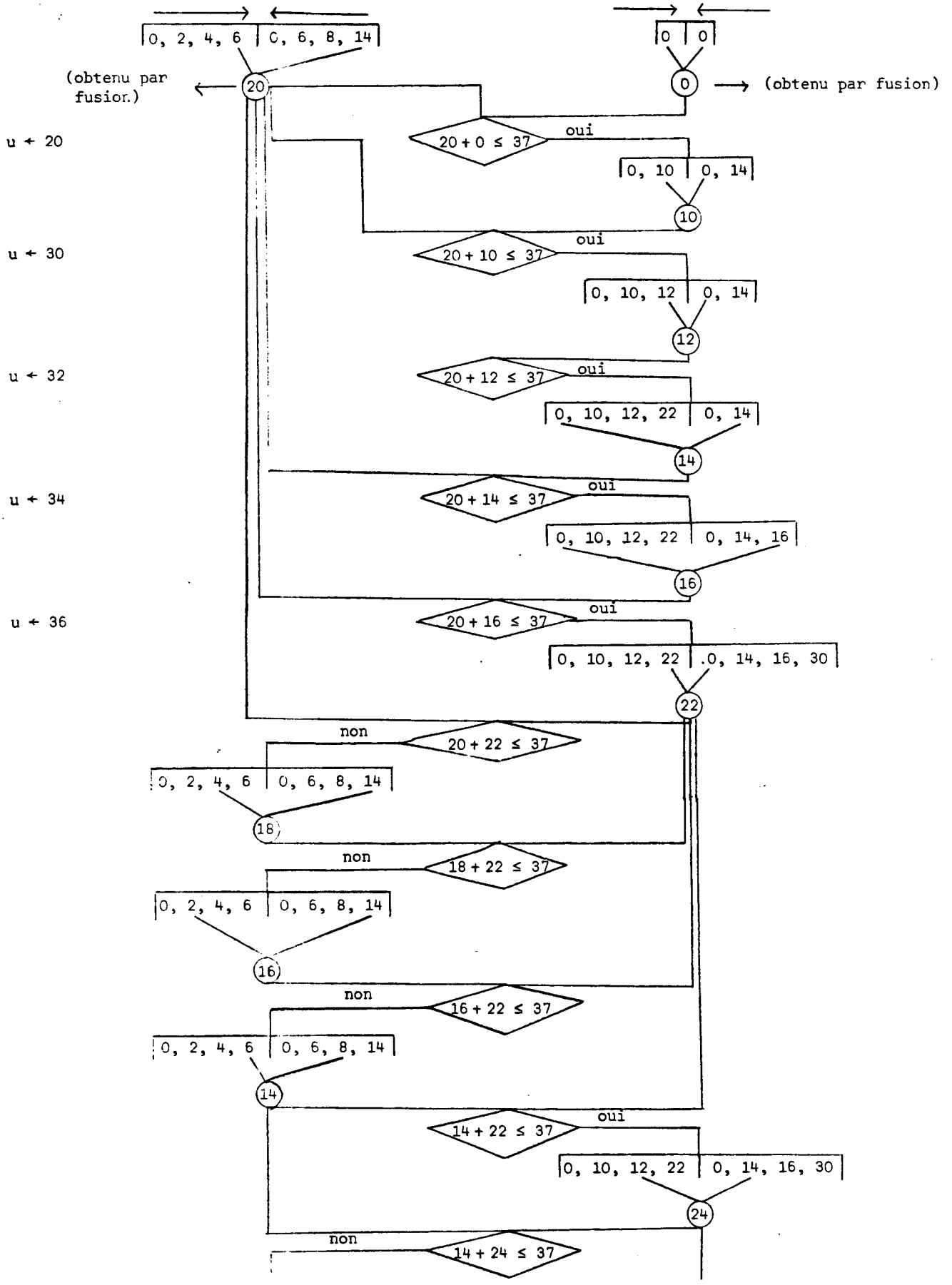
$LH = \{0, 2, 4, 6\}$, $LI = \{0, 6, 8, 14\}$, $LK = \{0, 10, 12, 22\}$,

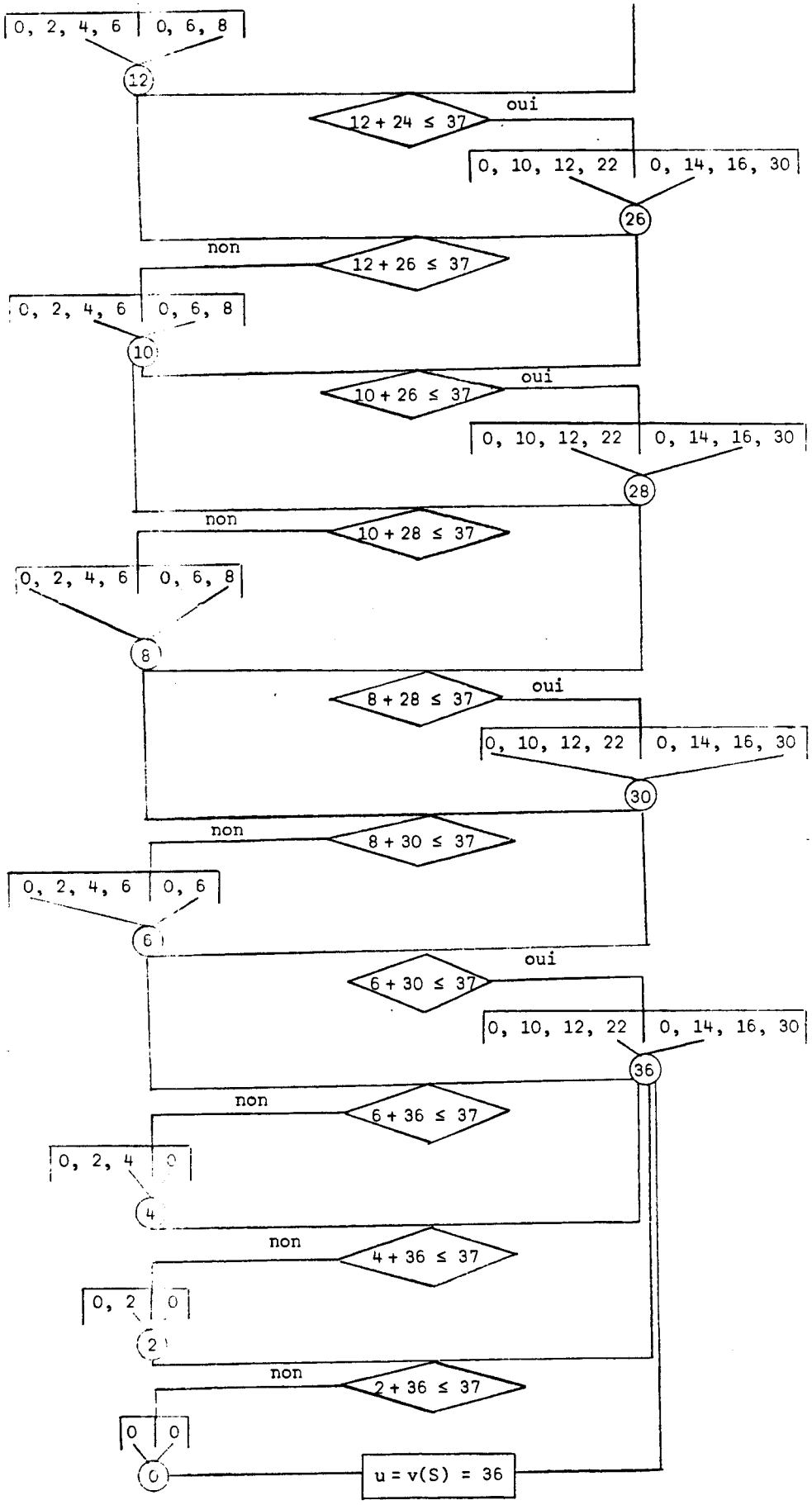
$LM = \{0, 14, 16, 30\}$

Phase 2 :

But : trouver $u = v(S)$.

L'exploration des listes se fait dans le sens des flèches indiquées :


0, 2, 4, 6 | 0, 6, 8, 14
(obtenu par fusion.)
20
0 | 0
0
(obtenu par fusion)
u ← 20
20 + 0 ≤ 37
oui
0, 10 | 0, 14
10
u ← 30
20 + 10 ≤ 37
oui
0, 10, 12 | 0, 14
12
u ← 32
20 + 12 ≤ 37
oui
0, 10, 12, 22 | 0, 14
14
u ← 34
20 + 14 ≤ 37
oui
0, 10, 12, 22 | 0, 14, 16
16
u ← 36
20 + 16 ≤ 37
oui
0, 10, 12, 22 | 0, 14, 16, 30
22
non
20 + 22 ≤ 37
0, 2, 4, 6 | 0, 6, 8, 14
18
non
18 + 22 ≤ 37
0, 2, 4, 6 | 0, 6, 8, 14
16
non
16 + 22 ≤ 37
0, 2, 4, 6 | 0, 6, 8, 14
14
14 + 22 ≤ 37
oui
0, 10, 12, 22 | 0, 14, 16, 30
24
non
14 + 24 ≤ 37


0, 2, 4, 6 | 0, 6, 8
12
12 + 24 ≤ 37
oui
0, 10, 12, 22 | 0, 14, 16, 30
26
non
12 + 26 ≤ 37
0, 2, 4, 6 | 0, 6, 8
10
10 + 26 ≤ 37
oui
0, 10, 12, 22 | 0, 14, 16, 30
28
non
10 + 28 ≤ 37
0, 2, 4, 6 | 0, 6, 8
8
8 + 28 ≤ 37
oui
0, 10, 12, 22 | 0, 14, 16, 30
30
non
8 + 30 ≤ 37
0, 2, 4, 6 | 0, 6
6
6 + 30 ≤ 37
oui
0, 10, 12, 22 | 0, 14, 16, 30
36
non
6 + 36 ≤ 37
0, 2, 4 | 0
4
non
4 + 36 ≤ 37
0, 2 | 0
2
non
2 + 36 ≤ 37
0 | 0
0
u = v(S) = 36

Fin

L'intérêt de diviser le problème en quatre sous-problèmes au lieu de deux, est uniquement pour réduire la compléxité spatiale, puisqu'on ne peut pas faire mieux au niveau de la compléxité temporelle que de diviser par deux (cf. I.2.5).

Pour fixer les idées, supposons sur cet exemple que le nombre d'opérations est égal au nombre d'éléments dans les listes :

- Division par quatre

. La phase 1 nécessite $4 \times 2^2 = 16$ opérations.

. En ajoutant le nombre d'éléments des listes l'un à l'autre (puisque la fusion est linéaire (cf. I.2.5) par rapport au nombre d'éléments des listes) la phase 2 nécessite 140 opérations.

Le nombre total d'opérations est donc 156.

Il faut $4 \times 2^2 = 16$ places mémoire.

- Division par deux

. La phase 1 nécessiterai $2 \times 2^4 = 32$ opérations.

. Le nombre d'opérations de la fusion de deux listes (chacune ayant 2^4 éléments) est $2^4 + 2^4 = 32$.

Le nombre total d'opérations est donc 64.

Il faut $2 \times 2^4 = 32$ places mémoire.

Compléxité de l'algorithme A8

Phase 1 : La construction de chaque liste par la programmation dynamique nécessite une compléxité temporelle $O(\min(2^{n/4}, bn/4))$ et spatiale $O(\min(2^{n/4}, b))$ ce qui donne à la phase 1 les compléxités temporelle de $O(\min(42^{n/4}, bn))$ et spatiale de $O(\min(42^{n/4}, 4b))$.

Phase 2 : On suppose que :

- $|L.H| = |L.I| = |L.K| = |L.M| = \min(2^{n/4}, b) = \alpha$
- $WH_{J_H} < WI_1$ (WH_{J_H} (resp. WI_1) étant le grand (resp. le plus petit) élément non nul de la liste L.H (resp. L.I)).
- $WH_{J_H} + WI_{J_I} \geq b$.

Sous les hypothèses précédentes, l'énumération de chacun des α plus grands éléments de la liste L.H $\cup$ I (dans l'exemple : 20, 18, 16, 14) nécessite une fusion de 2α opérations ; celles des α valeurs est donc de $2\alpha^2$ opérations. En négligeant le coût de l'énumération des autres éléments de L.H $\cup$ I et puisque le travail ci-dessus est le même que sur L.K $\cup$ I, on déduit

- d'une part que la compléxité temporelle de la phase 2 est $O(\min(2^{n/2}, 2b^2)$

- d'autre part que les compléxités de l'algorithme A8 :
  - temporelle $O(\min(22^{n/2} + 42^{n/4}, bn + 2b^2))$
  - spatiale $O(\min(42^{n/4}, 4b))$. □

II.1.4. Méthode de Martello et Toth : 1982 [51]

La méthode de Martello et Toth est une méthode mixte utilisant de la programmation dynamique et de l'énumération implicite qui se déroule en deux phases :

Phase 1 : La programmation dynamique est appliquée à une partie du problème (S) seulement.

Phase 2 : Un schéma d'énumération implicite est appliqué à la partie complémentaire

PHASE 1 :

Etant donné, m un entier naturel ($m < n$), la liste $L = \{w_j\}_{j=0}^{J}$ est constituée par toutes les sommes des différentes combinaisons parmi les m derniers a_i, vérifiant :

$$w_0 < w_1 < \ldots, w_J \leq b$$

Etant donnés, deux entiers naturels m' ($m < m' < n$) et $\bar{b}$ ($\bar{b} < b$), la liste $L' = \{w'_j\}_{j=0}^{J'}$ contient toutes les sommes des différentes combinaisons parmi les m' derniers a_i, vérifiant :

$$w'_0 < w'_1 < \ldots, w'_{J'} \leq \bar{b}$$

En posant $n_1 = n - m$ et $n_2 = n - m'$, la figure 5 montre la place occupée par les deux listes L et L'.

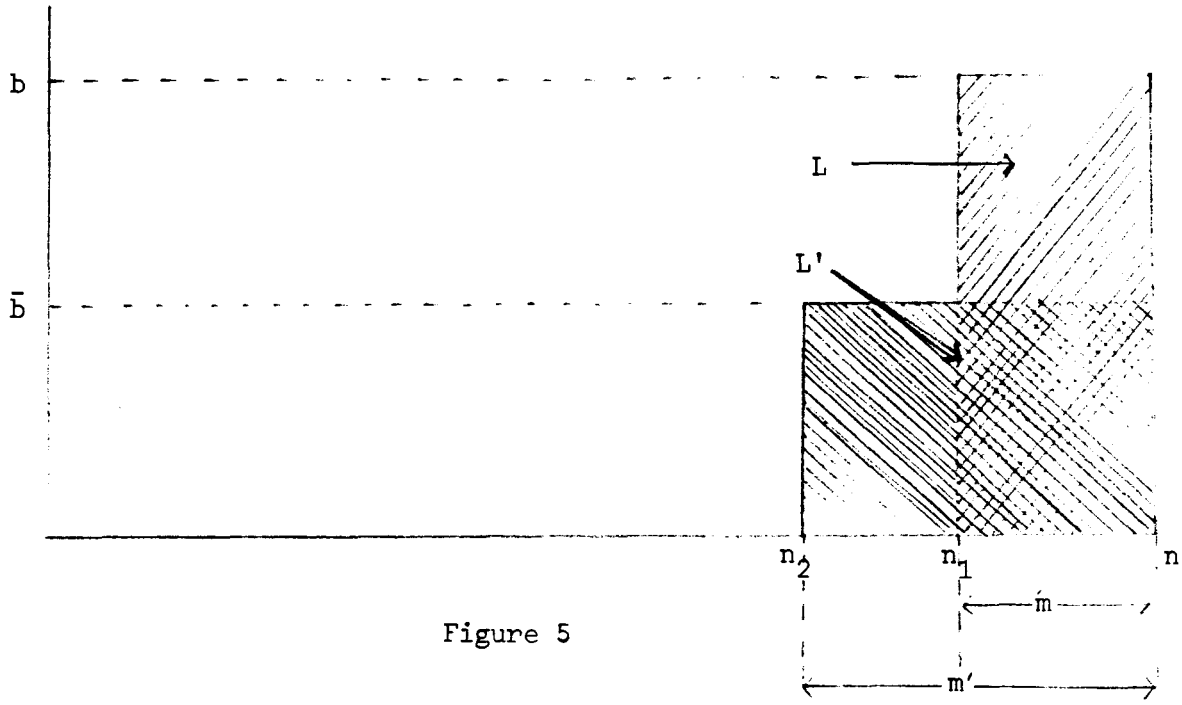


Figure 5

Les constantes m, m' et $\bar{b}$ sont fixées expérimentalement par les auteurs de manière à tenir compte de la "difficulté" de leurs problèmes traités :

S'il existe de nombreuses solutions x telles que ax = b, alors le problème (S) est considéré comme facile ; par conséquent, il est plus intéressant d'avoir m, m' et $\bar{b}$ petits puisque l'énumération implicite est efficace pour un tel problème.

Pour les problèmes difficiles (peu de solutions x telles que ax = b) il faut au contraire favoriser la programmation dynamique en augmentant les paramètres m, m' et $\bar{b}$.

PHASE 2 :

L'algorithme d'énumération implicite utilisé pour cette phase est une adaptation au cas du problème (S) de l'algorithme général de Martello et Toth [49].

Lorsque la phase dynamique est terminée, l'énumération implicite est appliquée seulement aux n_1 coefficients a_i ($i = 1,\ldots, n_1$) puisque toutes les combinaisons réalisables des a_i ($i = n_1 + 1,\ldots, n$) sont gérées dans la liste L construite initialement :

Supposons être à l'itération k du schéma énumératif :

soit x_I ($I = \{1, 2,\ldots, n_1\}$) une solution partielle, courante de valeur

$$w = a_I\, x_I$$

et R la capacité résiduelle : $R = b - w$ la méthode détermine :

. $j_1^* = \max\{j \mid w_j \leq R \quad j = 0, 1,\ldots, J\}$

. $j_2^* = \max\{j \mid w_j' \leq R \text{ et } x_i\, y_i = 0 \quad i = n_2,\ldots, n_1\}$

où $$w_j' = \sum_{i=n_2}^{n_1} a_i\, y_i$$

$$y_i = 1 \text{ ou } 0 \quad i = n_2,\ldots, n_1$$

En posant : $\Delta = \max\{w_{j_1^*}, w_{j_2^*}\}$

(Δ est le complément maximal de R obtenu à partir des deux listes).

. si $b - R + \Delta = b$ alors le problème (S) est résolu. □

Remarques :

(i) Cette méthode est implémentée avec le classement dans l'ordre décroissant des a_i ($a_1 \geq a_2 \geq \ldots \geq a_n$). Ainsi les listes contiennent des coefficients petits proches les uns des autres, ce qui permet d'avoir des listes en général pas très grandes.

D'autre part, le fait d'appliquer l'énumération implicite sur des données de grande taille implique que la capacité résiduelle R devient petite après quelques itérations ce qui permet d'utiliser les informations des listes plus rapidement.

(ii) Pour les problèmes de grande taille ($n \geq 1000$) et de coefficients a_j relativement petits, il existe de nombreuses solutions atteignant b. Pour ce type de problèmes Martello et Toth proposent d'utiliser leur méthode seulement à une partie du problème (S) appelée "noyau" du problème (fig. 6) (cette idée est donnée par Balas et Zemel [3] dans un algorithme d'énumération implicite).

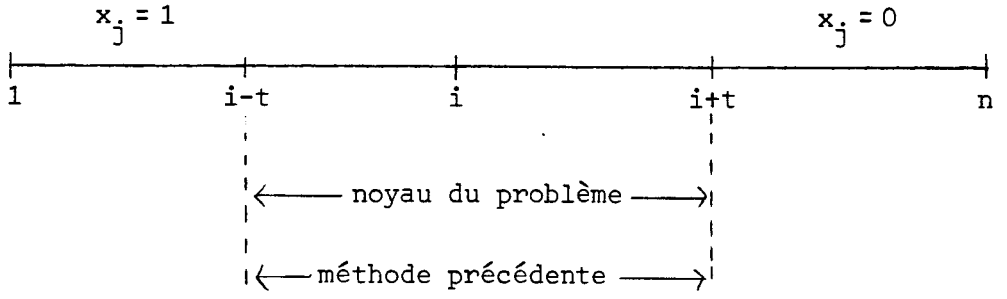


Figure 6

i est définie par :

$$\sum_{j=1}^{i-1} a_j \leq b < \sum_{j=1}^{i} a_j$$

et t est un entier donné ($t \leq 20$).

Il est à noter que si l'optimalité est non prouvée, la méthode peut être éventuellement répétée sur un noyau de taille plus grande.

La méthode de Martello et Toth est efficace pour les problèmes petits et difficiles ($n \leq 40$) ou grands et faciles ($n \geq 1000$).

Elle l'est beaucoup moins au niveau de la résolution des problèmes grands et difficiles ($n \geq 50$). En effet la phase d'énumération implicite, occupe une grande partie du problème d'origine ($n_1 = n - m$ avec $m \leq 24$) contrairement à la phase dynamique limitée à m ($m \leq 24$) donné.

Par conséquent son temps d'exécution peut devenir excessif.

D'où l'idée de donner une nouvelle méthode plus générale, où, la phase dynamique est prépondérante par rapport à celle de l'énumération implicite ; c'est l'objet du paragraphe suivant.

II.2. NOUVELLE METHODE GENERALE : ALGORITHME IDS

II.2.1. Introduction

Les méthodes précédentes sont destinées à des formes particulières du problème (S).

D'où l'idée de donner une méthode générale qui soit au moins aussi efficace que toutes les méthodes précédentes.

Cette méthode cumule les avantages de la programmation dynamique et de l'énumération implicite :

. Pour les problèmes difficiles, il est conseillé d'utiliser la programmation dynamique lorsque la place mémoire le permet.

. Au contraire, pour les problèmes faciles, c'est l'énumération implicite qui est conseillée.

Mais comment reconnaître à priori qu'un problème est facile ou ne l'est pas La question reste ouverte.

La nouvelle méthode est construite dans ce sens : elle tente de trouver un compromis entre la programmation dynamique et l'énumération implicite en donnant la priorité à l'utilisation de la première au sein de laquelle est incorporé un test d'optimalité.

Si le problème est facile alors au bout de quelques itérations l'optimalité est prouvée.

Si ce n'est pas le cas, l'exécution de la programmation dynamique se poursuit jusqu'à épuisement des ressources permises en places mémoire.

Lorsque la phase dynamique est terminée sans preuve de l'optimalité, une phase d'énumération implicite vient se combiner avec la précédente comme dans la méthode de Martello et Toth.

Nous allons montrer en II.2.2. que la programmation dynamique basée sur l'équation récursive (1.20) (donnée en I.2.2.) est de compléxité spatiale O(b), pour trouver à la fois, la valeur et une solution optimale du problème (S) (alors que pour le problème général (B) la compléxité spatiale pour trouver une solution optimale est O(nb)).

Les expériences numériques reportées en (II.3) montrent nettement les bonnes performances de notre méthode par rapport à celle de Martello et Toth. □

II.2.2. Phase hybride

La compléxité spatiale de la programmation dynamique basée sur l'équation récursive :

$$f_k(y) = \max\{f_{k-1}(y),\ a_k + f_{k-1}(y - a_k),\ a_k \le y\}$$
$$k = 1,\ldots, n, \quad y = 0, 1,\ldots, b$$

est O(nb) pour trouver la solution optimale.

Cette compléxité peut être réduite à O(b), en utilisant l'indicateur I_k suivant :

$$I_k(y) = \begin{cases} I_{k-1}(y) & \text{si } f_{k-1}(y) \ge f_{k-1}(y - a_k) + a_k \\ k & \text{sinon} \end{cases} \qquad (2.6)$$
$$k = 1,\ldots, n, \quad y = 0, 1,\ldots, b$$

Le dernier vecteur I_n contient les informations suffisantes pour trouver la solution optimale x^* du problème (S).

Détermination de x^* :

```
y ← f_n(b)
pour k de 1 à n faire x*_k ← 0 fait
Tant que y ≠ 0 faire x_{I_n(y)} ← 1 ; y ← y − a_{I_n(y)} fait
```

Théorème 6 :

La complexité spatiale de la programmation dynamique pour trouver la valeur optimale ainsi qu'une solution optimale du problème (S) *est* O(b).

Démonstration : Evidente en utilisant (2.6).

Exemple :

$n = 4,\ b = 19$

$(a_i) = (8, 11, 7, 8)$

$i = 1, \ldots, 4$

Dans cet exemple, l'indicateur I est géré parallèlement aux valeurs w (w étant la somme d'une partie des a_j $j \in \{1, \ldots, 4\}$). La dernière colonne du tableau ci-dessous suffit pour déterminer la solution optimale x^*.

7							19	2	$\rightarrow x_2^* = 1$
6					19	2	18	3	19 - 11
5					18	3	16	4	
4					15	3	15	3	
3			19	2	11	2	11	2	
2			11	2	8	1	8	1	$\rightarrow x_1^* = 1$
1	8	1	8	1	7	3	7	3	8 - 8
0	0	0	0	0	0	0	0	0	
j	w_j	I_j	w_j	I_j	w_j	I_j	w_j	I_j	
	k = 1		k = 2		k = 3		k = 4		

$v(S) = 19$ et $x^* = (1, 1, 0, 0)$

Le principe de la phase dynamique ci-dessous est le suivant :

Etant donné l'indice i défini par

$$\sum_{j=1}^{i-1} a_j \leq b < \sum_{j=1}^{i} a_j$$

pour k = 1, 2,... deux listes L_1 et L_2 sont construites par la programmation dynamique :

$$L_1 = \{ \sum_{j\in K} a_j \mid \sum_{j\in K} a_j \leq b \text{ et } K \subset \{i-k,\ldots, i-1\}\}$$

$$L_2 = \{ \sum_{j\in K} a_j \mid \sum_{j\in K} a_j \leq b \text{ et } K \subset \{i,\ldots, i+k-1\}\}$$

si $\sum_{j=1}^{i-k-1} a_j + \max\{\omega + \omega' \mid \omega + \omega' \leq b - \sum_{j=1}^{i-k-1} a_j, \omega \in L_1 \text{ et } \omega' \in L_2\}$ est égal à b alors le problème (S) est résolu.

Phase dynamique

1. Trouver i tel que

$$\sum_{j=1}^{i-1} a_j \leq b < \sum_{j=1}^{i} a_j$$

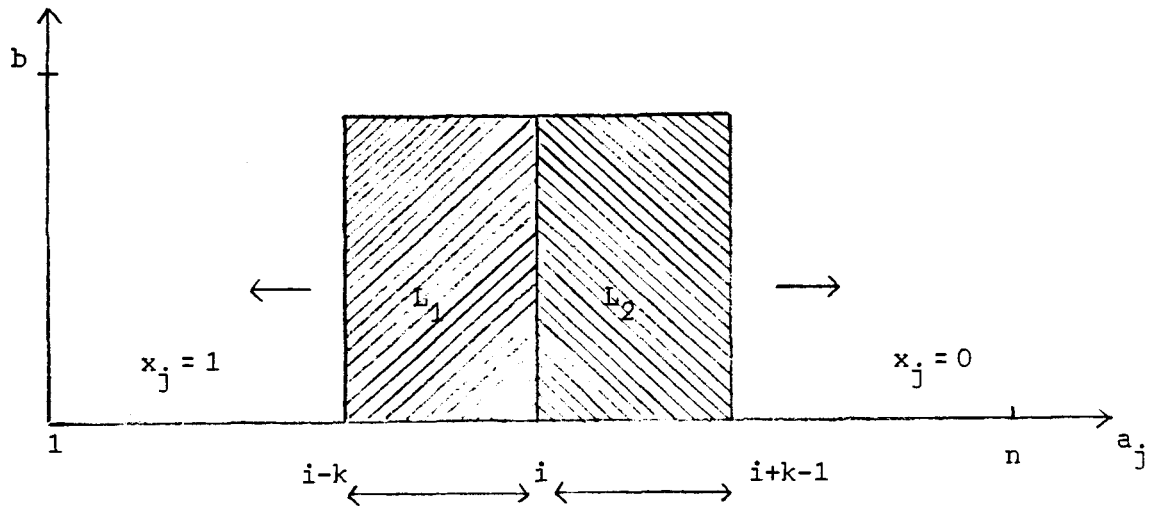
$L_1 \leftarrow \emptyset$; $L_2 \leftarrow \emptyset$; $k \leftarrow 1$ (L_1 et L_2 étant deux listes)

2. Tant que $k \leq n/2$ et la place mémoire est suffisante faire

* Augmenter la liste L_1 (resp. L_2) en ajoutant des combinaisons avec a_{i-k} (resp. a_{i+k-1}) inférieure ou égale à b.

* Fusionner les listes L_1 et L_2 en supposant $x_j = 1$ pour $j = 1,\ldots, i-k-1$; $x_j = 0$ $j = i+k,\ldots, n$ (i.e. avec un second membre égal à

$$b(k) = b - \sum_{j=1}^{i-k-1} a_j)$$



si la valeur fournie par la fusion est égale à b(k) alors
le problème (S) est résolu.
sinon $k \leftarrow k+1$
fsi
fait

II.2.3. Phase de l'énumération implicite

On suppose dans cette phase que le problème n'a pas pû être résolu par la programmation dynamique (i.e. le problème est difficile). On note R le sous-ensemble d'indices (R étant inclus dans {1, 2,..., n}) complémentaires à ceux des données considérées par la programmation dynamique.

L'énumération implicite génère un ensemble de décisions binaires en fixant tour à tour les variables x_i à 0 ou 1, i appartient à R ; les combinaisons réalisables des a_i ($i \in \{1,\ldots, n\}\backslash R$) sont dans les listes finales L_1 et L_2.

Phase énumérative

Pour, chaque x_R^k $k \in \{1,\ldots, 2^{|R|}\}$ de $\mathbb{R}^{|R|}$ généré par l'algorithme d'énumération implicite décrit plus loin (II.2.4), faire
fusionner les listes L_1 et L_2 avec pour second membre $b(k) = b - a_R \cdot x_R^k$
si la valeur fournie par la fusion est égale à b(k) alors le problème (S) est résolu
fsi
fait

II.2.4. Algorithme d'énumération implicite : LIFO

Cet algorithme est une adaptation de la méthode de Fayard et Plateau [18] (p. 134) (donnée dans le cas du problème (B) général) au problème (S).

Cette méthode utilise une technique énumérative dite L.I.F.O. (i.e. Last In First Out) qui consiste à gérer une liste PF d'indices des variables fixées à 0 ou 1 liée au déplacement dans l'arborescence.

Principe

Etant donné un noeud de cette arborescence, c'est-à-dire une partition $\{X_0, X_1, X_2\}$ de $\{1,\ldots, n\}$

$$X_0 = \{j \mid x_j = 0\}$$
$$X_1 = \{j \mid x_j = 1\}$$
X_2 = indices de variables libres

et la particularité du problème (S), les choix de l'évaluation et la séparation en un noeud sont très simples :

1 La séparation s'opère sur la variable du premier indice rencontré $\overset{*}{j} \notin PF$ (les données ne sont pas triées) pour fixer $x_{\overset{*}{j}}$ à 1.

2 L'évaluation par excès $\bar{v}(S)$ en ce noeud est

$$\bar{v}(S) = \min(b, \sum_{j \in X_1 \cup X_2} a_j).$$

La liste PF est gérée de la manière suivante :

- *Descente dans l'arborescence*

(i) Lorsqu'une variable x_k est fixée temporairement à :

. 0 par le test de réduction

$$a_k > b - \sum_{j \in X_1} a_j$$

alors PF devient PF $\cup$ {-k} (X_0 devient $X_0 \cup \{k\}$).

. 1 par le test de réduction

$$a_k + \sum_{j \in X_2} a_j \leq b - \sum_{j \in X_1} a_j$$

alors PF devient PF $\cup$ {-k} (X_1 devient $X_1 \cup \{k\}$).

- *Remontée dans l'arborescence*

Lorsqu'il est montré l'inutilité de résoudre un sous-problème (P) de (S) (soit $\bar{v}(P) \leq Z$ où Z étant la meilleure valeur de (S) rencontré jusqu'à lors), la remontée jusqu'au noeud associé au dernier choix se traduit par la recherche dans PF de l'indice positif le plus à droite pour,

- d'une part, le rendre négatif,
- d'autre part, supprimer tous les indices négatifs situés à sa droite.

II.3. EXPERIENCES NUMERIQUES

L'algorithme IDS présenté en partie II.2 est codé en fortran IV et mis en oeuvre sur CII HB IRIS 80. La performance de l'algorithme IDS est comparée aux algorithmes suivants :

- Algorithme d'Ahrens et Finke [1] donné en II.1.3.
- Algorithme de Martello et Toth [51] donné en II.1.4.
- Algorithme de Balas et Zemel [3]

les problèmes testés sont tirés au hasard dans les intervalles précisés dans les tableaux 1, 2, 3 suivant la loi uniforme à raison de

- 10 problèmes de 16, 20, 24, 28, 32, et 40 variables (tableaux 1 et 2).

- 20 problèmes de 1000, 2500 et 5000 variables (tableau 3). Pour illustrer la performance de l'algorithme IDS par rapport à celle de Martello et Toth [51], nous avons construit un problème difficile (tableau 4) ne possédant qu'une seule solution x telle que ax = b.

Les temps sont donnés en dixièmes de secondes qu'est le pas de la procédure de calcul des temps d'exécution du CII HB IRIS 80.

Notes :

Il est à noter, qu'à l'exception du tableau 4, les résultats fournis pour les méthodes de la littérature sont donnés par des calculateurs (CDC Cyber 730 et CDC 6600) au moins trois fois plus rapides que le CII HB IRIS 80.

Tableau 1

$1 \leq a_j \leq W$; b Wn/4 ; Moyenne (maximum) des temps pour 10 problèmes.

W	n	Ahrens-Finke * [1]	Tree-search * [49]	Martello-Toth* [51]	IDS **
10^3	16	0.40 (0.67)	0.14 (0.21)	0.11 (0.26)	0.
	20	0.69 (1.12)	0.08 (0.16)	0.07 (0.21)	0.
	24	1.37 (2.03)	0.08 (0.14)	0.10 (0.29)	0.
	28	3.49 (5.31)	0.07 (0.18)	0.10 (0.19)	0.
	32	9.40 (19.34)	0.07 (0.12)	0.09 (0.24)	0.
	36	23.41 (36.72)	0.08 (0.22)	0.09 (0.18)	0.
	40	55.90 (93.06)	0.09 (0.21)	0.11 (0.26)	0.
10^6	16	0.91 (1.34)	2.29 (4.73)	0.49 (0.74)	0. (0.)
	20	3.22 (5.43)	21.38 (50.89)	1.85 (2.63)	2.3 (3.)
	24	6.40 (12.82)	101.66 (384.06)	5.13 (9.92)	5.2 (15.)
	28	13.41 (18.44)	115.59 (282.54)	6.47 (10.54)	5.1 (9.)
	32	22.84 (59.23)	73.44 (296.21)	6.61 (15.93)	4.7 (7.)
	36	42.68 (129.17)	55.44 (301.34)	6.05 (17.32)	4.3 (9.)
	40	97.12 (253.46)	55.74 (336.83)	6.63 (23.06)	5.2 (10.)

Tableau 2

$1 \leq a_j \leq 10^3$; $b = 10^3$ n/4+1 ; Moyenne (maximum) des temps pour 10 problèmes

pair impair

n	Ahrens-Finke * [1]	Martello-Toth* [51]	IDS **
16	0.90 (1.46)	0.53 (0.90)	0.6 (1)
20	3.92 (5.13)	1.90 (2.20)	1.9 (2)
24	18.04 (21.87)	5.25 (6.43)	6.0 (7)
28	70.91 (86.42)	9.69 (12.13)	15.6(17)
32	219.61 (333.04)	14.96 (18.61)	23.7(26)
36	-	21.84 (24.37)	38.9(43)
40	-	29.41 (33.86)	52.1(57)

* CDC Cyber 730 ; ** CII HB Iris 80

Tableau 3

$10 \leq a_j \leq W$; $b = 0.5 \sum_{j=1}^{n} a_j$; Moyenne (maximum) des temps pour 20 problèmes.

W	n	Balas-Zemel * [3]	Martello-Toth ** [51]	IDS ***
10^2	1000	0.06 (0.24)	0.09 (0.16)	0.
	2500	0.09 (0.18)	0.16 (0.37)	0.
	5000	0.18 (0.62)	0.25 (0.41)	0.
10^3	1000	0.27 (0.75)	0.25 (0.41)	0.
	2500	0.30 (0.81)	0.32 (0.47)	0.
	5000	0.30 (0.58)	0.34 (0.62)	0.1 (1.)
10^4	1000	2.45 (8.82)	0.83 (1.25)	0.20 (1.)
	2500	1.93 (3.98)	0.92 (1.30)	0.55 (1.)
	5000	2.37 (10.71)	1.00 (1.42)	0.60 (1.)

Tableau 4

$$\forall j \in a_j = \begin{cases} p(j+1)/2, \text{ pour } j \text{ impair} \\ p(j/2)+1, \text{ pour } j \text{ pair} \end{cases} \quad ; \; b = \sum_{j \text{ impair}} a_j$$

Si $p > \lfloor (n+1)/2 \rfloor$ alors ce problème a une solution unique telle que $ax = b$.

n	p	b	Martello et Toth*** [51]	IDS***
56	29	11774	> 2400	40

* CDC 6600 ; ** CDC Cyber 730 ; *** CII HB IRIS 80.

Tableau 4 bis

Les problèmes S_1 et S_2 tirés de la Thèse de Ribeiro [64] sont donnés en détail dans l'annexe.

	n	a_j	b	IDS*
S_1	200	$[1, 2 \times 10^4]$	$0.1 \sum_{j=1}^{n} a_j = 192\,429$	0.1
S_2	200	$[1, 2 \times 10^4]$	$0.5 \sum_{j=1}^{n} a_j = 962\,148$	0.1

* CII HB IRIS 80.

CHAPITRE III

METHODE HYBRIDE POUR LE KNAPSACK 0-1

INTRODUCTION

Le problème du knapsack 0-1

$$(B)\quad \begin{cases} \max \sum_{j=1}^{n} c_j x_j \\ \text{s.c} \sum_{j=1}^{n} a_j x_j \leq b \\ x_j = 0 \text{ ou } 1 \qquad j = 1, \ldots, n \end{cases}$$

où les données vérifient (cf. I.1) :

$$c_j, a_j \in \mathbb{N}^* \; j \in \{1, \ldots, n\} \; ; \; b \in \mathbb{N}^* \; ; \; \max_{1 \leq j \leq n} a_j \leq b < \sum_{j=1}^{n} a_j$$

peut être résolu par deux types d'algorithmes :

- Algorithmes de la programmation dynamique étudiés aux Chapitres I et II.

- Algorithmes d'énumération implicite (branch and bound) : Kolesar [42], Greenberg et Hegerich [27], Horowitz et Sahni [31], Nauss [58], Barr et Ross [5], Zoltners [77], Martello et Toth [49] Fayard et Plateau [19].

La performance des algorithmes du type deux, dépend largement de la nature des données du problème (B). Plus les c_j et les a_j $j \in \{1, 2, \ldots, n\}$ sont corrélées, plus le problème est difficile à résoudre.

Par contre les algorithmes du premier type sont moins affectés par cette corrélation, mais nécessitent un encombrement en place mémoire important (O(nb)) et ne peuvent résoudre dans un temps raisonnable que des problèmes avec un second membre b petit.

D'où l'idée de donner dans ce chapitre, une méthode hybride - notée par ID - combinant, la programmation dynamique et l'énumération implicite, dans le but de résoudre une vaste classe de problèmes (B) que les deux approches considérées séparément.

Cette classe, contiendra aussi bien les problèmes (B) résolubles par l'énumération implicite - représentés par (I) - que ceux résolubles par la programmation dynamique - représentés par (D) - (voir figure 7).

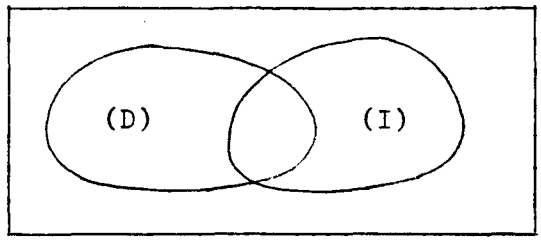


Figure 7

L'algorithme ID décrit en (III.1) fait appel à l'algorithme IDS donné dans le Chapitre II dans le cas où le problème (B) est équivalent à (S) (au sens de l'ensemble des solutions optimales). Les résultats numériques (partie III.3) montrent l'efficacité de l'algorithme ID, en particulier pour le problème du knapsack 0-1 avec une contrainte en égalité (partie III.2).

III.1. DESCRIPTION DE L'ALGORITHME HYBRIDE

L'algorithme ID résoud le problème (B) ($v(B) = cx^*$) en trois phases :

PHASE 1 : RECHERCHE DE PROBLEMES PARTICULIERS

Une analyse simple des données permet de reconnaître les trois cas particuliers suivants :

a) Toutes les composantes de c sont égales :

La résolution de (B) revient à chercher le nombre maximum des plus petits a_j parmi $a_1, a_2, \ldots, a_n$ telle que leur somme soit inférieure ou égale à b. Cette recherche est réalisée par un algorithme de compléxité temporelle linéaire en moyenne (voir Fayard et Plateau [20]).

b) Toutes les composantes de a sont égales :

La résolution de (B) revient à chercher le nombre maximum des plus grands c_j parmi $c_1, c_2, \ldots, c_n$ telle que la somme des a_j correspondants soit inférieure ou égale à b.

Cette recherche est aussi réalisée par un algorithme de compléxité temporelle linéaire en moyenne [20].

c) Tous les rapports c_j/a_j $j = 1, \ldots, n$ sont pratiquement égaux :

Quand les données satisfont la condition du théorème ci-dessous, le problème (B) est résolu par l'algorithme IDS (voir Chapitre II).

Théorème 7

si $\max_{1\leq j\leq n} c_j/a_j < \min_{1\leq j\leq n} c_j/a_j + 1/b$ *alors :*

le problème (B) est équivalent (au sens de l'ensemble des solutions optimales) au suivant :

$$(S)\ \max \sum_{j=1}^{n} a_j x_j \ \text{s.c} \sum_{j=1}^{n} a_j x_j \leq b,\ x_j = 0 \text{ ou } 1 \quad j = 1, \ldots, n.$$

Démonstration

Les problèmes (S) et (B) ont même ensemble de solutions réalisables :

i.e. $F(B) = F(S)$

soit x^* (resp. $\tilde{x}$) une solution optimale de (B) (resp. de (S)) :

i.e. $v(B) = cx^*$ et $v(S) = a\tilde{x}$

Montrons que $cx^* = c\tilde{x}$. Posons $\bar{r} = \max_{1\leq j\leq n} c_j/a_j$ et $\underline{r} = \min_{1\leq j\leq n} c_j/a_j$

$$\left.\begin{array}{l} cx^* = \sum_{j=1}^{n} c_j x_j^* \leq \bar{r} \sum_{j=1}^{n} a_j x_j^* \\ c\tilde{x} = \sum_{j=1}^{n} c_j \tilde{x}_j \geq \underline{r} \sum_{j=1}^{n} a_j \tilde{x}_j \end{array}\right\} \Rightarrow cx^* - c\tilde{x} \leq \bar{r}\, ax^* - \underline{r}\, a\tilde{x}$$

$$\left.\begin{array}{l} cx^* - c\tilde{x} \leq \bar{r}\, ax^* - \underline{r}\, a\tilde{x} \\ v(S) = a\tilde{x} \Rightarrow ax^* \leq a\tilde{x} \end{array}\right\} \Rightarrow \ldots$$

$$\left.\begin{array}{l} cx^* - c\tilde{x} \leq (\bar{r} - \underline{r})\, a\tilde{x} \\ a\tilde{x} \leq b \end{array}\right\} \Rightarrow cx^* - c\tilde{x} \leq (\bar{r} - \underline{r})\, b < 1$$

d'où $cx^* - c\tilde{x} = 0$ □

PHASE 2 : REDUCTION DE LA TAILLE DE (B)

De manière classique, la résolution de $(\bar{B})$ (par des algorithmes de complexité temporelle linéaire [3, 19, 47]) et la détermination d'une borne inférieure $\underline{v}(B)$ de $v(B)$ permettent de fixer à 0 ou à 1 une partie de variables du problème (B) [17, 19, 29, 33, 45].

Soient ainsi :

$$X_\varepsilon = \{j \in \{1,\ldots, n\} \mid x_j \text{ est fixée à } \varepsilon\}, \qquad \varepsilon \in \{0, 1\}$$

$$X_2 = \{1,\ldots, n\} \setminus (X_0 \cup X_1)$$

si (B) est non résolu ($v(B) < \lfloor \underline{v}(\bar{B}) \rfloor$) alors désignons par i l'indice de la variable de base et par

$$d_j = c_j - \frac{c_i}{a_i} a_j \qquad \forall j \in X_2$$

les coûts réduits optimaux du problème $(\bar{B})$.

PHASE 3 : RESOLUTION DU PROBLEME REDUIT

Le principe de la phase 3 de l'algorithme FPK 79 (Fayard et Plateau [19]) est généralisé de la manière suivante : une bipartition des données du problème réduit

$$(RB)\quad \begin{cases} c_{X_1} \cdot e + \max c_{X_2} \cdot x_{X_2} \\ \qquad s.c\ a_{X_2} \cdot x_{X_2} \leq b' = b - a_{X_1} \cdot e \\ \qquad\qquad x_j \in \{0, 1\} \qquad\qquad \forall j \in X_2 \end{cases}$$

est réalisée dans le but de combiner l'énumération implicite et la programmation dynamique :

Etant donnée une bipartition (I, D) de X_2 (voir plus loin a) pour la construction de (I, D)), pour chaque solution $x^k \in V$ ($k = 1,\ldots, 2^{|I|}$) générée par le schéma lexicographique suivant [19] :

. Etant donnée la hiérarchie suivante entre les variables :
$|d_1| \leq |d_2| \leq \ldots |d|_{|I|}$

à partir de x^1 définie par $x_j^1 = \lfloor \bar{x}_j \rfloor$ $\forall j \in I$ ($\bar{x}$ étant la solution optimale du problème $(\bar{B})$), les $2^{|I|} - 1$ autres sommets du cube unité x^j ($j = 2,\ldots, 2^{|I|}$) sont générés dans l'ordre suivant :

$x^2 : (1 - x_1^1, x_2^1, x_3^1,\ldots, x_n^1)$
$x^3 : (x_1^1, 1 - x_2^1, x_3^1,\ldots, x_n^1)$
$x^4 : (1 - x_1^1, 1 - x_2^1, x_3^1,\ldots, x_n^1)$
$x^5 : (x_1^1, x_2^1, 1 - x_3^1,\ldots, x_n^1)$
.
.
.
$x^{2^{|I|}} : (1 - x_1^1, 1 - x_2^1, 1 - x_3^1,\ldots, 1 - x_n^1)$

le problème auxiliaire suivant :

$$(PA^k)\quad \begin{cases} c_{X_1} \cdot e + c_I \cdot x_I^k + \max c_D \cdot x_D \\ \qquad s.c\ a_D \cdot x_D \leq b' - a_I\, x_I^k \\ \qquad\qquad x_j \in \{0, 1\} \quad \forall j \in D \end{cases}$$

est résolu par une procédure de programmation dynamique donnée plus loin en b). □

a) Construction de la bipartition de X_2 :

Cette bipartition est réalisée par une inspection des coûts réduits optimaux du problème $(\bar{B})$ dans le but d'appliquer la programmation dynamique aux données ayant les plus petits $|d_j|$ (les tests d'énumération implicite sont inefficaces pour de telles données) et l'énumération implicite aux autres.

a.1) Cas d'équivalence avec un problème du type (S) :

Lorsque l'ensemble

$J = \{j \in X_2 \mid |d_j| < \max(1/|X_2|, a_j/b')\}$ est égal à X_2, nous montrons dans le théorème ci-dessous que le problème (RB) est équivalent (au sens de l'ensemble des solutions optimales) au suivant

$$(RS) \begin{cases} a_{X_1} \cdot e + a_I \, x_I^k + \max a_D \cdot x_D \\ \text{s.c } a_D \cdot x_D \leq b' - a_I \cdot x_I^k \\ x_j \in \{0, 1\} \quad \forall j \in D \end{cases}$$

$(e_j = 1 \quad \forall j \in X_1)$

qui est résoluble par l'algorithme IDS (Chapitre II).

Lemme 1 : [29]

Soit $x \in F(B)$ *et* $v(\bar{B}) = c\bar{x}$ *alors :*

$$v(\bar{B}) - cx = \sum_{x_j \neq \bar{x}_j} |d_j| + \frac{c_i}{a_i} (b - ax)$$

où i est l'indice de la variable de base.

Théorème 8

Si les données du problème (B) vérifient :

$$|d_j| < \max\left(\frac{1}{n},\ a_j / 2b\right) \forall j \in \{1,\ldots, n\}$$

alors :

(B) a les mêmes solutions optimales que le problème

(S) max ax s.c. $ax \leq b$, $x \in V$.

Démonstration

(1) Montrons que si $|d_j| \leq 1/n \quad \forall j \in \{1,\ldots, n\}$ alors (B) et (S) ont les mêmes solutions optimales.

Soient $v(B) = cx^*$ et $v(S) = a\tilde{x}$

d'après le lemme on a :

$$\left.\begin{aligned} v(\bar{B}) &= cx^* + \sum_{x_j^* \neq \bar{x}_j} |d_j| + \frac{c_i}{a_i}(b - ax^*) \\ v(\bar{B}) &= c\tilde{x} + \sum_{\tilde{x}_j \neq \bar{x}_j} |d_j| + \frac{c_i}{a_i}(b - a\tilde{x}) \end{aligned}\right\} \Rightarrow \ldots$$

$$\left.\begin{aligned} cx^* - c\tilde{x} + \frac{c_i}{a_i}(a\tilde{x} - ax^*) &= \sum_{\tilde{x}_j \neq \bar{x}_j} |d_j| - \sum_{x_j^* \neq \bar{x}_j} |d_j| \leq \sum_{\tilde{x}_j \neq \bar{x}_j} |d_j| \\ cx^* - c\tilde{x} \geq 0,\ a\tilde{x} - ax^* \geq 0 \quad &\text{et } |d_j| \leq \frac{1}{n} \end{aligned}\right\} \Rightarrow$$

$$\left.\begin{aligned} 0 \leq cx^* - c\tilde{x} + \frac{c_i}{a_i}(a\tilde{x} - ax^*) < 1 \\ cx^* - c\tilde{x} \text{ est un entier} \end{aligned}\right\} \Rightarrow cx^* = c\tilde{x}.$$

(2) Montrons que si $|d_j| < a_j/2b \ \forall j \in \{1,\ldots, n\}$ alors (B) et (S) ont les mêmes solutions optimales.

Posons $\bar{r} = \max c_j/a_j = c_{\bar{j}}/a_{\bar{j}}$; $\underline{r} = \min c_j/a_j = c_{\underline{j}}/a_{\underline{j}}$

$$\left.\begin{array}{l} cx^* \le \bar{r} \sum_{j=1}^{n} a_j x^*_j \\ c\tilde{x} \ge \underline{r} \sum_{j=1}^{n} a_j \tilde{x}_j \end{array}\right\} \Rightarrow cx^* - c\tilde{x} \le \bar{r}\, ax^* - \underline{r}\, a\tilde{x} \left.\begin{array}{l} \\ ax^* \le a\tilde{x} \le b \end{array}\right\} \Rightarrow \ldots$$

$$cx^* - c\tilde{x} \le (\bar{r} - \underline{r})\, b \qquad \text{(i)}$$

$$\left.\begin{array}{l} \bar{r} - \underline{r} = c_{\bar{j}}/a_{\bar{j}} - c_i/a_i + c_i/a_i - c_{\underline{j}}/a_{\underline{j}} \\ \forall j \quad d_j = c_j - \frac{a_i}{c_i} a_j \end{array}\right\} \Rightarrow \bar{r} - \underline{r} = d_{\bar{j}}/a_{\bar{j}} - d_{\underline{j}}/a_{\underline{j}} \qquad \text{(ii)}$$

$$\left.\begin{array}{l} \text{(i) et (ii)} \Rightarrow cx^* - c\tilde{x} \le (|d|_{\bar{j}}/a_{\bar{j}} + |d|_{\underline{j}}/a_{\underline{j}})\, b \\ \forall j \ |d_j| < a_j/2b \end{array}\right\} \Rightarrow$$

$$\left.\begin{array}{l} cx^* - c\tilde{x} < 1 \\ \text{or } cx^* - cx^* \ge 0 \text{ et entier} \end{array}\right\} \Rightarrow cx^* = c\tilde{x}.$$

□

a.2) Construction effective de la bipartition

Lorsque l'ensemble J ci-dessus n'est pas égal à X_2, l'ensemble D est composé par l'union de J et de l'ensemble

$$J' = \{j \in X_2 \backslash J \mid |d_j| < \mu(v(\bar{B}) - \underline{v}(B))\}$$

où μ est un paramètre donné expérimentalement au voisinage de 0.8 et $\underline{v}(B)$ est une borne inférieure de v(B), améliorée pendant la construction itérative de l'ensemble J' par la procédure suivante basée sur la programmation dynamique.

Procédure dynamique

But : Construire (I, D) et préparer la résolution des problèmes auxiliaires (voir b)).

% Initialisation %

$D \leftarrow J$; $J' \leftarrow \emptyset$; $I \leftarrow X_2 \backslash D$; fin $\leftarrow$ faux

Tant que $|D| <$ seuil (seuil étant la taille maximale fixée pour la partie dynamique) et non fin faire

1. Appliquer la programmation dynamique au problème

$$(PA) \begin{cases} \max c_D \cdot x_D \\ a_D \cdot x_D \leq b' \\ x_j \in \{0, 1\} \quad \forall j \in D \end{cases}$$

pour construire une ou deux listes (suivant la taille de D (voir b))) constituées par des éléments ou des paires du type

$$(v, w) = \left(\sum_{j \in K} c_j, \sum_{j \in K} a_j \right)$$

où $K \subset D$ et $\sum_{j \in K} a_j \leq b'$

vérifiant la propriété de non dominance :

i.e. étant donnée une liste L,

$\forall (v, w) \in L, \nexists (v', w') \in L : v' \geq v$ et $w' \leq w$.

2. Rechercher la meilleure solution sous l'hypothèse
$x_j = \bar{x}_j \quad \forall j \in I$ (i.e. pour un second membre égal à $b' - \sum_{j \in I} a_j \bar{x}_j$)

où $\bar{x}$ est la solution optimale de $(\bar{B})$.

(dans le cas de deux listes, cette recherche est réalisée par leur fusion)

Soit x_D la solution obtenue, alors :

$$\underline{v}(B) = c_{X_1} \cdot e + c_I \bar{x}_I + c_D \cdot x_D$$

3 Soit $J' = \{j \in I \mid |d_j| < \mu(v(\bar{B}) - \underline{v}(B))\}$

si $J' \neq \emptyset$ alors
$D \leftarrow D \cup \{\min(5, |J'|)$ plus petits $|d_j|$ $j \in J'\}$; $I \leftarrow X_2 \backslash D$
sinon fin ← vrai
fsi

fait □

b) Résolution des problèmes auxiliaires :

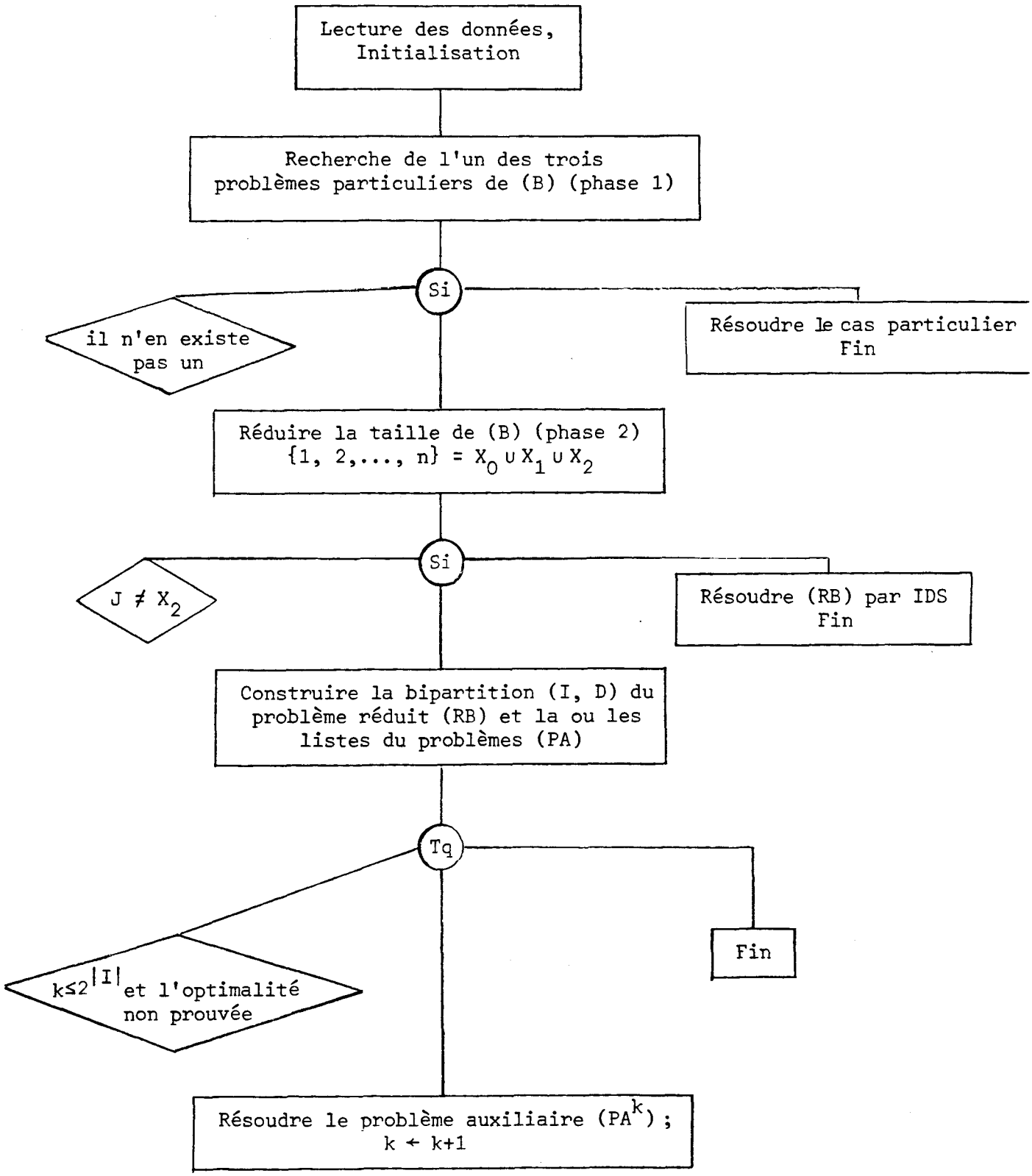
La résolution des problèmes auxiliaires (PA^k) $k = 1, 2, \ldots$, est préparée dans la phase 1 de la procédure dynamique précédente (voir a)).

Pour chaque problème auxiliaire (PA^k) $k = 1, 2, \ldots$, sa résolution est réduite à une simple exploration des ℓ ($\ell = 1$ ou 2) listes construites pendant la détermination de la bipartition (I, D) :

(i) $\ell = 1$: la valeur du problème (PA^k) s'obtient facilement par une recherche dichotomique sur l'unique liste L ; de compléxité temporelle $O(\min(|D|, \log_2 b')$.

(ii) $\ell = 2$: la valeur du problème (PA^k) est atteinte par la procédure de fusion (voir Chapitre I.2.5) des deux listes L_1 et L_2 ; avec une compléxité temporelle $O(\min(2^{|D|/2}, b'))$; les listes L_1 et L_2 étant associées chacune à la moitié des données du problème (PA). □

Arbre d'analyse descendante de ID.



III.2. LE PROBLEME DU KNAPSACK 0-1 AVEC UNE CONTRAINTE D'EGALITE

III.2.1. Introduction

Le problème du knapsack 0-1 avec une contrainte en égalité peut se formuler ainsi :

$$(E)\ \begin{cases} \max\ cx \\ s.c\ ax = b \\ \qquad x \in V \end{cases}$$

Ce problème peut être rencontré dans de nombreux problèmes, comme celui

- de "change monaitaire" ; problème connu sous le nom de "the change-making problem" (voir Chang et Gill [13], Wright [76]),

- du plus court chemin (voir Ribeiro [64]),

- de cryptographie et de signature de messages codés (voir [30, 34, 44, 52]

- de contraction de contraintes (voir Plateau et Guerch [61]).

Le problème (E) appartient également à la classe des problèmes NP-complets (voir Garey et Johnson [20 bis]) mais sa compléxité expérimentale est encore plus grande que celle du knapsack 0-1 avec une contrainte en inégalité (B) : l'obtention d'une borne inférieure $\underline{v}(E)$ (i.e. la recherche d'une solution réalisable) est en soit un problème NP-complet.

Le problème (E) peut être résolu par des algorithmes de programmation dynam (voir Chapitre I), d'énumération implicite adapté à ce problème et de plus courts chemin (voir Ribeiro [64]).

Cependant, ces algorithmes sont inefficaces pour les exemplaires du problème (E) les plus difficiles ; c'est-à-dire lorsque (E) possède peu de solutions réalisables.

Nous proposons dans cette partie de transformer le problème (E) en un problème équivalent - au sens de l'ensemble des solutions optimales - avec une contrainte en inégalité (section III.2.2), dans le but de résoudre celui-ci par l'algorithme ID décrit en (III.1).

Cette transformation a permis de résoudre effectivement des problèmes difficiles du type (E) par ID (section III.3, tableau 6). □

III.2.2. Problème équivalent avec une contrainte en inégalité

Une solution optimale $\tilde{x}$ pour le problème (E) peut être trouvée en résolvant le problème du knapsack 0-1 suivant avec une contrainte en inégalité :

$$(E(u)) \quad \begin{cases} \max\ cx + u\,ax \\ ax \leq b \\ x \in V \end{cases}$$

où le paramètre $u \in \mathbb{R}^+$ satisfait la condition du théorème ci-dessous.

Lemme 2

Etant donné $u \in \mathbb{R}^+$, *soit* $\tilde{x}(u)$ *une solution optimale du problème* (E(u)) :

Si $a\tilde{x}(u) = b$ *alors* $\tilde{x}(u)$ *est une solution optimale du problème* (E) (*i.e.* $v(E) = c\tilde{x}(u)$).

Démonstration

Soit $v(E) = c\tilde{x}$ ($\tilde{x}$ étant une solution optimale de (E)) $\tilde{x}(u)$ solution optimale de (E(u)) $\Rightarrow c\tilde{x} + u\,a\tilde{x} \leq c\tilde{x}(u) + u\,a\tilde{x}(u)$

$$\left.\begin{array}{l} \ldots \Rightarrow u(a\tilde{x} - a\tilde{x}(u)) \leq c\tilde{x}(u) - c\tilde{x} \\ u \in \mathbb{R}^+,\ a\tilde{x} = b = a\tilde{x}(u) \text{ et } c\tilde{x} \geq c\tilde{x}(u) \end{array}\right\} \Rightarrow c\tilde{x}(u) = c\tilde{x}$$ □

Théorème 9

$$Si\ u > u_0 = \max\left\{\frac{cx - ax}{b - ax} \mid ax < b,\ x \in V\right\}\ alors :$$

(i) *Toute solution optimale* $\tilde{x}(u)$ *de* (E(u)) *est optimale pour* (E).

(ii) *Si* $a\tilde{x}(u) < b$ *alors* $F(E) = \emptyset$.

Démonstration

On suppose que $F(E) \neq \emptyset$ soient $v(E(u)) = c\tilde{x}(u) + u\,\tilde{x}(u)$ et $v(E) = c\tilde{x}$ $\tilde{x}(u)$ solution optimale de (E(u)) $\Rightarrow c\tilde{x} + u\,a\tilde{x} \leq c\tilde{x}(u) + u\,a\tilde{x}(u)$

$$\left.\begin{array}{l} \text{soit } u(a\tilde{x} - a\tilde{x}(u)) \leq c\tilde{x}(u) - c\tilde{x} \\ a\tilde{x} = b \end{array}\right\} \Rightarrow \ldots$$

$$\ldots \text{ (1) } \left.\begin{array}{l} u(b - a\tilde{x}(u)) \leq c\tilde{x}(u) - c\tilde{x} \\ u > u_0 \end{array}\right\} \Rightarrow b = a\tilde{x}(u) \Rightarrow c\tilde{x}(u) = c\tilde{x}$$

d'où (i)

(1) et $b > a\tilde{x}(u)$ est impossible, d'où (ii). □

Notes :

- Une borne inférieure triviale sur u est

$$\lfloor v(\bar{B}) \rfloor - \lceil \min\{cx \mid ax \geq b\ ;\ x \in [V]\} \rceil .$$

- Les valeurs expérimentales de u aboutissant à l'équivalence sont petites (voir les résultats de la section III.3, tableau 6).

- Les données du problème (E(u)) vérifient trivialement les propriétés suivantes :

a) si $c_1/a_1 \geq c_2/a_2 \geq \ldots \geq c_n/a_n$ alors $\frac{c_1 + u\, a_1}{a_1} \geq \frac{c_2 + u\, a_2}{a_2} \geq \ldots \geq \frac{c_n + u\, a_n}{a_n}$

b) si on note par $d_j(u)$ $j \in \{1, \ldots, n\}$ les coûts réduits optimaux du problème (E(u)) alors

$$d_j(u) = d_j \qquad j = 1, \ldots, n$$

(où d_j désignent les coûts réduits optimaux du problème (B)).

c) $v(\overline{E(u)}) = v(\overline{B}) + u\, b$; $v(E) = v(E(u)) - u\, b$. □

III.3. EXPERIENCES NUMERIQUES

L'algorithme ID présenté en III.1. est codé en fortran IV et mis en oeuvre sur le calculateur CII HB IRIS 80. Tous les résultats sont donnés en dixièmes de secondes. Ils montrent bien l'efficacité de cet algorithme, aussi bien pour le problème du knapsack 0-1 avec une contrainte en inégalité qu'avec une contrainte en égalité.

III.3.1. Problèmes avec une contrainte d'inégalité

L'algorithme ID est comparé à deux algorithmes performants :

- FPK 79 : algorithme d'énumération implicite pure (donné par Fayard et Plateau [19]).

- DPT 3 : algorithme de programmation dynamique pure (donné par Toth [70]).

Les données des problèmes tests sont tirées au hasard dans les intervalles spécifiés (tableau 5) suivant la loi uniforme. L'analyse des résultats du tableau 5 dégagent les conclusions suivantes :

- L'énumération implicite est affectée par la corrélation des données.

- La programmation dynamique est affectée par la taille du problème (c'est-à-dire nb grand).

- L'algorithme ID est général puisque il résoud des problèmes où les autres méthodes ont échouées et la taille de la partie dynamique ([D]) augmente avec la difficulté du problème.

Tableau 5

Min (resp. max) = Moyenne des 10 plus petits (resp. plus grands) temps parmi 40.

(Av) = Moyenne des temps de 40 problèmes.

Avt (resp. maxt) = Moyenne (resp. maximum) des temps de 200 problèmes.

[D] : Moyenne des tailles des 40 problèmes auxiliaires.

a_j	c_j	b	n	FPK 79 * Min (Av) Max	ID * [D] Min (Av) Max	Toth ** (Avt) Maxt
données non corrélées						
$[1,10^3]$	$[1,10^3]$	$0.5 \sum_{j=1}^{n} a_j$	1000	2.4 (4.85) 9.5	6 3.0 (4.73)8.1	-
			5000	13.3 (19.3) 27.5	12 16.2 (18.3)22.4	-
données peu corrélées						
$[1,10^3]$	$[a_j-10^2, a_j+10^2]$	$0.5 \sum_{j=1}^{n} a_j$	1000	3.4 (11.55)25.3	9 3.7 (10.85)24.8	-
			5000	14.8 (21.65)35.9	10 17.5 (24.) 34.1	-
données corrélées						
$[1,10^2]$	$c_j = a_j+10$	$0.5 \sum_{j=1}^{n} a_j$	50	18'	21 0.2 (10.) 28.1	(1.8) 3.7
			100	3'	48 4.6 (27.) 42.4	(7.) 15.2
			200	2'	51 6.8 (29.8)40.6	(19.1) 50.
$[1,10^2]$	$c_j = a_j+10$	$0.8 \sum_{j=1}^{n} a_j$	50	17'	20 0.4 (7.2)18.2	(0.7) 1.9
			100	4'	40 1. (19.7)42.3	(2.7) 6.8
			200	0'	50 2.9 (29.3)41.5	(10.5) 26.9
données fortement corrélées						
$[1,10^2]$	$c_j = a_j+1$	$0.5 \sum_{j=1}^{n} a_j$	1000	-	- 2.7 (3.9) 5.5	-
			5000	-	- 18.5 (23.2)33.5	-

* CII HB IRIS 80 ; ** CDD 6600

i' = signifie le nombre de problèmes résolus en moins de 4 minutes.

III.3.2. Problèmes avec une contrainte d'égalité

Dans cette partie, il s'agit de la résolution de problèmes en minimisation

$$(P)\quad \begin{cases} \min\ cx \\ s.c\ ax = b \\ \qquad x \in V \end{cases}$$

par les algorithmes ID et FPK 79, après les avoir transformés en problèmes équivalents de maximisation :

$$(E)\quad \begin{cases} c \cdot e + \max\ cx' \\ \qquad s.c\ ax' = a \cdot e - b \\ \qquad\qquad x' = e - x,\ x \in V \end{cases}$$

les problèmes (E) ainsi obtenus sont transformés aux problèmes du type (E(u)) (voir III.2).

Les problèmes (P) sont décrits dans le tableau 6 où les données sont tirées au hasard suivant une distribution uniforme dans les intervalles spécifiés (le détail de chaque problème ainsi qu'une solution optimale correspondante sont donnés dans l'annexe).

Il est à noté que pour les problèmes difficiles :

- P_2, P_3 et P_4 la valeur v(P) est éloignée de la valeur $v(\overline{P})$ ($(\overline{P})$ étant le problème en continu).

- P_4 et P_5, ID est 5 fois plus rapide que FPK 79.

Tableau 6 [64]

Problem	n	a_j	c_j	b
P.1	100	$[1, 10^4]$	[1, 100]	49802 ≃ 0.1 Σa_j
P.2	100	$[1, 10^5]$	[1, 100]	502968 ≃ 0.1 Σa_j
P.3	100	$[10^4, 10^5]$	[1, 100]	507884 ≃ 0.1 Σa_j
P.4	100	$[1, 10^6]$	[1, 100]	4578665 ≃ 0.1 Σa_j
P.5	100	$[10^5, 10^6]$	[1, 100]	4909599 ≃ 0.1 Σa_j
P.6	200	$[1, 2\times10^4]$	[1, 100]	192429 = 0.1 Σa_j
P.7	200	$[1, 2\times10^4]$	[1, 100]	962148 = 0.5 Σa_j

r = Le nombre total de variables fixées à la fin des méthodes.

Problemes	$v(\bar{P})$	$v(P)$	u	r	|D|	ID *	FPK79 *
P.1	21.45	42	10	73	20	16.	53.
P.2	29.86	100	10	27	20	1395.	1030.
P.3	51.67	79	10	65	20	56.	58.
P.4	46.06	102	50	35	20	2710.	13440.
P.5	48.98	112	10	38	20	566.	3007.
P.6	68.02	77	10	179	20	31.	57.
P.7	1927.26	1936	10	164	20	75.	57.

* CII HB IRIS 80.

CHAPITRE IV

ROTATION DE CONTRAINTES

INTRODUCTION

Etant donné un problème (B) en nombres entiers, il existe plusieurs méthodes d'obtention de formulations équivalentes à celle de (B) avec exactement le même ensemble de solutions réalisables ; par exemple :

. La contraction de contraintes

(Anthonisse [2], Bradley [10, 11], Chvatal [14], Glover [25], Glover et Woolsey [26], Kaliszewski et Libura [36], Kannau [37], Kendall et Zionts [39], Mayer [53], Plateau et Guerch [61], Rosenberg [65], Padberg [60]).

. La réduction de coefficients de contraintes

(Bradley, Hammer et Wolsey [12], Wilson [74], Kostreva et Johnson [43]).

La rotation de contraintes (Kianfar [40, 41], Walukiewcz et Kalisezewsi [71]) rentre dans ce cadre de méthodes. Elle consiste à déplacer les contraintes - soit celles du problème (B), soit les coupes générées en cours des méthodes algébriques de résolution de (B) - de sorte que les nouvelles contraintes obtenues passent par au moins autant de points entiers que les originales.

Les résultats rapportés par Beale et Tomlin [6], Crowder, Johnson et Padberg [28] Geoffrion et Gravaves [22], Williams [73] et Walukiewicz et Kaliszewski [71] indiquent que le problème (B) est d'autant plus facile à résoudre que le domaine de $(\overline{B})$ est bien réduit.

Après un rappel de la formulation de la rotation de contraintes et de son interprétation géométrique (Section IV.2) nous introduisons celle avec une contrainte additionnelle (Section IV.3), celle-ci permet de réduire davantage le domaine de $(\overline{B})$. Cette idée a soulevé le choix de la contrainte additionnelle (IV.4) ainsi que celui de la contrainte éliminant une contrainte donnée (réduction de contraintes).

En plus des applications classiques de la rotation de contraintes, nous montrerons (IV.4.2) comment celle-ci peut contribuer aux méthodes d'élimination de contraintes.

Au paragraphe IV.5.3 nous donnons un nouvel algorithme, calculant les coefficients de la contrainte obtenue après rotation d'une contrainte de n variables et de second membre b, de compléxité temporelle O(nb). Cet algorithme améliore considérablement celui de Kianfar [40] dont la compléxité temporelle est $O(n^2b)$ (Section IV.5.2). Une comparaison théorique et expérimentale entre ces deux algorithmes est présentée en section IV.5.4.

IV.1. FORMULATION DU PROBLEME

On suppose, sans perdre la généralité [40], que la rotation est appliquée à des contraintes d'un programme linéaire suivant à m contraintes et n variables bivalentes :

$$(B) \quad \begin{cases} \max\ cx \\ \text{s.c } Ax \leq b \\ \quad x \in V \end{cases}$$

(dont les coefficients c_j de c, b_i de b et a_{ij} de A ($i = 1,\ldots, m$, $j = 1,\ldots, n$) sont supposés être des entiers non négatifs).

L'étude de la rotation de contraintes aura pour support l'utilisation des problèmes de knapsack extraits de (B) notés comme suit, pour $p, q \in \{1,\ldots, m\}$ donnés :

$$(B_p^0) \quad \begin{cases} \max\ cx \\ \text{s.c } A_p\, x \leq b_p \\ \quad x \in V \end{cases} \quad ; \qquad (B_q^p) \quad \begin{cases} \max\ A_p\, x \\ \text{s.c } A_q\, x \leq b_q \\ \quad x \in V \end{cases}$$

$$(\hat{B}_p^0) \quad \begin{cases} \max\ cx \\ \hat{A}_p\, x \leq b_p \\ \quad x \in V \end{cases} \quad ; \quad (B_{p,q}^p) \quad \begin{cases} \max\ A_p\, x \\ \text{s.c } A_p\, x \leq b_p \\ A_q\, x \leq b_q \\ \quad x \in V \end{cases}$$

où la contrainte $\hat{A}_p\, x \leq b_p$ est obtenue par rotation de la contrainte $A_p\, x \leq b_p$.

IV.2. ROTATION SANS CONTRAINTE ADDITIONNELLE : KIANFAR [40]

IV.2.1. Formulation

La rotation géométrique de la contrainte

$$A_p\, x \leq b_p \qquad p \in \{1, \ldots, m\}$$

consiste à déplacer l'hyperplan

$$A_p\, x = b_p$$

de sorte que le nouvel hyperplan

$$\hat{A}_p\, x = b_p$$

passe par au moins autant de points entiers que l'original c'est-à-dire

$$H_p = \{x \in V \mid A_p\, x = b_p\} \subset \hat{H}_p = \{x \in V \mid \hat{A}_p\, x = b_p\}$$

Les coefficients $\hat{a}_{pj}$ de $\hat{A}_p$ $(j = 1, \ldots, n)$ sont calculés par le procédé itératif suivant :

Etant donné $r \in \{1, 2, \ldots, n\}$, considérons le problème

$$(B_p^p \mid x_r = 1)\ a_{pr} + \max \sum_{\substack{j=1 \\ j\neq r}}^{n} a_{pj}\, x_j \quad \text{s.c} \sum_{\substack{j=1 \\ j\neq r}}^{n} a_{pj}\, x_j \leq b_p - a_{pr}\ ;\ x_j = 0 \text{ ou } 1 \quad \forall j \neq r$$

en posant

$$\hat{a}_{pr} = a_{pr} + b_p - v(B_p^p \mid x_r = 1).$$

Le procédé est alors réitéré sur un autre coefficient de la nouvelle contrainte

$$\hat{a}_{pr}\, x_r + \sum_{\substack{j=1 \\ j\neq r}}^{n} a_{pj}\, x_j \leq b_p.$$

Si $\hat{a}_{pr} > a_{pr}$ alors le nouvel hyperplan

$$\hat{a}_{pr}\, x_r + \sum_{\substack{j=1 \\ j\neq r}}^{n} a_{pj}\, x_j = b_p$$

passe par le nouveau point entier $\hat{x}$ défini par :

$$\hat{x}_r = 1 \text{ et } \hat{x}_j = x_j^* \quad j = 1,\ldots, r-1, r+1,\ldots, n$$

(x^* étant une solution optimale du problème $(B_p^p \mid x_r = 1)$). Ce qui n'était pas le cas pour l'hyperplan

$$\sum_{j=1}^{n} a_{pj}\, x_j = b_p.$$

Exemple : Considérons la contrainte

$$2x_1 + 3x_2 \leq 4$$
$$x_1 \; ; \; x_2 = 0 \text{ ou } 1$$

les solutions bivalentes de $2x_1 + 3x_2 \leq 4$ sont (0, 0), (1, 0) et (0, 1) ; l'application du principe exposé ci-dessus permet, en deux rotations successives, d'aboutir à la contrainte $4x_1 + 4x_2 \leq 4$ dont l'ensemble des solution bivalentes est identique à la précédente, mais qui s'appuie sur les deux points entiers (1, 0) et (0, 1) alors que la contrainte d'origine ne s'appuyait sur aucun de ces deux points :

$$\max\{2x_1 \mid 2x_1 \leq 4 - 3 \; ; \; x_1 = 0 \text{ ou } 1\} = 0 \Rightarrow \hat{a}_2 = 4$$

$\Rightarrow$ la contrainte initiale est transformée en $2x_1 + 4x_2 \leq 4$.

$$\max\{4x_2 \mid 4x_2 \leq 4 - 2 \; ; \; x_2 = 0 \text{ ou } 1\} = 0 \Rightarrow \hat{a}_1 = 4$$

d'où la contrainte finale :

$$4x_1 + 4x_2 \leq 4$$ (voir figure 8).

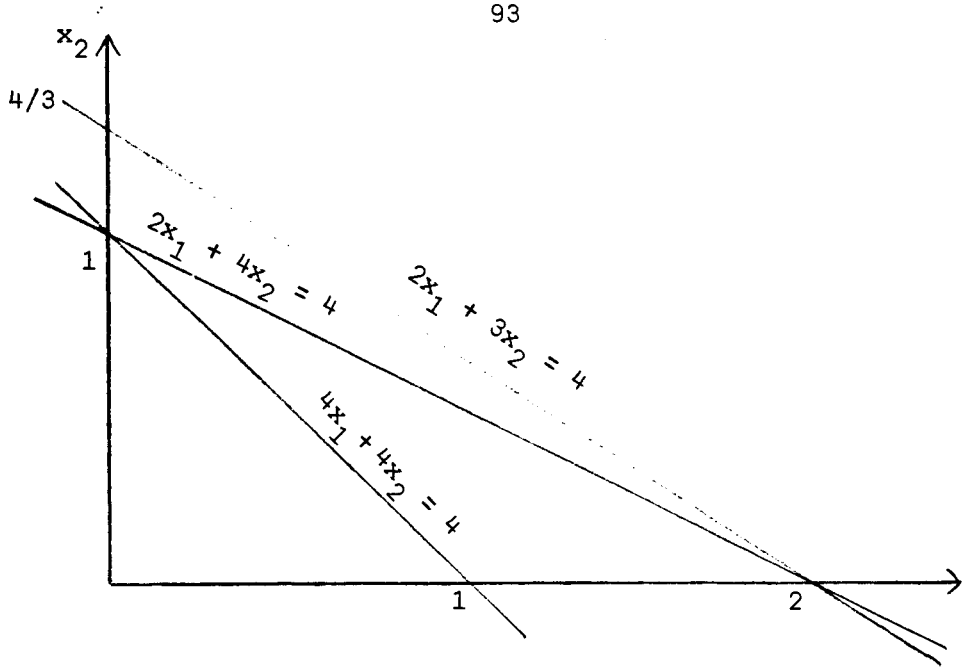


Figure 8

Les coefficients $\hat{a}_{pj}$ (j = 1,..., n) se calculent de manière récursive par l'algorithme suivant :

Algorithme :

$I \leftarrow \{1, 2, \ldots, n\}$

Tant que $I \neq 0$ *faire*

choisir $r \in I$

$a_{pr} \leftarrow a_{pr} + b_p - v(B_p^p \mid x_r = 1)$

$I \leftarrow I - \{r\}$

fait

Théorème 10 :

Etant donné $p \in \{1, \ldots, m\}$, *en notant* $\hat{A}_p\, x \leq b_p$ *la contrainte obtenue par rotation de la contrainte* $A_p\, x \leq b_p$ *et*

$$R = \{j \in \{1, \ldots, n\} \mid \hat{a}_{pj} > a_{pj}\}$$

alors :

1° $R \neq \emptyset \Rightarrow F(\bar{\bar{B}}_p^0) \subsetneq (\bar{B}_p^0)$

2° $\mathrm{Card}(\hat{H}_p) = \mathrm{Card}(H_p) + \mathrm{Card}(R)$.

Démonstration

1) Puisque $\forall j \in \{1,\ldots, n\}\ \hat{a}_{pj} \geq a_{pj}$ alors $F(\tilde{B}_p^0) \subseteq F(\bar{B}_p^0)$ montrons que cette inclusion est stricte si $R \neq \emptyset$.

Pour tout $r \in \{1,\ldots, n\}$, on rappelle que

$$\hat{a}_{pr} = a_{pr} + b - v(B_p^p \mid x_r = 1) \qquad (i)$$

Soit x^* une solution optimale de $(B_p^p \mid x_r = 1)$ dont la composante d'indice $\ell \in \{1,\ldots, n\}$ est nulle (i.e. $x_\ell^* = 0$). En définissant $\underline{x}$ par

$$\underline{x}_j = \begin{cases} x_j^* & \forall j \neq \ell \\ (b_p - v(B_p^p \mid x_r = 1))/a_{p\ell} & j = \ell \end{cases}$$

nous allons montrer que $\underline{x} \in F(\bar{B}_p^0)$ et $\underline{x} \notin F(\tilde{B}_p^0)$:

* Montrons que $\underline{x}_\ell \leq 1$:

$$\left.\begin{array}{l} x_\ell^* = 0 \\ v(B_p^p \mid x_r = 1) \leq b \end{array}\right\} \Rightarrow v(B_p^p \mid x_r = 1) \leq b < v(B_p^p \mid x_r = 1) + a_{p\ell} \left.\begin{array}{l} \\ \\ \end{array}\right\} \Rightarrow 0 \leq \underline{x}_\ell < 1$$

$$\text{or} \quad \underline{x}_\ell = (b_p - v(B_p^p \mid x_r = 1)/a_{p\ell}$$

* Montrons que $\underline{x} \in F(\bar{B}_p^0)$ (i.e. $A_p\, \underline{x} \leq b_p$) si $r \in R$:

$$A_p\, \underline{x} = \sum_{\substack{j=1 \\ j\neq\ell}}^{n} a_{pj}\, \underline{x}_j + a_{p\ell}\, (b_p - v(B_p^p \mid x_r = 1))/a_{p\ell}$$

$$= \left.\begin{array}{l} \displaystyle\sum_{\substack{j=1 \\ j\neq\ell}}^{n} a_{pj}\, \underline{x}_j + b_p - v(B_p^p \mid x_r = 1) \\ \\ \text{or } \underline{x}_j = x_j^* \quad \forall j \neq \ell \quad \text{et } x_\ell^* = 0 \end{array}\right\} \Rightarrow \ldots$$

$$\ldots\ A_p\, \underline{x} = v(B_p^p \mid x_r = 1) + b_p - v(B_p^p \mid x_r = 1) = b_p.$$

* Montrons que $\underline{x} \notin F(\bar{B}^0_p)$ (i.e. $\hat{A}_p \underline{x} > b_p$) :

$$\left.\begin{array}{l} \hat{A}_p \underline{x} = \sum_{\substack{j=1 \\ j\neq r \text{ et } \ell}}^{n} \hat{a}_{pj} \underline{x}_j + \hat{a}_{pr} \underline{x}_r + \hat{a}_{p\ell} \underline{x}_\ell \\ \\ (i) \Rightarrow \hat{a}_{pj} \geq a_{pj} \quad \forall j \end{array}\right\} \Rightarrow \ldots$$

$$\ldots \hat{A}_p \underline{x} \geq \sum_{\substack{j=1 \\ j\neq r \text{ et } \ell}}^{n} a_{pj} \underline{x}_j + \hat{a}_{pr} \underline{x}_r + a_{p\ell} \underline{x}_\ell \Bigg\} \Rightarrow \ldots$$

$$\text{or } \underline{x}_\ell = (b_p - v(B^p_p \mid x_r = 1))/a_{p\ell}$$

$$\ldots \hat{A}_p \underline{x} \geq \sum_{\substack{j=1 \\ j\neq r \text{ et } \ell}}^{n} a_{pj} \underline{x}_j + \hat{a}_{pr} \underline{x}_r + b_p - v(B^p_p \mid x_r = 1) \Bigg\} \Rightarrow \ldots$$

$$\text{or } \underline{x}_j = x^*_j \ \forall j \neq r \text{ et } \ell \ ; \ \underline{x}_r = x_r = 1 \text{ et } \hat{a}_{pr} = a_{pr} + b_p - v(B^p_p \mid x_r = 1)$$

$$\ldots \hat{A}_p \underline{x} \geq \sum_{\substack{j=1 \\ j\neq r \text{ et } \ell}}^{n} a_{pj} x^*_j + a_{pr} + b_p - v(B^p_p \mid x_r = 1) + b_p - v(B^p_p \mid x_r = 1)$$

puisque $\sum_{\substack{j=1 \\ j\neq r \text{ et } \ell}}^{n} a_{pj} x^*_j + a_{pr} = v(B^p_p \mid x_r = 1)$ on a :

$$\left.\begin{array}{l} \hat{A}_p \underline{x} \geq b_p + b_p - v(B^p_p \mid x_r = 1) \\ \\ r \in R \text{ et } (i) \Rightarrow b_p - v(B^p_p \mid x_r = 1) > 0 \end{array}\right\} \Rightarrow \hat{A}_p \underline{x} > b_p.$$

2) Il suffit de remarquer que $\hat{H}_p = H_p \cup R_p$ et $H_p \cap R_p = \emptyset$

où $R_p = \{x^r \mid r \in R \text{ et } x^r \text{ solution optimale de } (B^p_p \mid x_r = 1)\}$. □

IV.2.2. Exemple et interprétation géométrique

Considérons la contrainte

$$3x_1 + 5x_2 + 6x_3 \leq 11$$

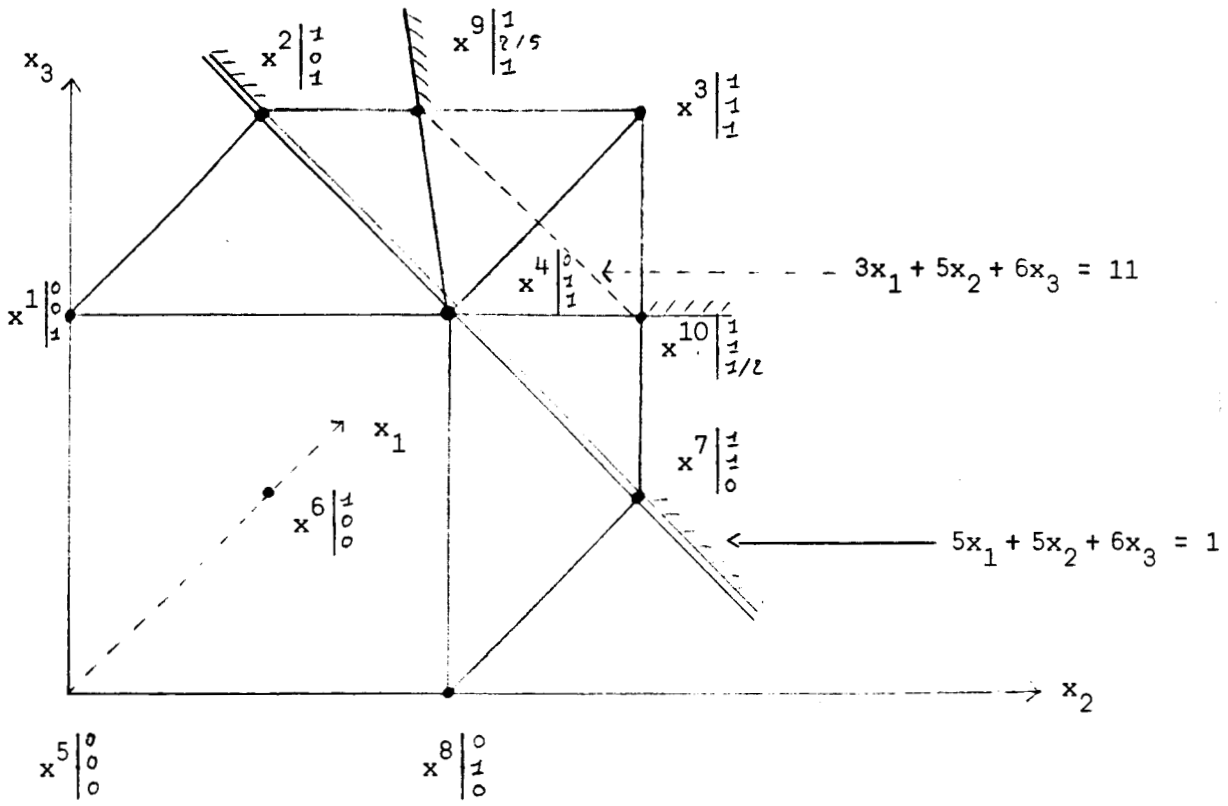


Figure 9

Une rotation permet d'aboutir à la contrainte

$$5x_1 + 5x_2 + 6x_3 \leq 11.$$

Avant la rotation, l'hyperplan

$$3x_1 + 5x_2 + 6x_3 = 11$$

s'appuie sur les points x^4, x^9 et w^{10} ; les points x^1, x^2, x^5, x^6, x^7, x^8 sont strictement en dessous et x^3 est au dessus de celui-ci.

Après rotation le nouvel hyperplan

$$5x_1 + 5x_2 + 6x_3 = 11$$

vient s'appuyer sur les points x^2 et x^7 tout en restant appuyé sur x^4 et éliminant les points x^9 et x^{10} à coordonnées non entières.

La rotation a géométriquement forcé l'hyperplan $3x_1 + 5x_2 + 6x_3 = 11$ a bouger de façon à venir s'appuyer sur deux nouveaux points à coordonnées entières.

IV.2.3. Direction de la rotation et choix de la bonne direction

a) Définition

Etant donnée une contrainte $A_p\ x \leq b_p$; $S = (s_1, s_2, \ldots, s_n)$ est dite une *direction de la rotation* de cette contrainte si les coefficients $\hat{a}_{pj}$ de $\hat{A}_p$ sont calculés dans l'ordre suivant :

$$\hat{a}_{ps_1}, \hat{a}_{ps_2}, \ldots, \hat{a}_{ps_n}$$

b) Choix de la "bonne" direction

Puisque la rotation d'une contrainte $A_p\ x \leq b_p \quad p \in \{1, \ldots, m\}$ peut se réaliser dans n'importe quelle direction, Walukiewicz et Kaliszewski [71] proposent de la faire suivant une direction permettant une diminution de $v(\bar{B}_p^0)$, dans le but d'améliorer par exemple les bornes d'évaluation par excès de (B) :

Etant donné $p \in \{1, \ldots, m\}$ et le problème $(\bar{B}_p^0)$ résolu sous l'hypothèse :

$$c_1/a_{p1} \geq c_2/a_{p2} \geq \ldots \geq c_n/a_{pn}$$

$$v(\bar{B}_p^0) = \sum_{j=1}^{i-1} c_j + \frac{c_i}{a_{ip}} \left(b_p - \sum_{j=1}^{i-1} a_{pj}\right)$$

où i est l'indice de base

$$\text{i.e.} \quad \sum_{j=1}^{i-1} a_{pj} \leq b_p < \sum_{j=1}^{i} a_{pj}$$

Walukiewicz et Kaliszewski suggèrent la direction suivante :
$S_1 = (1, 2, \ldots, n)$.

En désignant par

. $\bar{x}$: la solution optimale de $(\bar{B}_p^0)$

. $U = \{1, 2, \ldots, i-1\}$, $L = \{i, \ldots, n\}$

. $\hat{i}$: l'indice de base du problème $(\bar{\bar{B}}_p^0)$

. $d_j = c_j - \frac{c_i}{a_{pi}} a_{pj}$ $\forall j$: les coûts réduits optimaux de $(\bar{B}_p^0)$

. $\hat{d}_j = c_j - \frac{c_i}{a_{p\hat{i}}} a_{pj}$ $\forall j$: les coûts réduits optimaux de $(\bar{\bar{B}}_p^0)$

Notre choix de direction de la rotation : $S_2 = (i, i-1, \ldots, n, n-1, \ldots, i+1)$ est basé sur les objectifs suivants :

- Diminuer $v(\bar{B}_p^0)$,
- Eliminer $\bar{x}$ (si $\bar{x}_i$ n'est pas un entier),
- Augmenter les plus grands $|d_j|$ (dans le but de rendre efficaces les tests de réduction [17, 19, 29, 33, 45]).

Ces objectifs constitueront l'objet du théorème ci-dessous.

Théorème 11

1° *Soit* $r \in U \cup \{i\}$ *tel que* $\hat{a}_{pr} > a_{pr}$, *alors la contrainte* $\hat{A}_p x \leq b_p$ *élimine* $\bar{x}$.

2° *Soient* $\varepsilon \in \mathbb{R}^+$ *et* $r \in U \cup \{i\}$ *tels que :*

$$\varepsilon \leq \frac{1}{\max\limits_{j \in U \cup \{i\}} \{c_j\}} \; ; \; \hat{a}_{pr} > a_{pr} \text{ et } \frac{a_{pi}}{c_i} - \varepsilon \leq \frac{a_{pr}}{c_r} \leq \frac{a_{pi}}{c_i}$$

alors :

(i) $\hat{i} \geq i$ *et* $\hat{U} \supset U - \{r\} \cup \{i\}$ *si* $\sum\limits_{j \in U - \{r\}} a_{pj} + a_{pi} \leq b_p$

$\hat{i} \geq i$ *et* $\hat{U} = U - r$ *sinon*

(ii) $\forall j < r$ *tel que* $\hat{a}_{pj} = a_{pj}$ *on a :* $\hat{d}_j \geq d_j$.

3) $\forall r \geq i$ *tel que* $\hat{a}_{pj} = a_{pj}$ $\forall j < r$ *on a* : $|\hat{d}_r| \geq |d_r|$.

Démonstration

1) Montrons $\hat{A}_p \bar{x} > b_p$

$$\left.\begin{array}{l} \hat{A}_p \bar{x} = \sum_{\substack{j=1 \\ j\neq r}}^{n} \hat{a}_{pj} + \hat{a}_{pr} \bar{x}_r \\ \hat{a}_{pj} \geq a_{pj} \quad \forall j \neq r \text{ et } \hat{a}_{pr} > a_{pr} \end{array}\right\} \Rightarrow \hat{A}_p \bar{x} > A_p \bar{x} = b_p$$

2) (i) :

Pour montrer que $\hat{i} \geq i$, il suffit de montrer que $\frac{\hat{a}_{pr}}{c_r} \geq \frac{a_{pi}}{c_i}$

$$\left.\begin{array}{l} \frac{\hat{a}_{pr}}{c_r} = \frac{a_{pr}}{c_r} + \frac{\hat{a}_{pr} - a_{pr}}{c_r} \\ \frac{a_{pi}}{c_i} - \varepsilon \leq \frac{a_{pr}}{c_r} \leq \frac{a_{pi}}{c_i} \end{array}\right\} \Rightarrow \left.\begin{array}{l} \frac{\hat{a}_{pr}}{c_r} \geq \frac{a_{pi}}{c_i} - \varepsilon + \frac{\hat{a}_{pr} - a_{pr}}{c_r} \\ \hat{a}_{pr} > a_{pr} \Rightarrow \frac{\hat{a}_{pr} - a_{pr}}{c_r} > \frac{1}{c_r} \end{array}\right\} \Rightarrow \ldots$$

$$\left.\begin{array}{l} \ldots \frac{\hat{a}_{pr}}{c_r} \geq \frac{a_{pi}}{c_i} - \varepsilon + \frac{1}{c_r} \\ \varepsilon \leq \frac{1}{\max_{j\in\{i\}\cup U}\{c_j\}} \Rightarrow -\varepsilon + \frac{1}{c_r} \geq 0 \end{array}\right\} \Rightarrow \frac{\hat{a}_{pr}}{c_r} \geq \frac{a_{pi}}{c_i}$$

si $\sum_{j\in U-\{r\}} a_{pj} + a_{pi} \leq b_p$ alors $\hat{U} \supset U - \{r\} \cup \{i\}$, sinon $\hat{i} = i$ et $\hat{U} = U - \{r\}$

2) (ii) :

$$\left.\begin{array}{l} \forall j < r \quad \hat{d}_j = c_j - \frac{c_{\hat{i}}}{a_{p\hat{i}}} a_j \\ 2)\ (i) \Rightarrow \hat{i} \geq i \Rightarrow \frac{c_{\hat{i}}}{a_{p\hat{i}}} \leq \frac{c_i}{a_{pi}} \end{array}\right\} \Rightarrow \hat{d}_j \geq c_j - \frac{c_i}{a_{pi}} a_j = d_j$$

3) $r \geq i$ et $\hat{a}_{pj} = a_{pj}$ $\forall j < r \Rightarrow \hat{i} = i \Rightarrow \left| \hat{d}_r \right| = \dfrac{c_i}{a_{pi}} \hat{a}_{pr} - c_r$

$\hat{a}_{pr} \geq a_{pr}$ $\Big\}$ ...⇒

... $\left| \hat{d}_r \right| \geq \dfrac{c_i}{a_{pi}} a_{pr} - c_r = |d_r|$ □

Interprétation du Théorème :

1) En effectuant la rotation dans la direction (i, i-1,..., 1), il y a de fortes chances que :

- $\bar{x}$ soit éliminée et que par conséquent $v(\bar{\bar{B}}_p^0) < v(\bar{B}_p^0)$.

- Les premiers coefficients a_{pj} de A_p auxquels sont en général associés les plus grands coûts réduits ne soient pas modifiés ; ce qui contribue à renforcer les tests d'élimination de variables puisque $\hat{d}_j \geq d_j$ (voir les méthodes de réduction [17, 19, 29, 33, 45]).

2) Lorsque $\hat{a}_{pi} \simeq a_{pi}$, en effectuant la rotation dans la direction (n, n-1,..., i+1), il y a de fortes chances que les derniers coefficients a_{pj} de A_p auxquels sont en général associés les plus grands coûts réduits $|d_j|$ soient augmentés ce qui contribue, dans le cas aussi, à renforcer les tests d'élimination de variables.

Dans l'exemple suivant, l'utilisation de la direction S_2 permet d'éliminer davantage de variables que par l'utilisation de la direction S_1 ; en considérant le test classique d'élimination de variable :

$$\left| v(\bar{B}_p^0) - |d_j| \right| \leq \underline{v}(B_p^0) \Rightarrow \begin{cases} x_j^* = 1 & \text{si } j \in U \\ x_j^* = 0 & \text{sinon} \end{cases}$$

$$(B) \begin{cases} \max\ 52x_1 + 17x_2 + 40x_3 + 793x_4 + 813x_5 + 768x_6 + 724x_4 \\ \text{s.c}\ 45x_1 + 15x_2 + 40x_3 + 793x_4 + 820x_5 + 777x_6 + 737x_7 \leq 870 \\ x_j = 0 \text{ ou } 1 \quad j = 1, \ldots, 7 \end{cases}$$

$v(B) = 52 + 813 = 865$, $x^* = (1, 0, 0, 0, 1, 0, 0)^t$ ([18] p. 129 où b = 837).

	1	2	3	i=4	5	6	7	
c_j	52	17	40	793	813	768	724	b
a_j	45	15	40	793	820	777	737	870
d_j	7	2	0	0	-7	-9	-13	i=4

$v(\bar{B}) = 879$, $\underline{v}(B) = v(B) = 865$

$\forall j \in \{1, \ldots, 7\}$ $\lfloor v(\bar{B} - |d_j| \rfloor > \underline{v}(B)$ donc aucune élimination.

Après rotation de la contrainte suivant la direction $S_1 = (1, 2, \ldots, 7)$ le nouveau tableau est :

	1	4	$\hat{i}$=5	6	3	7	2	
c_j	52	793	813	768	40	724	17	b
$\hat{a}_j$	50	793	820	777	43	750	27	870
$\hat{d}_j$	2.4268	6.7695	0	-2.367	-2.6329	-19.5975	-9.7695	$\hat{i}$=5

$\hat{v}(B) = 871.7695$

$\lfloor \hat{v}(B) - |\hat{d}_2| \rfloor < \underline{v}(B) \Rightarrow x_2^* = 0$; $\lfloor \hat{v}(B) - |\hat{d}_7| \rfloor < \underline{v}(B) \Rightarrow x_7^* = 0$

Rotation suivant la direction $S_2 = (4, 3, 2, 1, 7, 6, 5)$:

	1	2	$\hat{i}$=5	4	6	3	7	
c_j	52	17	813	793	768	40	724	b
$\hat{a}_j$	45	15	825	810	780	45	765	870
$\hat{d}_j$	7.654	2.218	0	-5.218	-0.654	-4.345	-29.872	$\hat{i}$=5

$\hat{v}(B) = 867.218$

$\lfloor \hat{v}(B) - |\hat{d}_j| \rfloor \leq \underline{v}(B)$ $\forall j \neq 5$ et $6 \Rightarrow x_j$ est fixées $\forall j \neq 5$ et 6.

IV.3. ROTATION AVEC UNE CONTRAINTE ADDITIONNELLE

Le principe de la rotation de la contrainte $A_p\ x \leq b_p$ sous une contrainte additionnelle $A_q\ x \in b_q$ $(p, q \in \{1,\ldots, m\})$ est le même que celui de la rotation sans contrainte additionnelle :

Les coefficients $\hat{a}_{pj}$ de $\hat{A}_p$ sont calculés comme suit

Etant donné $r \in \{1,\ldots, n\}$, considérons le problème

$$(B^p_{p,q} \mid x_r = 1) \quad \left\lceil \begin{array}{l} a_{pr} + \max \sum_{\substack{j=1 \\ j\neq r}}^{n} a_{pj}\, x_j \\ \text{s.c} \sum_{\substack{j=1 \\ j\neq r}}^{n} a_{pj}\, x_j \leq b_p - a_{pr} \\ \sum_{\substack{j=1 \\ j\neq r}}^{n} a_{qj}\, x_j \leq b_q - a_{qr} \\ x_j = 0 \text{ ou } 1 \quad \forall j \neq r \end{array} \right.$$

en posant $\hat{a}_{pr} = a_{pr} + b_p - v(B^p_{p,q} \mid x_r = 1)$, le procédé est alors réitéré sur un autre coefficient de la nouvelle contrainte :

$$\hat{a}_{pr}\, x_j + \sum_{\substack{j=1 \\ j\neq r}}^{n} a_{pj}\, x_j \leq b_p.$$

Ce procédé est résumé par l'algorithme suivant :

Algorithme :

0 $I \leftarrow \{1,\ldots, n\}$

1 Tant que $I \neq \emptyset$ faire ; choisir $r \in I$;
$\hat{a}_{pr} \leftarrow a_{pr} + b_p - v(B^p_{p,q} \mid x_r = 1)$;
$I \leftarrow I - \{r\}$

fait

L'introduction d'une contrainte additionnelle permet de réduire davantage le domaine de $(\bar{B})$ comme le montre le théorème ci-dessous.

Pour comparer les rotations avec et sans contrainte additionnelle, par la suite, le paramètre q sera ajouté à tout ce qui est relatif à la rotation avec une contrainte additionnelle $A_q\ x \leq b_q$; par exemple : $\hat{A}_p(q)\ x \leq b_p$ est la contrainte obtenue après rotation de la contrainte $A_q\ x \leq b_p$ sous la contrainte $A_q\ x \leq b_q$.

Théorème 12 :

1) $\hat{a}_{pj}(q) \geq \hat{a}_{pj} \qquad j = 1, \ldots, n$

2) $F(\hat{B}^0_p(q)) \subseteq F(\hat{B}^0_p) = F(B^0_p)$

3) *S'il existe* $r \in \{1, \ldots, n\}$ *tel que* $\hat{a}_{pr}(q) > \hat{a}_{pr}$ *alors* $F(\hat{B}^0_p(q)) \subsetneq F(\hat{B}^0_q)$

4) $F(\bar{\hat{B}}^0_p(q) \subseteq F(\bar{\hat{B}}^0_p)$.

Démonstration

1) $\forall j \in \{1, \ldots, n\}$

$$\left.\begin{aligned} \hat{a}_{pj}(q) &= a_{pj} + b_p - v(B^p_{p,q} \mid x_j = 1) \\ \hat{a}_{pj} &= a_{pj} + b_p - v(B^p_p \mid x_j = 1) \end{aligned}\right\} \Rightarrow \hat{a}_{pj}(q) - \hat{a}_{pj} = v(B^p_p \mid x_j = 1) - v(B^p_{p,q} \mid x_j = 1)$$

or $v(B^p_p \mid x_j = 1) \geq v(B^p_{p,q} \mid x_j = 1)$ d'où $\hat{a}_{pj}(q) \geq \hat{a}_{pj}$.

2) et 4) se déduisent facilement du 1).

3) Soit $r \in \{1, \ldots, n\}$ tel que $\hat{a}_{pr}(q) > \hat{a}_{pr}$ et
x^p telle que $v(B^p_p \mid x_r = 1) = A_p\ x^p$,
montrons que :
$x^p \in F(\hat{B}^0_p)$ et $x^p \notin F(\hat{B}^0_p(q))$:

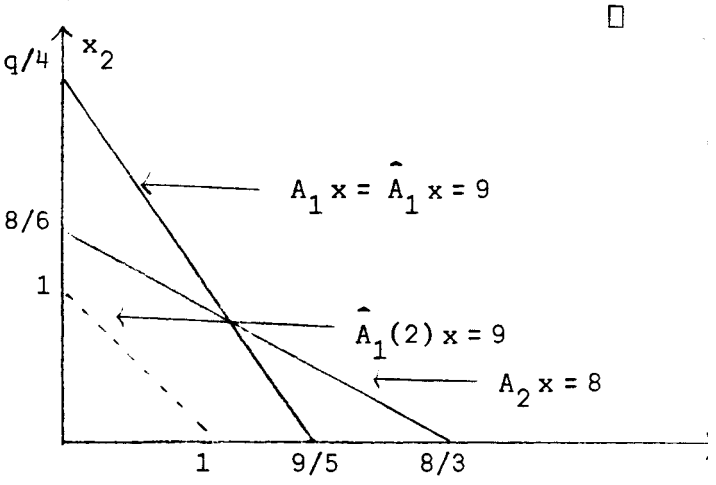
$$1) \Rightarrow \left.\begin{array}{l} \hat{A}_p(q)\, x^p \geq \hat{A}_p\, x^p \\ \hat{a}_{pr}(q) > \hat{a}_{pr} \end{array}\right\} \Rightarrow \left.\begin{array}{l} \hat{A}_p(q)\, x^p > \hat{A}_p\, x^p \\ \text{or} \quad \hat{A}_p\, x^p = b_p \end{array}\right\} \Rightarrow \hat{A}_p(q)\, x^p > b_p$$

□

Exemple : Soient les deux contraintes :

(1) $A_1 x = 5x_1 + 4x_2 \leq 9$

(2) $A_2 x = 3x_2 + 6x_2 \leq 8$



La rotation de la contrainte (1) sans la contrainte additionnelle (2) donne $A_1 = \hat{A}_1$, par contre avec la contrainte (2), on obtient

$$(\hat{1}) \qquad \hat{A}_1(2)\, x = 9x_1 + 9x_2 \leq 9.$$

Remarques :

1) La contrainte $(\hat{1})$ obtenue a éliminé la contrainte additionnelle (2).

2) La contrainte obtenue après rotation, n'élimine pas en général la contrainte additionnelle comme le montre l'exemple suivant :

Soient les deux contraintes

$$(1) \qquad 10x_1 + 14x_2 + 25x_3 + 32x_4 \leq 46$$

$$(2) \qquad 32x_1 + 25x_2 + 14x_3 + 10x_4 \leq 46$$

$$x_j = 0 \text{ ou } 1 \qquad j = 1, 2, 3, 4$$

La rotation de la contrainte (1) avec la contrainte additionnelle (2) est la contrainte :

$\hat{(1)}$ $\quad 14x_1 + 14x_2 + 32x_3 + 32x_4 \leq 46$

La contrainte $\hat{(1)}$ n'élimine pas la contrainte (2) puisque x = (1, 1, 0, 0) vérifie $\hat{(1)}$ mais ne vérifie pas (2). Il est à noter aussi que la rotation de la contrainte (1) sans contrainte additionnelle est $\hat{(1)}$. □

Ces remarques permettent de dégager deux problèmes importants :

- Comment choisir la contrainte additionnelle (celle qui permet un grand déplacement de l'hyperplan d'origine) ?

- Comment la rotation de contraintes peut-elle être utilisée dans les méthodes d'élimination de contraintes ?

Ces deux points seront abordés dans la partie qui suit.

IV.4. LES APPLICATIONS DE LA ROTATION DE CONTRAINTES

La rotation de contraintes peut être utilisée :

- Comme méthode d'obtention de formulations équivalentes : Etant donné un problème (B), il existe plusieurs méthodes pour obtenir un problème équivalent (B') (i.e. F(B) = F(B') et $F(\overline{B'}) \subseteq F(\overline{B})$). La rotation de contraintes rentre dans le cadre de ces méthodes avec la possibilité de les améliorer (partie IV.4.1).

- Dans les méthodes d'élimination de contraintes en renforçant les tests d'élimination (partie IV.4.2) :

Etant donnée une contrainte à éliminer par une autre contrainte du problème (B) ; les tests d'élimination peuvent être efficaces si :

. La contrainte d'élimination est bien choisie ; d'où un critère de choix s'impose.

. La rotation de la contrainte d'élimination est réalisée avec une contrainte additionnelle bien choisie (celle qui permet un grand déplacement) ; d'où un critère de choix s'impose.

- Dans les méthodes algébriques (partie IV.4.3) en

  . renforçant les contraintes d'origines
  . renforçant les coupes générées à chaque itération de la méthode.

IV.4.1 Méthode d'obtention de problèmes équivalents

Etant donné le problème (B) formulé précédemment comme suit

$$(B)\quad \begin{cases} \max \sum_{j=1}^{n} c_j x_j \\ \text{s.c} \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i = 1,\ldots, m \\ \qquad\qquad x_j = 0 \text{ ou } 1,\ j = 1,\ldots, n \end{cases}$$

Il existe différentes techniques d'obtention de formulations dites équivalentes [2, 9, 10, 11, 14, 25, 26, 35, 36, 37, 39, 43, 60, 61, 65, 69, 74] avec exactement les mêmes solutions entières et réalisables. Les expériences répertoriées dans [6, 22, 28, 71, 73] indiquent que (B) est d'autant plus facile à résoudre, ou possède de meilleures solutions approchées (obtenues rapidement) lorsque son domaine est le plus réduit possible.

Comme, il est montré dans les théorèmes précédents, non seulement la rotation de contraintes rentre dans ce cadre des méthodes, mais de plus elle permet de les renforcer en réduisant encore les domaines trouvés par les autres méthodes : voilà quelques contraintes rencontrées dans la littérature et où la rotation de contrainte aboutit à des améliorations :

Contrainte construite par :

. Réduction de ses coefficients (voir l'inégalité (9) page 275 de la référence [12]
. Contraction de contraintes [10, 39].
. La méthode de coupes profondes dites "deep cuts" (voir l'exemple 2(a) de la référence [9]).
. La théorie de groupes et les multiplicateurs lagrangiens généralisés [35, 69].

□

IV.4.2. Méthodes de réduction

Le but de cette partie est de montrer que l'élimination définitive de contraintes (celle des variables à déjà été abordée dans la partie V.2.3) du problème (B) peut être réalisée efficacement par l'introduction de la rotation de contraintes sur la "meilleure" contrainte d'élimination.

IV.4.2.1. Rappel de quelques tests d'élimination de contrainte

La contrainte

$$A_k\, x \leq b_k \qquad k \in \{1,\ldots, m\}$$

peut être éliminée du problème (B) sous l'une des conditions suivantes (Fayard et Plateau [18]) :

$$(C_1) \qquad \sum_{j=1}^{n} a_{kj} \leq b_k$$

$\exists p \in \{1,\ldots, k-1, k+1,\ldots, m\}$ tel que

$$(C_2) \qquad a_{kj} \leq a_{pj} \qquad \forall j \in \{1,\ldots, n\} \text{ et } b_k \geq b_p$$

$$(C_3) \qquad \lfloor v(\bar{B}_p^k) \rfloor \leq b_k$$

$$(C_4) \qquad \lfloor \max\{v(\bar{B}_p^k \mid x_{i(p)} = 0),\ v(\bar{B}_p^k \mid x_{i(p)} = 1)\} \rfloor \leq b_k$$

où i(p) est l'indice de base du problème $(\bar{B}_p^k)$.

Dans le cas où la contrainte d'élimination

$$A_p\, x \leq b_p$$

définie par

$$(I) \qquad v(\bar{B}_p^k) = \min\{v(\bar{B}_q^k),\ q = 1,\ldots, k-1, k+1,\ldots, m\}$$

(une estimation - moins coûteuse que la résolution des m-1 problèmes $(\bar{B}_p^k)$ - de cette contrainte sera proposée en IV.4.2.2.)

ne vérifie pas l'un des tests (C_3) et (C_4), il est proposé en IV.4.2.3. de faire la rotation de celle-ci, avec de préférence une contrainte additionnelle "bien choisie" (i.e. celle qui permet une grande diminution de $v(\overline{B}_p^k)$). □

IV.4.2.2. "Estimation" de la contrainte d'élimination

L'étude du cas particulier (a) permet de comprendre le critère "d'estimation" donné en b.

a) Etude d'un cas particulier

Etant donnée une contrainte

$$A_k\, x \leq b_k \qquad k \in \{1, 2, \ldots, m\}$$

Considérons une contrainte

$$A_p\, x \leq b_p \qquad p \in \{1, \ldots, k-1, k+1, \ldots, m\}$$

et les deux ordonnancements suivants :

$$(e) \begin{cases} a_{k_1} \geq a_{k_2} \geq \ldots \geq a_{k_n} \\ a_{pe_1} \geq a_{pe_2} \geq \ldots \geq a_{pe_2} \end{cases}$$

$$(f) \begin{cases} a_{k_1} \geq a_{k_2} \geq \ldots \geq a_{k_n} \\ a_{pf_1} \leq a_{pf_2} \leq \ldots \leq a_{pf_n} \end{cases}$$

en désignant par :

$$A_p(e) = (a_{pe_1}, \ldots, a_{pe_n})$$

$$A_p(f) = (a_{pf_1}, \ldots, a_{pf_n})$$

$(B_p^k(e))$ max $A_k\, x$ s.c $A_p(e)\, x \leq b_p$, $x \in V$

$(B_p^k(f))$ max $A_k\, x$ s.c $A_p(f)\, x \leq b_p$, $x \in V$

$$\cos^2(c, a) = \Big(\sum_{j=1}^{n} c_j\, a_j\Big)^2 \Big/ \sum_{j=1}^{n} c_j^2 \sum_{j=1}^{n} a_j^2.$$

(où $c = (c_1, \ldots, c_n)$, $a = (a_1, \ldots, a_n)$

nous avons les résultats suivants :

Lemme 3

1) $\cos^2(A_k, A_p(e)) \geq \cos^2(A_k, A_p(f))$

2) $v(\bar{B}_p^k(e)) \leq v(\bar{B}_p^k(f))$

Démonstration

1) Evident compte tenu de (e) et (f).

2) En notant par :

$i(e)$ = indice de base optimale de $(\bar{B}_p^k(e))$

i.e. $\exists U \subset \{1, \ldots, n\}$: $\sum_{j\in U} a_{pe_j} \leq b_p < \sum_{j\in U} a_{pe_j} + a_{pe_{i(e)}}$ et

$$a_{kj}/a_{pe_j} \geq a_{ki(e)}/a_{pe_{i(e)}} = r(e) \quad \forall j \in U$$

$i(f)$ = indice de base optimale de $(\bar{B}_p^k(f))$

i.e. $\sum_{j=1}^{i(f)-1} a_{pf_j} \leq b_p < \sum_{j=1}^{i(f)} a_{pf_j}$, soit $r(f) = a_{ki(f)}/a_{pf_{i(f)}}$

alors, $v(\bar{B}_p^k(e)) = \sum_{j\in U} a_{kj} + (b_p - \sum_{j\in U} a_{pe_j})\, r(e)$

et

$$v(\bar{B}_p^k(f)) = \sum_{j=1}^{i(f)-1} a_{kj} + (b_p - \sum_{j=1}^{i(f)-1} a_{pf_j})\, r(f)$$

montrons que $v(\bar{B}_p^k(e)) - v(\bar{B}_p^k(f)) \leq 0$.

$$\text{(1)} \qquad v(\bar{B}_p^k(e)) - v(\bar{B}_p^k(f)) = \Big(\sum_{j\in U} a_{kj} - \sum_{j=1}^{i(f)-1} a_{kj}\Big) + \Big(b_p - \sum_{j\in U} a_{pe_j}\Big)\, r(e) - \Big(b_p - \sum_{j=1}^{i(f)-1} a_{pf_j}\Big)\, r(f)$$

$$(e) \text{ et } (f) \Rightarrow \Big(\sum_{j\in U} a_{kj} - \sum_{j=1}^{i(f)-1} a_{kj}\Big) \le 0 \qquad \text{(2)}$$
$$\text{et } |U| \le i(f) - 1$$

$$\left.\begin{array}{l} \text{si } |U| = i(f) - 1 \text{ alors } r(f) \ge r(e) \\ \text{et } \displaystyle\sum_{j=1}^{i(f)-1} a_{pf_j} \le \sum_{j\in U} a_{pe_j} \\ \text{(1)} \end{array}\right\} \Rightarrow v(\bar{B}_p^k(e) - v(\bar{B}_p^k(f)) \le 0$$

$$\left.\begin{array}{l} \text{Sinon : } |U| < i(f) - 1 \\ (e) \text{ et } (f) \end{array}\right\} \rightarrow \sum_{j\in U} a_{kj} + a_{ki(e)} \le \sum_{j=1}^{i(f)-1} a_{kj} \qquad \text{(3)}$$

$$\left.\begin{array}{l} \text{or : } \Big(b_p - \displaystyle\sum_{j\in U} a_{pe_j}\Big)\, r(e) \le a_{ki(e)} \\ \text{(1) , (3)} \end{array}\right\} \Rightarrow v(\bar{B}_p^k(e)) - v(\bar{B}_p^k(f)) \le 0 \qquad \square$$

Ce cas particulier permet de donner la conclusion suivante :

En considérant toutes les permutations (s) des coefficients de A_p il y a de fortes chances que le "meilleur" knapsack (celui qui a la plus petite valeur $v(\bar{B}_p^k(s))$) soit celui qui a le plus grand $\cos^2(A_k, A_p(s))$. Géométriquement cela signifie que la contrainte du "meilleur" knapsack est celle qui a la plus petite "déviation" avec la contrainte $A_k\, x \le b_k$.

La recherche de la contrainte $A_p\ x \leq b_p$ ayant le plus grand $\cos^2(A_k, A_p)$ n'est pas suffisant, puisque si

$$b_p \simeq \sum_{j=1}^{n} a_{pj}$$

alors

$$v(\overline{B}_p^k) \simeq \sum_{j=1}^{n} a_{kj}$$

Pour cela il faut aussi tenir compte du rapport $b_p / \sum_{j=1}^{n} a_{pj}$ quand p décrit l'ensemble $\{1,\ldots, m\}\backslash\{k\}$.

b) Critère "d'estimation"

Etant donnée une contrainte

$$A_k\ x \leq b_k \qquad k = 1,\ldots, m$$

la contrainte

$$A_p\ x \leq b_p \qquad p = 1,\ldots, k-1, k+1,\ldots, m$$

susceptible de vérifier (I) est "estimée" par le critère suivant :

$$\text{(II)} \quad \left(\frac{b_p}{\sum_{j=1}^{n} a_{pj}}\right) / \cos^2(A_k, A_p) = \min\left\{\left(\frac{b_q}{\sum_{j=1}^{n} a_{qj}}\right) / \cos^2(A_k, A_q)\ ;\ \forall q \neq k\right\}$$

Remarque : En supposant que le "meilleur" knapsack est tel que :

$$v(\overline{B}_1^0) = \min\{v(\overline{B}_k^0) \quad k = 2,\ldots, m\}$$

une estimation naturelle de la contrainte d'élimination serait de prendre la contrainte

$$A_1\ x \leq b_1$$

(Tests appliqués sur (B_1^k) $k = 2,\ldots, m$).

Cette estimation n'est pas tout-à-fait réaliste puisqu'elle ne tient pas compte du tout de la contrainte $A_k\ x \leq b_k$ à éliminer comme l'illustre l'exemple suivant :

$$(B)\quad \begin{cases} \max\ 4x_1 + 5x_2 + 2x_3 \\ \quad A_1\ x = 2x_1 + 3x_2 + x_1 \leq 3 = b_1 \\ \text{s.c}\ A_2\ x = 5x_1 + 2x_2 + 4x_2 \leq 8 = b_2 \\ \quad A_3\ x = 3x_1 + x_2 + 2x_2 \leq 4 = b_3 \\ \qquad x_j = 0 \text{ ou } 1 \quad j = 1, 2, 3 \end{cases}$$

. $v(\bar{B}_1^0) = 6 < v(\bar{B}_3^0) = 9 < v(\bar{B}_2^0) = 9{,}5$

. La contrainte 1 n'élimine pas les autres.

Dans cet exemple l'estimation (II) fournie la contrainte théorique donnée par (I), voir le tableau suivant :

i	1	2	3	(I^*)	(II^*)	$(C_9{}^*)$
$v(\bar{B}_i^1)$		5.25	5	3		non
$(b_i / \sum_{j=1}^{3} a_{ij})/\cos^2(A_1, A_i)$		1.14	1.08		3	
$v(\bar{B}_i^2)$	9		7.66	3		oui
$(b_i / \sum_{j=1}^{3} a_{ij})/\cos^2(A_2, A_i)$	0.78		0.6		3	
$v(\bar{B}_i^3)$	5	4.5		2		oui
$(b_i / \sum_{j=1}^{3} a_{ij})/\cos^2(A_3, A_i)$	0.81	0.73			2	

(I^*) (resp. (II^*)) désigne le numéro de la contrainte vérifiant (I) (resp. (II)).

$(C_3)^*$ indique si oui ou non le test (C_3) est vérifié.

Il est à noter que la contrainte 1 n'a pas pû être éliminée par la contrainte 3 ; par contre, si on fait la rotation de la contrainte 3 avec la contrainte additionnelle 1, la contrainte suivante obtenue :

$$\hat{A}_3 \, x = 4x_1 + 4x_2 + 4x_3 \leq 4$$

élimine la contrainte 1;

IV.4.2.3. Rotation de la contrainte d'élimination

Si la contrainte

$$A_k \, x \leq b_k \qquad k \in \{1, \ldots, m\}$$

n'a pas pû être éliminée par la contrainte

$$A_p \, x \leq b_p \qquad \text{(contrainte d'élimination)}$$

(où p vérifie (II)), alors en faisant la rotation de cette dernière, on espère que la contrainte obtenue

$$\hat{A}_p \, x \leq b_p$$

pourrait éliminer la contrainte $A_k \, x \leq b_k$.

La rotation de la contrainte $A_p \, x \leq b_p$ est d'autant plus efficace lorsque elle s'effectue avec une contrainte additionnelle "bien choisie". Nous avons constaté (sur des exemples) que la contrainte qui répond à ce choix est celle qui a les quantités $(b_q \, / \, \sum_{j=1}^{n} a_{qj})$ et $\cos^2(A_p, A_q)$ petites.

C'est la raison pour laquelle la contrainte additionnelle $A_q \, x \leq b_q$ $q \neq p$ sera choisie suivant le critère suivant :

(III) $$\left(b_q / \sum_{j=1}^{n} a_{qj}\right) \cos^2(A_p, A_q) = \min_{\forall i \neq p} \left\{\left(b_i / \sum_{j=1}^{n} a_{ij}\right) \cos^2(A_p, A_i)\right\}$$

Exemple : Petersen [62] (extrait du problème n° 1)

(B)
$$\begin{cases} \max\ 100x_1 + 600x_2 + 13x_3 + 64x_4 + 22x_5 + 41x_6 \leq 80 \\ \quad A_1\ x = 8x_1 + 12x_2 + 13x_3 + 64x_4 + 22x_5 + 41x_6 \leq 80 \\ \text{s.c}\ A_2\ x = 3x_1 + 6x_2 + 4x_3 + 18x_4 + 6x_5 + 4x_6 \leq 20 \\ \quad A_3\ x = 5x_1 + 10x_2 + 8x_3 + 32x_4 + 6x_5 + 12x_6 \leq 36 \\ \quad A_4\ x = 5x_1 + 13x_2 + 8x_3 + 42x_4 + 6x_5 + 20x_6 \leq 44 \\ \quad A_5\ x = 5x_1 + 13x_2 + 8x_3 + 48x_4 + 6x_5 + 20x_6 \leq 48 \\ \qquad x_j = 0 \text{ ou } 1 \quad j = 1, \ldots, 6 \end{cases}$$

i	1	2	3	4	5	(I*)	(II*)	(C
$v(\bar{B}_i^1)$		98.55	99	91.61	96	4		nor
$(b_i / \sum_{j=1}^{6} a_{ij})/\cos^2(A_1, A_i)$		0.55	0.52	0.49	0.51		4	
$v(\bar{B}_i^2)$	26.2		23.43	24.14	25	3		nor
$(b_i / \sum_{j=1}^{6} a_{ij})/\cos^2(A_2, A_i)$	0.56		0.51	0.512	0.515		3	
$v(\bar{B}_i^3)$	46.5	41.33	51.5	38.14	39.66	4		no
$(b_i / \sum_{j=1}^{6} a_{ij})/\cos^2(A_3, A_i)$	0.53	0.50	0.49	0.47	0.48		4	
$v(\bar{B}_i^4)$	57.5	57.33	56		48	5		no
$(b_i / \sum_{j=1}^{6} a_{ij})/\cos^2(A_4, A_i)$	0.52	0.52	0.49		0.48		5	
$v(\bar{B}_i^5)$	63.5	62.66	56	50		4		nc
$(b_i / \sum_{j=1}^{6} a_{ij})/\cos^2(A_5, A_i)$	0.53	0.52	0.49	0.46			4	

(I^*), (II^*) et (C_3^*) ont la même signification que précédemment.

Une première constatation fournie par le tableau précédent conserne l'efficacité de l'estimation (II) qui donne la contrainte théorique vérifiant (I).

La deuxième constatation est qu'aucune contrainte n'a été éliminée (test (C_3) n'est pas vérifié).

Puisque la contrainte 4 n'élimine pas la contrainte 1, on propose de faire la rotation de la contrainte 4 avec une contrainte additionnelle vérifiant le critère (III). Il se trouve que cette contrainte additionnelle est la contrainte 1 :

La contrainte suivante,

$$\hat{A}_4\, x = 11x_1 + 13x_2 + 11x_3 + 44x_4 + 11x_5 + 20x_6 \leq 44$$

obtenue après rotation de la contrainte 4 avec la contrainte additionnelle 1, élimine toutes les autres.

On obtient ainsi le problème réduit suivant :

$$\begin{cases} \max\ 100x_1 + 600x_2 + 1200x_3 + 2400x_4 + 500x_5 + 2000x_6 \\ 11x_1 + 13x_2 + 11x_3 + 44x_4 + 11x_5 + 20x_6 \leq 44 \\ x_j = 0 \text{ ou } 1 \quad j = 1,\ldots, 6 \end{cases}$$

IV.5. ALGORITHME DE ROTATION DE CONTRAINTES

Introduction

Considérons une contrainte du problème (B), de la forme

$$ax = \sum_{j=1}^{n} a_j x_j \leq b \ ; \ x \in V$$

(où $\sum_{j=1}^{n} a_j > b$ et $0 \leq a_j < b \quad j = 1,\ldots, n$).

Le but des algorithmes de rotation est de calculer séquentiellement les coefficients $\hat{a}_j$ de la contrainte

$$\hat{a}x = \sum_{j=1}^{n} \hat{a}_j x_j \leq b$$

obtenue après rotation de la précédente. Ce sont tous des algorithmes de programmation dynamique. Kianfar en 1971 [40] a donné un algorithme de compléxité temporelle $O(n^2b)$ et spatiale $O(nb)$; et l'a amélioré en 1976 [41] en minimisant le nombre d'itérations (voir section IV.5.2.).

En section (IV.5.3), nous avons apporté une amélioration considérable à la méthode de Kianfar en donnant un algorithme de compléxité temporelle et spatiale $O(nb)$.

Les résultats théoriques et numériques (section IV.5.4) montrent nettement l'efficacité du nouvel algorithme par rapport à celui de Kianfar.

IV.5.1. Formulation

Par simplification, les coefficients $\hat{a}_j$ de $\hat{a}$ sont supposés calculés dans l'ordre suivant :

$$\hat{a}_n, \hat{a}_{n-1}, \ldots, \hat{a}_1$$

et sans contrainte additionnelle.

Etant donné $k \in \{1, \ldots, n\}$; $\hat{a}_k$ est calculé par :

$$\hat{a}_k = b - \mathbf{v}(S_k^n) \qquad (4.1)$$

où la valeur du problème

$$(S_k^n) \quad \max \sum_{j=1}^{k-1} a_j x_j + \sum_{j=k+1}^{n} \hat{a}_j x_j \text{ s.c. } \sum_{j=1}^{k-1} a_j x_j + \sum_{j=k+1}^{n} \hat{a}_j x_j \leq b - a_k ; x_j = 0 \text{ ou } 1 \quad \forall j \neq k$$

$$\text{(avec } \sum_{j=1}^{0} a_j x_j = \sum_{j=n+1}^{n} \hat{a}_j x_j = 0)$$

est calculée par :

$$v(S_k^n) = \max\{\omega \mid \omega \in L_k^n,\ \omega \le b - a_k\} \qquad (4.2)$$

où L_k^n vérifie :

$$L_k^0 = \{0\}$$

$$L_k^j = L_k^{j-1} \cup \{\omega + a_j \mid \omega \in L_k^{j-1},\ \omega + a_j \le \bar{b}\} \qquad j = 1,\ldots, k-1 \qquad (4.3)$$

$$L_k^k = L_k^{k-1}$$

$$L_k^j = L_k^j \cup \{\omega + \hat{a}_j \mid \omega \in L_k^{j-1},\ \omega + \hat{a}_j \le \bar{b}\} \qquad j = k+1,\ldots, n \qquad (4.4)$$

avec $\bar{b} = b - \min\limits_{1\le j\le n} a_j$.

IV.5.2. Algorithme de Kianfar

En utilisant la programmation dynamique pour construire les listes L_k^j ($\forall j, \forall k \in \{1,\ldots, n\}$), Kianfar [40] donne un algorithme de calcul des coefficients $\hat{a}_j$ ($j = n, n-1,\ldots, 1$) de compléxités temporelle $O(n^2 b)$ et spatiale $O(nb)$:

Algorithme A9

But : Calculer $\hat{a}_n, \hat{a}_{n-1},\ldots, \hat{a}_1$

0 $k \leftarrow n$

1 Construire $L_n^1, L_n^2,\ldots, L_n^{n-1}$ en utilisant (4.3) : phase préliminaire
$\hat{a}_n \leftarrow b - v(S_n^n)$ ($v(S_n^n)$ est calculée par (4.2) avec $L_n^n = L_n^{n-1}$)
$k \leftarrow k-1$

2 Tant que $k \neq 0$ faire
$L_k^k \leftarrow L_n^{k-1}$ (puisque $L_k^{k-1} = L_n^{k-1}$)
pour j de k+1 à n faire
construire L_k^j par (4.4),
$\hat{a}_k \leftarrow b - v(S_k^n)$ ($v(S_k^n)$ étant calculée par (4.2))
$k \leftarrow k-1$
fait
fait

Théorème 13

L'algorithme A9 est de complexités temporelle $O(n^2b)$ *et spatiale* $O(nb)$.

Démonstration

Puisque on gère n listes de taille b chacune, la compléxité spatiale de l'algorithme A9 est à l'évidence O(nb).

Pour la compléxité temporelle, il suffit d'étudier les compléxités temporelles des étapes 1 et 2 :

- La construction par la programmation dynamique des listes L_n^1, L_n^2,..., L_n^{n-1} est de compléxité témporelle O((n-1)b).

- Dans l'étape 2 la construction de chaque liste L_k^n $k = n-1, n-2, \ldots, 1$ (la boucle Tant que) nécessite la construction des (n-k) listes L_k^j $j = k+1, \ldots, n$ (la boucle pour) de taille b chacune ; ce qui donne à l'étape 2 une compléxité temporelle

$$O(b \sum_{k=n-1}^{1} (n-k)) = O(b\ n(n-1)/2)$$

On déduit que l'algorithme A9 est de compléxité temporelle $O(n^2b)$. □

Exemple : Etant donnée la contrainte

$$5x_1 + 11x_2 + 3x_3 + 6x_4 \leq 12$$

Soit $\hat{a}_1 x_1 + \hat{a}_2 x_2 + \hat{a}_3 x_3 + \hat{a}_4 x_4 \leq 12$

la contrainte optenue après rotation de la contrainte précédente suivant la direction (4, 3, 2, 1).

Déroulement de l'algorithme A9

$\bar{b} = 12 - 3 = 9$, $L_4^0 = \{0\}$

$L_4^1 = \{0, 5\}$, $L_4^2 = \{0, 5\}$, $L_4^3 = \{0, 3, 5, 8\} = L_4^4$ $v(S_4^4) = 5$, $\hat{a}_4 = 12 - 5 = 7$

$L_3^3 = L_4^2$, $L_3^4 = \{0, 5, 7\}$, $v(S_3^4) = 7$, $\hat{a}_3 = 12 - 7 = 5$

$L_2^2 = L_4^1$, $L_2^3 = \{0, 5\}$, $L_2^4 = \{0, 5, 7\}$, $v(S_2^4) = 0$, $\hat{a}_2 = 12 - 0 = 12$

$L_1^1 = L_4^0$, $L_1^2 = \{0\}$, $L_1^3 = \{0, 5\}$; $L_1^4 = \{0, 5, 7\}$, $v(S_1^4) = 7$, $\hat{a}_1 = 12 - 7 = 5$

d'où $\hat{a}\ x = 5x_1 + 12x_2 + 5x_3 + 7x_4 \leq 12$ □

Dans l'algorithme A9, l'étape 1 nécessite (n-1) itérations de (4.3) et l'étape 2 nécessite

$$\sum_{k=n-1}^{1} (n-k) = n(n-1)/2$$

itérations de (4.4).

En désignant par

$$u(n) = n+1 + n(n-1)/2$$

le nombre total d'itérations de (4.3) et (4.4), Kianfar [41] montre qu'il est possible d'améliorer l'algorithme A9 en minimisant le nombre d'itérations u(n) :

Algorithme A9 amélioré : Algorithme 10

0 Etant donné un entier naturel q_n ($q_n \leq n-1$) et $L_n^0 \leftarrow \{0\}$

1 Tant que $n > 1$ faire

1.1. Calculer $\hat{a}_n, \hat{a}_{n-1}, \ldots, \hat{a}_{n-q_n+1}$ par l'algorithme A9,

1.2. Pour j de $n - q_n^* + 1$ à n faire

$$L_n^0 \leftarrow L_n^0 \cup \{\omega + \hat{a}_j \mid \omega \in L_n^0,\ \omega + \hat{a}_j \leq \bar{b}\} \qquad (4.5)$$

fait

• $n \leftarrow n - q_n$; donner q_n ; $L_n^0 \leftarrow L_{n+q_n}^0$

fait

2 $\hat{a}_1 \leftarrow b - \max\{\omega \mid \omega \in L_n^0,\ \omega \leq b - a_1\}$.

Détermination de q_n^* :

En désignant par

f(n) = le nombre minimum d'applications de (4.3), (4.4) ou (4.5) dans l'algorithme A10,

il est clair qu'après l'étape 1.1

$$f(q_n) = n-1 + \sum_{k=n-1}^{n-q_n+1} (n-k) = n-1 + q_n(q_n - 1)/2$$

et il reste $n - q_n$ coefficients à calculer. L'idée de Kianfar est de trouver q_n^* optimum ; solution de l'équation suivante :

$$f(n) = \min_{q_n=1,\ldots,n-1} \{f(q_n) + f(n - q_n)\} \qquad (4.6)$$

En résolvant par la programmation dynamique l'équation (4.6) pour différentes valeurs de n, l'auteur obtient les résultats suivants :

Théorème 14 : Kianfar [41]

$\forall n \in \mathbb{N}^*$ la solution optimale de (4.6) est :

- $q_n^* = \left\lfloor \frac{\sqrt{8n-7} - 1}{2} \right\rfloor$
- $f(n) = n - 1 + q_n^*(n - \frac{5}{6}) - \frac{1}{6} q_n^{*3}$

Il est à noter que la compléxité temporelle de l'algorithme A10 est O(b f(n)

Exemple : Considérons la contrainte précédente

$$5x_1 + 11x_2 + 3x_3 + 6x_4 \leq 12$$

Déroulement de l'algorithme A10

$n = 4 \quad q_4^* = 2,\ \bar{b} = 12 - 3 = 9$

1.1 $L_4^0 = \{0\},\ L_4^1 = \{0, 5\},\ L_4^2 = \{0, 5\},\ L_4^3 = \{0, 3, 5, 8\},\ L_4^4 = L_4^3 \rightarrow \begin{cases} v(S_4^4) = 5 \\ \hat{a}_4 = 12 - 3 = 7 \end{cases}$

$L_3^3 = L_4^2,\ L_3^4 = \{0, 5, 7\} \rightarrow \begin{cases} v(S_3^4) = 7 \\ \hat{a}_3 = 12 - 7 = 5 \end{cases}$

1.2 $L_4^0 \leftarrow \{0, 5\},\ L_4^0 \leftarrow \{0, 5, 7\}$

$n \leftarrow 4 - 2 = 2,\ q_2^* = 1\ ;\ L_2^0 \leftarrow L_4^0$

1.1 $L_2^0 = \{0, 5, 7\},\ L_2^1 = \{0, 5, 7\},\ L_2^2 = L_2^1 \rightarrow v(S_2^2) = 0,\ \hat{a}_2 = 12 - 0 = 12$

1.2 $L_2^0 \leftarrow \{0, 5, 7\},\ n \leftarrow 2 - 1 = 1$

2. $\hat{a}_1 = 12 - \max\{\omega \mid \omega \in L_2^0 \mid \omega \leq 12 - 5\} = 12 - 7 = 5.$ □

Le gain - en nombre d'itérations de (4.3), (4.4) ou (4.5) - de la deuxième méthode par rapport à la première est :

$$u(n) - f(n)$$

Dans l'exemple précédent, $u(4) - f(4) = 1$;

Si $n = 100$, $u(100) - f(100) = 5049 - 1030 = 4019.$ □

IV.5.3. Nouvel algorithme : DF

Cet algorithme est basé sur la programmation dynamique et la procédure de fusion étudiée dans le Chapitre I. Il est de compléxités temporelle et spatiale O(nb).

Il se déroule en deux phases :

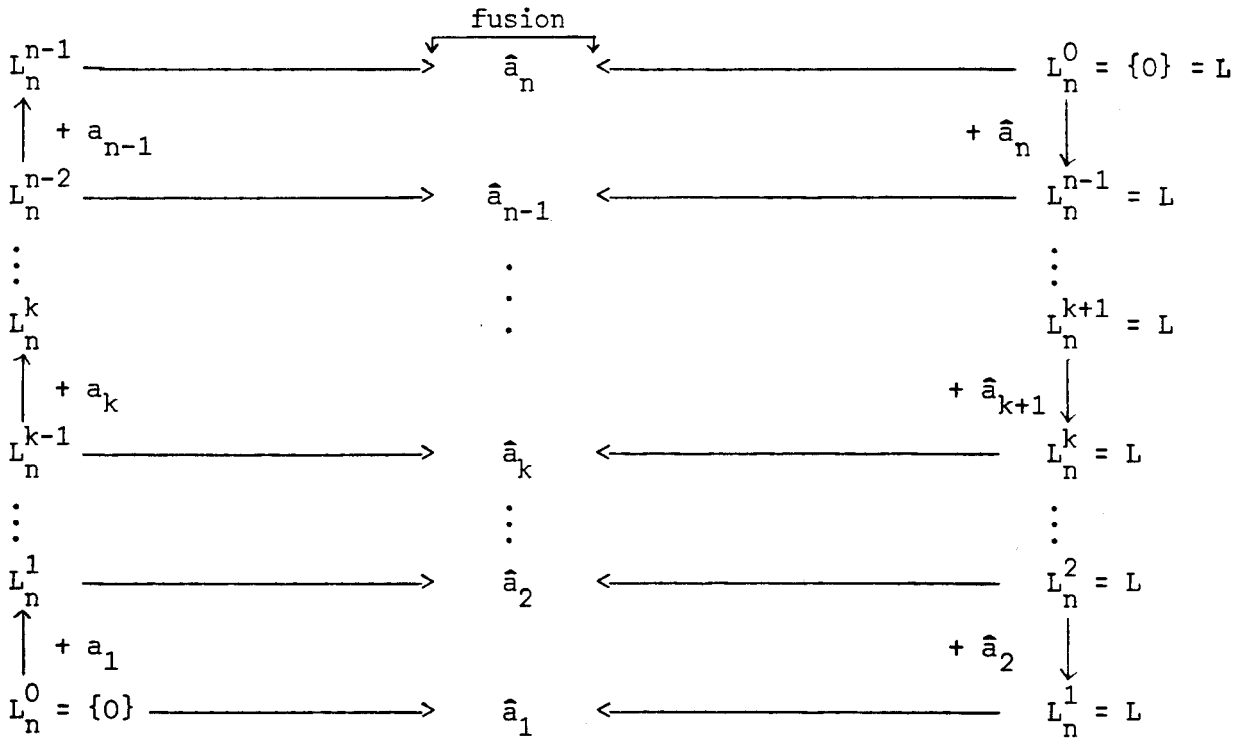
Phase 1 : Cette phase est analogue à la première étape du premier algorithme A9 (phase préliminaire), c'est-à-dire :

- on construit les listes $L_n^0, L_n^1, \ldots, L_n^{n-1}$,
- puis on calcule $\hat{a}_n$ par (4.1) : $\hat{a}_n = b - v(S_n^n)$ où $v(S_n^n)$ est calculée par (4.2) ou en fusionnant les listes L_n^{n-1} et $\{0\}$ pour le second membre $b - a_n$.

Phase 2 : $k \leftarrow n-1$, $L \leftarrow \{0\}$

Tant que $k \neq 0$ faire
$L \leftarrow L \cup \{\omega + \hat{a}_{k+1} \mid \omega \in L,\ \omega + \hat{a}_{k+1} \leq \bar{b}\}$
$\hat{a}_k \leftarrow b - v(S_k^n)$ ($v(S_k^n)$ étant donnée par la fusion des listes L_n^{k-1} et L pour le second membre $b - a_k$) ;
$k \leftarrow k-1$;
fait

Les deux phases sont schématisées ainsi :



Phase 1 Phase 2

Théorème 15

L'algorithme DF est de complexités temporelle et spatiale O(nb).

Démonstration

Dans la phase 1, on construit (n-1) listes ; la taille de chacune d'elles étant majorée par b.

Dans la phase 2, on construit (n-1) listes à la place des (n-1) listes de la phase 1.

On déduit :

- D'une part que la compléxité spatiale de DF est O(nb).

- D'autre part (sachant que la compléxité temporelle de la fusion de deux listes est O(2b) = O(b)) que la compléxité temporelle est $O(2(n-1)b + 2b(n-1)) = O(4(n-1)b) = O(nb)$. □

Exemple : Considérons la contrainte précédente

$$5x_1 + 11x_2 + 3x_3 + 6x_4 \leq 12$$

Déroulement de l'algorithme DF

$L_4^3 = \{0, 3, 5, 8\} \rightarrow v(S_4^4) = 5,\ \hat{a}_4 = 12 - 5 = 7 \leftarrow \{0\} = L_4^0$

$L_4^2 = \{0, 5\} \rightarrow v(S_3^4) = 7,\ \hat{a}_3 = 12 - 7 = 5 \leftarrow \{0, 7\} = L_4^3$

$L_4^1 = \{0, 5\} \rightarrow v(S_2^4) = 0,\ \hat{a}_2 = 12 - 0 = 12 \leftarrow \{0, 5, 7\} = L_4^2$

$L_4^0 = \{0\} \rightarrow v(S_1^4) = 7,\ \hat{a}_1 = 12 - 7 = 5 \leftarrow \{0, 5, 7\} = L_4^1$

Phase 1 Phase 2

La contrainte obtenue après rotation est

$$5x_1 + 12x_2 + 5x_3 + 7x_4 \leq 12$$

IV.5.4. Comparaison de ces deux algorithmes

IV.5.4.1. Comparaison théorique

Nous allons comparer l'algorithme DF avec l'algorithme amélioré A10. D'après ce qui précède, le nombre :

. minimum d'opérations élémentaires théoriques
  - pour l'algorithme A10 est bf(n)
  - pour l'algorithme DF est 2(n-1)b

. maximum d'opérations élémentaires théoriques
  - pour A10 est b(n-1)(n+2)/2
  - pour DF est 4(n-1)b

En désignant par T_1 et T_2 les nombres d'opérations élémentaires pratiques, respectivement de DF et de A10, les doubles inégalités suivantes sont vérifiées :

$$2(n-1)b \leq T_1 \leq 4(n-1)b$$
$$f(n)b \leq T_2 \leq b(n-1)(n+1)/2$$

Il est à noter que b(n-1) est le nombre d'opérations élémentaires théoriques nécessaires pour la construction des listes L_n^1, L_n^2,..., L_n^{n-1} (phase préliminaire) dans les deux algorithmes ; soit donc T = (n-1)b.

En désignant par :

- $R_1 = T_1/T$; $R_2 = T_2/T$

et en remplaçant dans f(n) $q_n^* = \left\lfloor \frac{\sqrt{8n-7}-1}{2} \right\rfloor$ par $\frac{\sqrt{8n-7}-1}{2}$, nous avons :

. $f(n) = n-1 + \frac{n-1}{3}\sqrt{8n-7}$
. $2 \leq R_1 \leq 4$
. $1 + \sqrt{8n-7}/3 \leq R_2 \leq (n+1)/2$
. $g = T_2/T_1 \geq f(n)b/4(n-1)b = (1 + \sqrt{8n-7}/3)/4 = \bar{g}$

Le dernier point signifie que l'algorithme DF est au moins $\bar{g}$ fois plus rapide que A10.

IV.5.4.2. Comparaison expérimentale

Les algorithmes DF et A10 sont codés en fortran IV et mis en oeuvre sur le calculateur CII HB IRIS 80. Ils sont testés sur quatre contraintes de 40, 60, 80 et 100 variables où les a_j sont tirés au hasard dans l'intervalle $[10^2, 10^3]$, et b est fixé à $n+10^3$.

Les temps T, T_1 et T_2 sont donnés en dixième de secondes. Tous les paramètres du tableau 7, ont été définis dans le paragraphe précédent.

n	T	T_1	T_2	$2 \leq R_1 \leq 4$	$1+\frac{\sqrt{8n-7}}{3} \leq R_2 \leq \frac{n+1}{2}$			$\bar{g}$	g
40	3.	12.	61.	4	6.9	20.3	20.5	1.7	5
60	72.	260.	1838.	3.61	8.2	25.5	30.5	2.1	7
80	169.	485.	2881.	2.87	9.4	17.1	40.5	2.4	6
100	305.	723.	4806.	2.37	10.4	15.4	50.3	2.6	6.6

Tableau 7

L'analyse des résultats de ce tableau indique que :

- L'algorithme DF est en moyenne 6 fois plus rapide que A10.

- Lorsque n augmente, R_1 et R_2 ont tendance à s'approcher des bornes inférieures ; cela s'explique par le fait que $b = 10^3 + n$ n'augmente pas beaucoup lorsque n varie ; en conséquence le nombre d'éléments de chaque liste de la phase préliminaire a tendance à se stabiliser.

****
* ANNEXE *
****

PROBLEMES DETAILLES DU CHAPITRE II
(Tableau 4 bis) ET DU CHAPITRE III (Tableau 6)

PROBLEME P.1

$x^*(j)$: fixée par la réduction	variables	c(j)	a(j)	$x^*(j)$
0	x(1)	82	1307	-
0	x(2)	91	4052	-
0	x(3)	48	5608	-
-	x(4)	7	9556	0
0	x(5)	40	3754	-
0	x(6)	57	7429	-
0	x(7)	86	1656	-
0	x(8)	56	225	-
-	x(9)	21	7603	1
-	x(10)	9	7621	0
-	x(11)	10	679	0
0	x(12)	97	312	-
-	x(13)	14	3178	0
-	x(14)	3	314	0
0	x(15)	99	8531	-
0	x(16)	62	7629	-
0	x(17)	56	7594	-
-	x(18)	6	7570	1
0	x(19)	51	8267	-
0	x(20)	40	8694	-
0	x(21)	99	3189	-
-	x(22)	16	6563	0
0	x(23)	26	6658	-
0	x(24)	85	772	-
0	x(25)	68	2578	-
0	x(26)	92	3971	-
0	x(27)	89	1421	-
0	x(28)	98	1816	-
0	x(29)	82	2984	-
-	x(30)	13	1974	0
0	x(31)	60	2003	-
0	x(32)	48	8507	-
-	x(33)	6	2591	0
-	x(34)	19	4013	0
-	x(35)	23	9358	0
0	x(36)	56	9943	-
0	x(37)	27	437	-
0	x(38)	41	9309	-
0	x(39)	56	2594	-
0	x(40)	95	8340	-
0	x(41)	54	4174	-
0	x(42)	45	3167	-
0	x(43)	35	7215	-
-	x(44)	2	466	1
0	x(45)	99	1545	-
0	x(46)	60	7955	-
0	x(47)	59	8581	-
-	x(48)	13	4171	0
0	x(49)	56	6533	-
-	x(50)	20	2517	0

PROBLEME P.1

$x^*(j)$: fixée par la réduction	variables	c(j)	a(j)	$x^*(j)$
-	x(51)	1	7181	1
0	x(52)	24	3259	-
-	x(53)	1	1016	1
0	x(54)	77	3975	-
0	x(55)	88	9864	-
-	x(56)	10	6775	0
-	x(57)	16	8139	0
-	x(58)	4	9149	0
0	x(59)	46	787	-
-	x(60)	2	4986	1
0	x(61)	76	1271	-
0	x(62)	52	7116	-
0	x(63)	39	9271	-
-	x(64)	6	7910	1
0	x(65)	91	9814	-
-	x(66)	11	5342	0
0	x(67)	71	3772	-
0	x(68)	62	3339	-
0	x(69)	65	7254	-
0	x(70)	72	1548	-
0	x(71)	51	264	-
0	x(72)	33	6475	-
0	x(73)	29	6975	-
-	x(74)	20	2485	0
0	x(75)	48	7358	-
0	x(76)	62	7986	-
-	x(77)	1	6745	1
0	x(78)	84	7464	-
0	x(79)	46	7292	-
0	x(80)	40	3197	-
0	x(81)	45	5041	-
0	x(82)	34	8931	-
0	x(83)	92	2382	-
0	x(84)	5	4857	0
-	x(85)	88	2654	-
0	x(86)	41	8813	-
0	x(87)	55	7558	-
0	x(88)	28	4339	-
0	x(89)	85	2620	-
0	x(90)	30	5834	-
0	x(91)	2	6325	1
-	x(92)	88	8646	-
0	x(93)	59	9004	-
0	x(94)	39	4413	-
0	x(95)	66	4968	-
0	x(96)	43	4326	-
0	x(97)	92	5974	-
0	x(98)	70	9863	-
0	x(99)	28	8022	-
0	x(100)	73	3013	-

second membre = 49802

PROBLEME P.2

$x^*(j)$: fixée par la réduction	variables	c(j)	a(j)	$x^*(j)$
	x(1)	93	602	
0	x(2)	18	6998	-
-	x(3)	74	62757	0
-	x(4)	2	24848	0
-	x(5)	43	54533	0
-	x(6)	89	90738	0
0	x(7)	99	230	-
0	x(8)	34	12926	-
-	x(9)	72	80253	0
-	x(10)	86	89176	0
0	x(11)	91	38949	-
0	x(12)	43	40958	-
-	x(13)	1	94762	0
-	x(14)	94	74672	1
0	x(15)	51	69480	-
-	x(16)	88	2015	0
0	x(17)	71	52014	-
-	x(18)	87	9608	0
0	x(19)	23	69475	-
-	x(20)	67	90890	0
-	x(21)	65	80934	0
-	x(22)	31	82484	0
-	x(23)	9	55156	0
-	x(24)	43	3275	0
-	x(25)	57	24956	0
-	x(26)	62	97516	0
0	x(27)	94	19230	-
-	x(28)	2	22798	1
0	x(29)	95	48087	-
-	x(30)	24	63508	0
-	x(31)	34	23048	0
-	x(32)	39	96152	0
-	x(33)	70	71766	0
-	x(34)	65	79562	0
0	x(35)	83	21861	-
-	x(36)	54	8815	0
-	x(37)	35	33379	0
-	x(38)	38	75234	0
-	x(39)	40	15969	0
-	x(40)	33	72566	0
-	x(41)	43	8868	0
0	x(42)	93	28785	-
-	x(43)	67	32556	0
-	x(44)	36	25711	0
-	x(45)	48	75772	0
-	x(46)	40	86673	1
-	x(47)	57	32070	0
-	x(48)	76	62920	0
-	x(49)	62	74279	0
0	x(50)	84	27222	-

PROBLEME P.2

$x^*(j)$: fixée par la réduction	variables	c(j)	a(j)	$x^*(j)$
-	x(51)	70	67078	0
-	x(52)	48	49093	0
-	x(53)	72	80618	0
-	x(54)	67	95745	0
-	x(55)	62	21004	0
0	x(56)	93	64996	-
0	x(57)	90	78355	-
0	x(58)	81	40695	-
0	x(59)	92	25932	-
-	x(60)	32	63933	0
-	x(61)	4	60545	1
-	x(62)	17	40008	0
-	x(63)	16	53418	1
-	x(64)	33	27810	0
-	x(65)	10	93593	0
0	x(66)	82	10114	0
-	x(67)	27	80285	1
-	x(68)	7	95651	0
-	x(69)	65	29883	0
-	x(70)	38	82484	0
-	x(71)	62	6035	0
-	x(72)	35	3739	0
-	x(73)	35	68096	1
-	x(74)	30	89121	0
-	x(75)	51	19328	-
0	x(76)	89	35249	0
-	x(77)	21	1842	0
-	x(78)	44	74253	0
-	x(79)	9	77003	-
0	x(80)	85	55945	0
-	x(81)	1	86342	0
-	x(82)	15	93289	-
0	x(83)	88	33081	0
-	x(84)	55	6990	0
-	x(85)	28	5821	0
-	x(86)	37	76767	-
0	x(87)	86	67477	0
-	x(88)	19	26140	0
-	x(89)	25	30914	0
-	x(90)	24	71158	0
-	x(91)	25	76167	-
0	x(92)	79	54157	0
-	x(93)	74	91244	0
-	x(94)	42	95667	-
0	x(95)	89	85727	0
-	x(96)	31	38008	-
0	x(97)	96	58351	-
0	x(98)	80	14532	-
0	x(99)	78	47691	-
-	x(100)	59	89001	0

second membre = 502968

PROBLEME P.3

$x^*(j)$: fixée par la réduction	variables	c(j)	a(j)	$x^*(j)$
-	x(1)	29	66990	0
0	x(2)	77	35948	-
0	x(3)	58	48108	-
-	x(4)	14	80943	0
0	x(5)	81	60387	-
-	x(6)	26	45380	0
-	x(7)	23	41178	0
0	x(8)	74	62828	-
-	x(9)	10	49097	0
0	x(10)	37	17025	-
-	x(11)	3	61758	1
0	x(12)	46	49435	-
0	x(13)	36	46966	-
0	x(14)	62	14011	-
-	x(15)	34	42093	0
0	x(16)	39	44544	-
-	x(17)	38	91626	0
0	x(18)	47	78625	-
0	x(19)	43	12872	-
0	x(20)	72	30625	-
0	x(21)	86	72201	-
0	x(22)	78	40667	-
-	x(23)	25	41664	0
0	x(24)	39	11442	-
0	x(25)	89	86545	-
0	x(26)	88	68845	-
-	x(27)	27	74365	0
-	x(28)	33	95346	0
0	x(29)	77	41681	-
0	x(30)	48	58458	-
0	x(31)	75	95044	-
0	x(32)	63	48179	-
-	x(33)	10	64695	1
0	x(34)	46	44507	-
-	x(35)	16	35855	0
0	x(36)	93	66604	-
0	x(37)	96	79947	-
0	x(38)	67	23511	-
-	x(39)	10	51542	0
-	x(40)	20	68278	0
0	x(41)	35	34192	-
0	x(42)	92	75073	-
-	x(43)	31	54494	0
0	x(44)	58	36543	-
0	x(45)	65	52332	-
0	x(46)	80	16117	-
0	x(47)	77	89371	-
0	x(48)	77	11395	-
0	x(49)	98	99110	-
0	x(50)	81	10186	-

PROBLEME P.3

$x^*(j)$: fixée par la réduction	variables	c(j)	a(j)	$x^*(j)$
-	x(51)	7	17657	0
-	x(52)	36	92975	0
-	x(53)	9	40751	0
0	x(54)	70	82622	-
-	x(55)	27	82481	1
-	x(56)	25	22417	0
0	x(57)	55	39078	-
0	x(58)	55	68134	-
-	x(59)	21	32863	1
-	x(60)	35	64844	0
0	x(61)	37	49401	-
0	x(62)	88	37110	-
-	x(63)	12	21806	0
0	x(64)	70	50936	-
0	x(65)	71	17160	-
0	x(66)	71	89536	-
-	x(67)	10	21558	0
0	x(68)	71	80099	-
0	x(69)	79	10961	-
0	x(70)	79	88085	-
0	x(71)	90	14976	-
0	x(72)	68	22093	-
0	x(73)	62	56189	-
0	x(74)	46	76118	-
0	x(75)	38	12117	-
-	x(76)	1	86604	1
-	x(77)	1	39485	1
-	x(78)	25	33582	0
-	x(79)	25	40888	0
-	x(80)	6	43649	1
-	x(81)	10	96349	1
0	x(82)	80	10093	-
-	x(83)	9	12290	0
0	x(84)	73	94818	-
0	x(85)	66	78770	-
0	x(86)	91	76892	-
0	x(87)	68	60782	-
-	x(88)	74	21074	0
0	x(89)	52	72626	-
-	x(90)	40	78144	0
-	x(91)	41	99095	0
0	x(92)	74	55069	-
0	x(93)	88	40897	-
0	x(94)	76	25172	-
0	x(95)	53	33880	-
0	x(96)	71	45998	-
0	x(97)	71	16253	-
0	x(98)	20	12608	-
0	x(99)	98	99488	-
0	x(100)	83	84339	-

second membre = 507884

PROBLEME P.4

$x^*(j)$: fixée par la réduction	variables	c(j)	a(j)	$x^*(j)$
-	x(1)	8	73074	1
0	x(2)	99	224965	-
-	x(3)	7	773687	0
-	x(4)	2	130991	0
-	x(5)	30	931832	0
-	x(6)	39	20771	0
-	x(7)	28	742949	0
-	x(8)	32	497652	0
-	x(9)	18	209133	0
0	x(10)	64	62247	-
-	x(11)	67	911354	0
-	x(12)	5	673151	1
-	x(13)	42	2047	0
0	x(14)	74	152438	-
0	x(15)	74	562055	-
-	x(16)	50	563768	0
-	x(17)	45	279803	0
-	x(18)	36	853666	0
-	x(19)	20	189247	0
0	x(20)	93	472969	-
-	x(21)	11	64386	0
0	x(22)	69	569239	-
0	x(23)	68	480645	-
-	x(24)	56	464919	0
0	x(25)	95	273103	-
-	x(26)	25	944376	0
-	x(27)	55	923776	0
-	x(28)	21	836930	0
-	x(29)	54	656674	0
-	x(30)	50	125714	0
-	x(31)	35	119794	0
-	x(32)	37	593399	0
-	x(33)	31	360290	0
0	x(34)	67	414801	-
-	x(35)	15	250934	0
0	x(36)	72	155108	-
-	x(37)	1	48122	0
-	x(38)	34	337651	0
0	x(39)	77	121498	-
-	x(40)	32	963766	0
0	x(41)	89	277461	-
0	x(42)	91	107167	-
-	x(43)	48	573917	0
-	x(44)	23	85913	0
-	x(45)	28	43408	0
0	x(46)	92	140590	-
0	x(47)	86	398342	-
-	x(48)	12	324058	0
0	x(49)	77	231432	-
0	x(50)	97	607792	-

PROBLEME P.4

$x^*(j)$: fixée par la réduction	variables	c(j)	a(j)	$x^*(j)$
-	x(51)	6	170580	1
0	x(52)	98	342991	-
-	x(53)	61	729076	0
-	x(54)	60	471495	0
-	x(55)	5	253860	1
0	x(56)	84	279127	-
-	x(57)	16	472107	0
-	x(58)	22	269334	0
-	x(59)	50	741142	0
-	x(60)	35	648624	0
0	x(61)	90	697646	-
0	x(62)	99	598856	-
-	x(63)	4	98106	0
0	x(64)	95	971658	-
-	x(65)	40	687525	0
-	x(66)	20	579463	0
-	x(67)	60	362872	0
-	x(68)	43	851786	0
-	x(69)	6	959318	1
-	x(70)	53	636891	0
0	x(71)	58	125964	-
-	x(72)	34	433520	0
0	x(73)	93	922713	-
0	x(74)	99	832693	-
0	x(75)	83	212185	-
-	x(76)	15	648788	0
-	x(77)	63	968319	0
-	x(78)	39	340289	0
0	x(79)	93	726438	-
-	x(80)	3	632337	0
0	x(81)	96	502759	-
-	x(82)	35	592358	0
-	x(83)	10	700534	1
-	x(84)	41	99125	0
0	x(85)	96	652885	-
0	x(86)	82	773797	-
0	x(87)	98	255117	-
-	x(88)	26	65543	0
0	x(89)	70	616781	-
-	x(90)	9	290387	1
-	x(91)	26	157708	1
-	x(92)	35	677163	0
-	x(93)	10	553455	1
0	x(94)	85	384743	-
-	x(95)	27	187298	0
-	x(96)	38	827502	0
0	x(97)	79	882038	-
0	x(98)	89	512962	-
-	x(99)	17	746598	1
-	x(100)	18	778182	0

second membre = 4578665

PROBLEME P.5

$x^*(j)$: fixée par la réduction

	variables	c(j)	a(j)	$x^*(j)$
0	x(1)	88	289235	-
0	x(2)	84	108638	-
-	x(3)	60	516893	0
-	x(4)	40	390417	0
0	x(5)	94	616161	-
-	x(6)	19	219580	0
-	x(7)	15	510556	0
-	x(8)	47	845439	0
-	x(9)	14	229910	0
0	x(10)	73	228201	-
-	x(11)	76	954569	0
-	x(12)	74	831877	0
0	x(13)	88	661946	-
-	x(14)	37	316269	0
-	x(15)	14	594878	0
-	x(16)	63	784481	0
0	x(17)	77	177044	-
-	x(18)	33	374336	1
-	x(19)	3	289861	1
-	x(20)	17	204254	0
-	x(21)	21	851083	0
-	x(22)	48	167655	0
0	x(23)	84	515240	-
-	x(24)	28	245708	0
-	x(25)	18	988450	0
0	x(26)	99	916621	-
0	x(27)	91	860111	-
0	x(28)	78	601177	-
-	x(29)	54	547793	0
0	x(30)	90	428939	-
0	x(31)	96	404912	-
0	x(32)	81	637695	-
-	x(33)	49	233633	0
0	x(34)	86	267955	-
-	x(35)	53	666508	0
0	x(36)	78	855306	-
0	x(37)	82	385385	-
0	x(38)	83	607667	-
-	x(39)	35	658836	0
0	x(40)	94	875290	-
-	x(41)	64	232680	0
-	x(42)	61	563977	0
0	x(43)	88	406432	-
-	x(44)	23	938961	0
-	x(45)	31	561405	0
-	x(46)	46	380136	0
-	x(47)	48	310155	0
0	x(48)	97	585601	-
-	x(49)	13	214054	0
-	x(50)	43	777048	0

PROBLEME P.5

$x^*(j)$: fixée par la réduction	variables	c(j)	a(j)	$x^*(j)$
-	x(51)	65	470200	0
-	x(52)	61	595704	0
-	x(53)	10	489159	1
-	x(54)	78	897997	0
-	x(55)	56	510452	0
-	x(56)	6	941754	1
0	x(57)	82	885127	-
-	x(58)	20	323361	0
-	x(59)	49	961978	0
-	x(60)	2	921584	0
0	x(61)	74	586650	-
0	x(62)	96	871645	-
-	x(63)	71	928903	0
-	x(64)	27	441833	0
0	x(65)	72	155257	-
-	x(66)	40	594250	0
-	x(67)	33	661289	1
0	x(68)	73	522591	-
-	x(69)	15	628155	0
-	x(70)	12	718886	1
0	x(71)	94	943309	-
0	x(72)	98	358866	-
-	x(73)	65	467536	0
-	x(74)	29	563708	0
-	x(75)	14	910657	1
-	x(76)	60	166063	0
0	x(77)	81	335667	-
0	x(78)	72	216024	-
-	x(79)	46	467445	0
0	x(80)	80	108921	-
0	x(81)	70	153308	-
-	x(82)	43	498696	0
-	x(83)	59	119418	0
0	x(84)	83	945600	-
-	x(85)	55	691842	0
-	x(86)	41	864396	0
-	x(87)	34	787174	0
0	x(88)	93	526035	-
-	x(89)	1	523657	1
-	x(90)	16	141073	0
0	x(91)	81	824434	-
-	x(92)	67	888686	0
0	x(93)	81	400728	-
0	x(94)	77	526720	-
-	x(95)	22	440795	0
-	x(96)	12	574662	0
-	x(97)	16	641227	0
0	x(98)	94	346096	-
-	x(99)	56	632301	0
0	x(100)	75	523991	-

second membre = 4909599

PROBLEME P.6

$x^*(j)$: fixée par la réduction

	variables	c(j)	a(j)	$x^*(j)$
0	x(1)	31	4205	-
0	x(2)	2	191	-
0	x(3)	20	9264	-
0	x(4)	25	6453	-
0	x(5)	82	11470	-
0	x(6)	64	2657	-
0	x(7)	66	9123	-
0	x(8)	94	16565	-
0	x(9)	67	2886	-
0	x(10)	48	2848	-
0	x(11)	28	18990	-
0	x(12)	28	16263	-
0	x(13)	28	12487	-
0	x(14)	2	4805	-
0	x(15)	30	10997	-
0	x(16)	22	15210	-
0	x(17)	7	1712	-
0	x(18)	63	6096	-
0	x(19)	17	4219	-
0	x(20)	50	2316	-
0	x(21)	38	16690	-
-	x(22)	86	1503	0
0	x(23)	16	9227	-
0	x(24)	39	3237	-
0	x(25)	84	19743	-
0	x(26)	70	18147	-
0	x(27)	75	16891	-
0	x(28)	5	11137	-
0	x(29)	47	9950	-
-	x(30)	85	7309	1
0	x(31)	41	6775	-
0	x(32)	9	11948	-
0	x(33)	96	2969	-
0	x(34)	21	3732	-
0	x(35)	42	12589	-
0	x(36)	6	16784	-
0	x(37)	13	6341	-
0	x(38)	86	11281	-
-	x(39)	95	12418	0
-	x(40)	92	17228	1
0	x(41)	39	2948	-
0	x(42)	8	10310	-
0	x(43)	72	6809	-
0	x(44)	47	18643	-
0	x(45)	1	10253	-
0	x(46)	3	6225	-
0	x(47)	43	4670	-
0	x(48)	12	10791	-
0	x(49)	22	2534	-
0	x(50)	46	15045	-

PROBLEME P.6

$x^*(j)$: fixée par la réduction	variables	c(j)	a(j)	$x^*(j)$
0	x(51)	72	8226	-
-	x(52)	35	11015	1
0	x(53)	25	8647	-
0	x(54)	43	17733	-
0	x(55)	39	9121	-
0	x(56)	83	18705	-
0	x(57)	92	17447	-
0	x(58)	92	4963	-
0	x(59)	73	19155	-
0	x(60)	82	18257	-
0	x(61)	36	10814	-
0	x(62)	63	17147	-
0	x(63)	80	18420	-
-	x(64)	23	7596	1
0	x(65)	24	1227	-
0	x(66)	16	10983	-
-	x(67)	86	12473	0
0	x(68)	50	9390	-
0	x(69)	57	11736	-
0	x(70)	75	13753	-
0	x(71)	30	18740	-
0	x(72)	7	5752	-
0	x(73)	95	8167	-
0	x(74)	19	10304	-
0	x(75)	46	18014	-
0	x(76)	24	1468	-
0	x(77)	78	5237	-
-	x(78)	43	2578	1
0	x(79)	82	8165	-
0	x(80)	1	198	-
0	x(81)	14	1184	-
-	x(82)	99	8859	0
0	x(83)	91	431	-
0	x(84)	75	18791	-
0	x(85)	56	13152	-
-	x(86)	65	16986	1
0	x(87)	58	15270	-
0	x(88)	50	9467	-
0	x(89)	12	9414	-
0	x(90)	1	912	-
0	x(91)	38	16098	-
-	x(92)	35	17526	0
0	x(93)	18	6682	-
0	x(94)	85	9462	-
-	x(95)	83	7573	1
-	x(96)	97	10548	1
0	x(97)	37	12027	-
-	x(98)	65	5468	0
0	x(99)	95	11828	-
0	x(100)	19	9422	-

PROBLEME P.6

$x^*(j)$: fixée par la réduction

	variables	c(j)	a(j)	$x^*(j)$
0	x(101)	80	1461	-
0	x(102)	52	4499	-
-	x(103)	4	15473	1
0	x(104)	43	2619	-
0	x(105)	65	18636	-
0	x(106)	31	415	-
0	x(107)	81	14858	-
0	x(108)	71	9953	-
0	x(109)	33	4182	-
0	x(110)	50	1244	-
0	x(111)	80	18227	-
-	x(112)	11	13463	0
0	x(113)	23	40	-
-	x(114)	5	3048	0
0	x(115)	94	11241	-
0	x(116)	74	11275	-
0	x(117)	79	5596	-
0	x(118)	47	17073	-
0	x(119)	33	3784	-
0	x(120)	16	9459	-
0	x(121)	62	1287	-
0	x(122)	51	11384	-
-	x(123)	1	9612	1
-	x(124)	2	9298	1
0	x(125)	62	5462	-
0	x(126)	58	18887	-
0	x(127)	21	18475	-
0	x(128)	68	16738	-
0	x(129)	63	13133	-
0	x(130)	25	2514	-
0	x(131)	27	2395	-
0	x(132)	67	11867	-
0	x(133)	51	7205	-
0	x(134)	20	8296	-
0	x(135)	22	5018	-
0	x(136)	15	3102	-
0	x(137)	99	962	-
0	x(138)	84	6753	-
0	x(139)	94	2429	-
-	x(140)	7	19275	1
0	x(141)	73	5549	-
0	x(142)	76	2143	-
0	x(143)	54	11478	-
0	x(144)	93	1718	-
0	x(145)	97	868	-
0	x(146)	92	2811	-
0	x(147)	19	7966	-
0	x(148)	28	6481	-
-	x(149)	6	4628	0
0	x(150)	23	12155	-

PROBLEME P.6

$x^*(j)$: fixée par la réduction	variables	c(j)	a(j)	$x^*(j)$
-	x(151)	1	3411	1
0	x(152)	90	6859	-
0	x(153)	79	14581	-
0	x(154)	40	9429	-
-	x(155)	7	5077	1
0	x(156)	50	5582	-
0	x(157)	37	9442	-
0	x(158)	94	5386	-
-	x(159)	8	14822	1
-	x(160)	7	12972	0
0	x(161)	27	13952	-
0	x(162)	84	11977	-
0	x(163)	74	1962	-
0	x(164)	90	19433	-
0	x(165)	44	13750	-
-	x(166)	6	11589	1
-	x(167)	1	7257	1
0	x(168)	87	17035	-
0	x(169)	89	19186	-
-	x(170)	1	12737	1
0	x(171)	14	2519	-
0	x(172)	90	8670	-
0	x(173)	93	18454	-
-	x(174)	6	16653	1
0	x(175)	39	4243	-
-	x(176)	9	12975	1
0	x(177)	26	19366	-
0	x(178)	95	6805	-
0	x(179)	72	14528	-
0	x(180)	67	12646	-
0	x(181)	20	10055	-
0	x(182)	26	11847	-
0	x(183)	47	14010	-
0	x(184)	10	1982	-
0	x(185)	88	13057	-
0	x(186)	34	15475	-
0	x(187)	69	5102	-
0	x(188)	80	1310	-
0	x(189)	68	12335	-
0	x(190)	74	5807	-
0	x(191)	64	3154	-
0	x(192)	36	13543	-
0	x(193)	41	11069	-
0	x(194)	45	7694	-
-	x(195)	3	3745	1
0	x(196)	55	16550	-
0	x(197)	72	17640	-
0	x(198)	33	10259	-
0	x(199)	64	14931	-
0	x(200)	93	15563	-

second membre = 192429

PROBLEME P.7

$x^*(j)$: fixée par la réduction	variables	c(j)	a(j)	$x^*(j)$
0	x(1)	31	4205	-
-	x(2)	2	191	1
1	x(3)	20	9264	-
-	x(4)	25	6453	1
0	x(5)	82	11470	-
0	x(6)	64	2657	-
0	x(7)	66	9123	-
0	x(8)	94	16565	-
0	x(9)	67	2886	-
0	x(10)	48	2848	-
1	x(11)	28	18990	-
1	x(12)	28	16263	-
1	x(13)	28	12487	-
1	x(14)	2	4805	-
1	x(15)	30	10997	-
1	x(16)	22	15210	-
-	x(17)	7	1712	0
0	x(18)	63	6096	-
-	x(19)	17	4219	0
0	x(20)	50	2316	-
1	x(21)	38	16690	-
0	x(22)	86	1503	-
1	x(23)	16	9227	-
0	x(24)	39	3237	-
-	x(25)	84	19743	0
-	x(26)	70	18147	0
0	x(27)	75	16891	-
1	x(28)	5	11137	-
0	x(29)	47	9950	-
0	x(30)	85	7309	-
0	x(31)	41	6775	-
1	x(32)	9	11948	-
0	x(33)	96	2969	-
-	x(34)	21	3732	0
-	x(35)	42	12589	1
1	x(36)	6	16784	-
1	x(37)	13	6341	-
0	x(38)	86	11281	-
0	x(39)	95	12418	-
0	x(40)	92	17228	-
0	x(41)	39	2948	-
1	x(42)	8	10310	-
0	x(43)	72	6809	-
1	x(44)	47	18643	-
1	x(45)	1	10253	-
1	x(46)	3	6225	-
0	x(47)	43	4670	-
1	x(48)	12	10791	-
0	x(49)	22	2534	-
1	x(50)	46	15045	-

PROBLEME P.7

$x^*(j)$: fixée par la réduction	variables	c(j)	a(j)	$x^*(j)$
0	x(51)	72	8226	-
-	x(52)	35	11015	1
1	x(53)	25	8647	-
1	x(54)	43	17733	-
-	x(55)	39	9121	0
0	x(56)	83	18705	-
0	x(57)	92	17447	-
0	x(58)	92	4963	-
-	x(59)	73	19155	1
0	x(60)	82	18257	-
-	x(61)	36	10814	1
-	x(62)	63	17147	1
0	x(63)	80	18420	-
-	x(64)	23	7596	1
0	x(65)	24	1227	-
1	x(66)	16	10983	-
0	x(67)	86	12473	-
0	x(68)	50	9390	-
0	x(69)	57	11736	-
0	x(70)	75	13753	-
1	x(71)	30	18740	-
1	x(72)	7	5752	-
0	x(73)	95	8167	-
1	x(74)	19	10304	-
1	x(75)	46	18014	-
0	x(76)	24	1468	-
0	x(77)	78	5237	-
0	x(78)	43	2578	-
0	x(79)	82	8165	-
-	x(80)	1	198	1
0	x(81)	14	1184	-
0	x(82)	99	8859	-
0	x(83)	91	431	-
-	x(84)	75	18791	0
-	x(85)	56	13152	0
-	x(86)	65	16986	0
-	x(87)	58	15270	1
0	x(88)	50	9467	-
1	x(89)	12	9414	-
-	x(90)	1	912	1
1	x(91)	38	16098	-
1	x(92)	35	17526	-
1	x(93)	18	6682	-
0	x(94)	85	9482	-
0	x(95)	83	7573	-
0	x(96)	97	10548	-
1	x(97)	37	12027	-
0	x(98)	65	5468	-
0	x(99)	95	11828	-
1	x(100)	19	9422	-

PROBLEME P.7

$x^*(j)$: fixée par la réduction	variables	c(j)	a(j)	$x^*(j)$
0	x(101)	80	1461	-
0	x(102)	52	4499	-
1	x(103)	4	15473	-
0	x(104)	43	2619	-
-	x(105)	65	18636	1
0	x(106)	31	415	-
0	x(107)	81	14858	-
0	x(108)	71	9953	-
0	x(109)	33	4182	-
0	x(110)	50	1244	-
0	x(111)	80	18227	-
1	x(112)	11	13463	-
0	x(113)	23	40	-
-	x(114)	5	3048	1
0	x(115)	94	11241	-
0	x(116)	74	11275	-
0	x(117)	79	5596	-
1	x(118)	47	17073	-
0	x(119)	33	3784	-
1	x(120)	16	9459	-
0	x(121)	62	1287	-
-	x(122)	51	11384	0
1	x(123)	1	9612	-
1	x(124)	2	9298	-
0	x(125)	62	5462	-
1	x(126)	58	18887	-
1	x(127)	21	18475	-
-	x(128)	68	16738	1
0	x(129)	63	13133	-
0	x(130)	25	2514	-
0	x(131)	27	2395	-
0	x(132)	67	11867	-
0	x(133)	51	7205	-
1	x(134)	20	8296	-
-	x(135)	22	5018	1
-	x(136)	15	3102	0
0	x(137)	99	962	-
0	x(138)	84	6753	-
0	x(139)	94	2429	-
1	x(140)	7	19275	-
0	x(141)	73	5549	-
0	x(142)	76	2143	-
0	x(143)	54	11478	-
0	x(144)	93	1718	-
0	x(145)	97	868	-
0	x(146)	92	2811	-
1	x(147)	19	7966	-
-	x(148)	28	6481	0
1	x(149)	6	4628	-
1	x(150)	23	12155	-

PROBLEME P.7

$x^*(j)$: fixée par la réduction	variables	c(j)	a(j)	$x^*(j)$
1	x(151)	1	3411	-
0	x(152)	90	6859	-
0	x(153)	79	14581	-
-	x(154)	40	9429	0
1	x(155)	7	5077	-
0	x(156)	50	5582	-
-	x(157)	37	9442	1
0	x(158)	94	5386	-
1	x(159)	8	14822	-
1	x(160)	7	12972	-
1	x(161)	27	13952	-
0	x(162)	84	11977	-
0	x(163)	74	1962	-
0	x(164)	90	19433	-
1	x(165)	44	13750	-
1	x(166)	6	11589	-
1	x(167)	1	7257	-
0	x(168)	87	17035	-
0	x(169)	89	19186	-
1	x(170)	1	12737	-
-	x(171)	14	2519	0
0	x(172)	90	8670	-
0	x(173)	93	18454	-
1	x(174)	6	16653	-
0	x(175)	39	4243	-
1	x(176)	9	12975	-
1	x(177)	26	19366	-
0	x(178)	95	6805	-
0	x(179)	72	14528	-
0	x(180)	67	12646	-
1	x(181)	20	10055	-
1	x(182)	26	11847	-
-	x(183)	47	14010	1
-	x(184)	10	1982	0
0	x(185)	88	13057	-
1	x(186)	34	15475	-
0	x(187)	69	5102	-
0	x(188)	80	1310	-
0	x(189)	68	12335	-
0	x(190)	74	5807	-
0	x(191)	64	3154	-
1	x(192)	36	13543	-
-	x(193)	41	11069	1
0	x(194)	45	7694	-
1	x(195)	3	3745	-
1	x(196)	55	16550	-
-	x(197)	72	17640	0
-	x(198)	33	10259	1
-	x(199)	64	14931	0
0	x(200)	93	15563	-

second membre = 962148

PROBLEME S.1

variables	c(j)	a(j)	$x^*(j)$
x(1)	4205	4205	1
x(2)	191	191	1
x(3)	9264	9264	1
x(4)	6453	6453	1
x(5)	11470	11470	1
x(6)	2657	2657	1
x(7)	9123	9123	1
x(8)	16565	16565	1
x(9)	2886	2886	1
x(10)	2848	2848	1
x(11)	18990	18990	1
x(12)	16263	16263	1
x(13)	12487	12487	1
x(14)	4805	4805	1
x(15)	10997	10997	1
x(16)	15210	15210	1
x(17)	1712	1712	0
x(18)	6096	6096	0
x(19)	4219	4219	1
x(20)	2316	2316	1
x(21)	16690	16690	0
x(22)	1503	1503	0
x(23)	9227	9227	1
x(24)	3237	3237	0
x(25)	19743	19743	0
x(26)	18147	18147	1
x(27)	16891	16891	0
x(28)	11137	11137	1
x(29)	9950	9950	0
x(30)	7309	7309	0
x(31)	6775	6775	0
x(32)	11948	11948	0
x(33)	2969	2969	1
x(34)	3732	3732	0
x(35)	12589	12589	0
x(36)	16784	16784	0
x(37)	6341	6341	0
x(38)	11281	11281	0
x(39)	12418	12418	0
x(40)	17228	17228	0
x(41)	2948	2948	0
x(42)	10310	10310	0
x(43)	6809	6809	0
x(44)	18643	18643	0
x(45)	10253	10253	0
x(46)	6225	6225	0
x(47)	4670	4670	0
x(48)	10791	10791	0
x(49)	2534	2534	0
x(50)	15045	15045	0

PROBLEME S.1

variables	c(j)	a(j)	$x^*(j)$
x(51)	8226	8226	0
x(52)	11015	11015	0
x(53)	8647	8647	0
x(54)	17733	17733	0
x(55)	9121	9121	0
x(56)	18705	18705	0
x(57)	17447	17447	0
x(58)	4963	4963	0
x(59)	19155	19155	0
x(60)	18257	18257	0
x(61)	10814	10814	0
x(62)	17147	17147	0
x(63)	18420	18420	0
x(64)	7596	7596	0
x(65)	1227	1227	0
x(66)	10983	10983	0
x(67)	12473	12473	0
x(68)	9390	9390	0
x(69)	11736	11736	0
x(70)	13753	13753	0
x(71)	18740	18740	0
x(72)	5752	5752	0
x(73)	8167	8167	0
x(74)	10304	10304	0
x(75)	18014	18014	0
x(76)	1468	1468	0
x(77)	5237	5237	0
x(78)	2578	2578	0
x(79)	8165	8165	0
x(80)	198	198	0
x(81)	1184	1184	0
x(82)	8859	8859	0
x(83)	431	431	0
x(84)	18791	18791	0
x(85)	13152	13152	0
x(86)	16986	16986	0
x(87)	15270	15270	0
x(88)	9467	9467	0
x(89)	9414	9414	0
x(90)	912	912	0
x(91)	16098	16098	0
x(92)	17526	17526	0
x(93)	6682	6682	0
x(94)	9482	9482	0
x(95)	7573	7573	0
x(96)	10548	10548	0
x(97)	12027	12027	0
x(98)	5468	5468	0
x(99)	11828	11828	0
x(100)	9422	9422	0

PROBLEME S.1

variables	c(j)	a(j)	$x^*(j)$
x(101)	1461	1461	
x(102)	4499	4499	0
x(103)	15473	15473	0
x(104)	2619	2619	0
x(105)	18636	18636	0
x(106)	415	415	0
x(107)	14858	14858	0
x(108)	9953	9953	0
x(109)	4182	4182	0
x(110)	1244	1244	0
x(111)	18227	18227	0
x(112)	13463	13463	0
x(113)	40	40	0
x(114)	3048	3048	0
x(115)	11241	11241	0
x(116)	11275	11275	0
x(117)	5596	5596	0
x(118)	17073	17073	0
x(119)	3784	3784	0
x(120)	9459	9459	0
x(121)	1287	1287	0
x(122)	11384	11384	0
x(123)	9612	9612	0
x(124)	9298	9298	0
x(125)	5462	5462	0
x(126)	18887	18887	0
x(127)	18475	18475	0
x(128)	16738	16738	0
x(129)	13133	13133	0
x(130)	2514	2514	0
x(131)	2395	2395	0
x(132)	11867	11867	0
x(133)	7205	7205	0
x(134)	8296	8296	0
x(135)	5018	5018	0
x(136)	3102	3102	0
x(137)	962	962	0
x(138)	6753	6753	0
x(139)	2429	2429	0
x(140)	19275	19275	0
x(141)	5549	5549	0
x(142)	2143	2143	0
x(143)	11478	11478	0
x(144)	1718	1718	0
x(145)	868	868	0
x(146)	2811	2811	0
x(147)	7966	7966	0
x(148)	6481	6481	0
x(149)	4628	4628	0
x(150)	12155	12155	

PROBLEME S.1

variables	c(j)	a(j)	$x^*(j)$
x(151)	3411	3411	
x(152)	6859	6859	0
x(153)	14581	14581	0
x(154)	9429	9429	0
x(155)	5077	5077	0
x(156)	5582	5582	0
x(157)	9442	9442	0
x(158)	5386	5386	0
x(159)	14822	14822	0
x(160)	12972	12972	0
x(161)	13952	13952	0
x(162)	11977	11977	0
x(163)	1962	1962	0
x(164)	19433	19433	0
x(165)	13750	13750	0
x(166)	11589	11589	0
x(167)	7257	7257	0
x(168)	17035	17035	0
x(169)	19186	19186	0
x(170)	12737	12737	0
x(171)	2519	2519	0
x(172)	8670	8670	0
x(173)	18454	18454	0
x(174)	16653	16653	0
x(175)	4243	4243	0
x(176)	12975	12975	0
x(177)	19366	19366	0
x(178)	6805	6805	0
x(179)	14528	14528	0
x(180)	12646	12646	0
x(181)	10055	10055	0
x(182)	11847	11847	0
x(183)	14010	14010	0
x(184)	1982	1982	0
x(185)	13057	13057	0
x(186)	15475	15475	0
x(187)	5102	5102	0
x(188)	1310	1310	0
x(189)	12335	12335	0
x(190)	5807	5807	0
x(191)	3154	3154	0
x(192)	13543	13543	0
x(193)	11069	11069	0
x(194)	7694	7694	0
x(195)	3745	3745	0
x(196)	16550	16550	0
x(197)	17640	17640	0
x(198)	10259	10259	0
x(199)	14931	14931	0
x(200)	15563	15563	0

second membre = 192429

PROBLEME S.2

variables	c(j)	a(j)	$x^*(j)$
x(1)	4205	4205	1
x(2)	191	191	1
x(3)	9264	9264	1
x(4)	6453	6453	1
x(5)	11470	11470	1
x(6)	2657	2657	1
x(7)	9123	9123	1
x(8)	16565	16565	1
x(9)	2886	2886	1
x(10)	2848	2848	1
x(11)	18990	18990	1
x(12)	16263	16263	1
x(13)	12487	12487	1
x(14)	4805	4805	1
x(15)	10997	10997	1
x(16)	15210	15210	1
x(17)	1712	1712	1
x(18)	6096	6096	1
x(19)	4219	4219	1
x(20)	2316	2316	1
x(21)	16690	16690	1
x(22)	1503	1503	1
x(23)	9227	9227	1
x(24)	3237	3237	1
x(25)	19743	19743	1
x(26)	18147	18147	1
x(27)	16891	16891	1
x(28)	11137	11137	1
x(29)	9950	9950	1
x(30)	7309	7309	1
x(31)	6775	6775	1
x(32)	11948	11948	1
x(33)	2969	2969	1
x(34)	3732	3732	1
x(35)	12589	12589	1
x(36)	16784	16784	1
x(37)	6341	6341	1
x(38)	11281	11281	1
x(39)	12418	12418	1
x(40)	17228	17228	1
x(41)	2948	2948	1
x(42)	10310	10310	1
x(43)	6809	6809	1
x(44)	18643	18643	1
x(45)	10253	10253	1
x(46)	6225	6225	1
x(47)	4670	4670	1
x(48)	10791	10791	1
x(49)	2534	2534	1
x(50)	15045	15045	

PROBLEME S.2

variables	c(j)	a(j)	$x^*(j)$
x(51)	8226	8226	1
x(52)	11015	11015	1
x(53)	8647	8647	1
x(54)	17733	17733	1
x(55)	9121	9121	1
x(56)	18705	18705	1
x(57)	17447	17447	1
x(58)	4963	4963	1
x(59)	19155	19155	1
x(60)	18257	18257	1
x(61)	10814	10814	1
x(62)	17147	17147	1
x(63)	18420	18420	1
x(64)	7596	7596	1
x(65)	1227	1227	1
x(66)	10983	10983	1
x(67)	12473	12473	1
x(68)	9390	9390	1
x(69)	11736	11736	1
x(70)	13753	13753	1
x(71)	18740	18740	1
x(72)	5752	5752	1
x(73)	8167	8167	1
x(74)	10304	10304	1
x(75)	18014	18014	1
x(76)	1468	1468	1
x(77)	5237	5237	1
x(78)	2578	2578	1
x(79)	8165	8165	1
x(80)	198	198	1
x(81)	1184	1184	1
x(82)	8859	8859	1
x(83)	431	431	1
x(84)	18791	18791	1
x(85)	13152	13152	1
x(86)	16986	16986	1
x(87)	15270	15270	1
x(88)	9467	9467	1
x(89)	9414	9414	0
x(90)	912	912	1
x(91)	16098	16098	0
x(92)	17526	17526	1
x(93)	6682	6682	0
x(94)	9482	9482	1
x(95)	7573	7573	1
x(96)	10548	10548	1
x(97)	12027	12027	0
x(98)	5468	5468	1
x(99)	11828	11828	1
x(100)	9422	9422	1

PROBLEME S.2

variables	c(j)	a(j)	$x^*(j)$
x(101)	1461	1461	1
x(102)	4499	4499	0
x(103)	15473	15473	1
x(104)	2619	2619	0
x(105)	18636	18636	0
x(106)	415	415	0
x(107)	14858	14858	0
x(108)	9953	9953	0
x(109)	4182	4182	0
x(110)	1244	1244	0
x(111)	18227	18227	0
x(112)	13463	13463	0
x(113)	40	40	0
x(114)	3048	3048	0
x(115)	11241	11241	0
x(116)	11275	11275	0
x(117)	5596	5596	0
x(118)	17073	17073	0
x(119)	3784	3784	0
x(120)	9459	9459	0
x(121)	1287	1287	0
x(122)	11384	11384	0
x(123)	9612	9612	0
x(124)	9298	9298	0
x(125)	5462	5462	0
x(126)	18887	18887	0
x(127)	18475	18475	0
x(128)	16738	16738	0
x(129)	13133	13133	0
x(130)	2514	2514	0
x(131)	2395	2395	0
x(132)	11867	11867	0
x(133)	7205	7205	0
x(134)	8296	8296	0
x(135)	5018	5018	0
x(136)	3102	3102	0
x(137)	962	962	0
x(138)	6753	6753	0
x(139)	2429	2429	0
x(140)	19275	19275	0
x(141)	5549	5549	0
x(142)	2143	2143	0
x(143)	11478	11478	0
x(144)	1718	1718	0
x(145)	868	868	0
x(146)	2811	2811	0
x(147)	7966	7966	0
x(148)	6481	6481	0
x(149)	4628	4628	0
x(150)	12155	12155	0

PROBLEME S.2

variables	c(j)	a(j)	$x^*(j)$
x(151)	3411	3411	0
x(152)	6859	6859	0
x(153)	14581	14581	0
x(154)	9429	9429	0
x(155)	5077	5077	0
x(156)	5582	5582	0
x(157)	9442	9442	0
x(158)	5386	5386	0
x(159)	14822	14822	0
x(160)	12972	12972	0
x(161)	13952	13952	0
x(162)	11977	11977	0
x(163)	1962	1962	0
x(164)	19433	19433	0
x(165)	13750	13750	0
x(166)	11589	11589	0
x(167)	7257	7257	0
x(168)	17035	17035	0
x(169)	19186	19186	0
x(170)	12737	12737	0
x(171)	2519	2519	0
x(172)	8670	8670	0
x(173)	18454	18454	0
x(174)	16653	16653	0
x(175)	4243	4243	0
x(176)	12975	12975	0
x(177)	19366	19366	0
x(178)	6805	6805	0
x(179)	14528	14528	0
x(180)	12646	12646	0
x(181)	10055	10055	0
x(182)	11847	11847	0
x(183)	14010	14010	0
x(184)	1982	1982	0
x(185)	13057	13057	0
x(186)	15475	15475	0
x(187)	5102	5102	0
x(188)	1310	1310	0
x(189)	12335	12335	0
x(190)	5807	5807	0
x(191)	3154	3154	0
x(192)	13543	13543	0
x(193)	11069	11069	0
x(194)	7694	7694	0
x(195)	3745	3745	0
x(196)	16550	16550	0
x(197)	17640	17640	0
x(198)	10259	10259	0
x(199)	14931	14931	0
x(200)	15563	15563	0

second membre = 962148

BIBLIOGRAPHIE

[1] AHRENS H., FINKE G. (1975) "*Merging and sorting applied to the zero-one knapsack problem*". OperationsResearch 23 (6), pp. 1099-1109.

[2] ANTHONISSE J.M. (1973) "*A note on equivalent systems of linear diophantine equation*". Operations Research 17, pp. 167-177.

[3] BALAS E., ZEMEL E. (1980) "*An algorithm for large zero-one knapsack problem*". Operations Research 28, pp. 1130-1154.

[4] BALINSKI M.L. (1965) "*Integer programming ; Methods, Uses, Computations*". Management Science 12 (3), pp. 253-313.

[5] BARR R.S., ROSS G.T. (1975) "*A linked data structure for a binary knapsack algorithm*". Research Report of the Center for Cybernetic Stuies, Uneversity of Texas.

[6] BEALE E.L., TOMLIN J.A. (1972) "*An integer programming approach to class of combinational problem*". Mathematical Programming 3, pp. 339-344.

[7] BELLMAN R. (1954) "*Some applications of the theory of dynamic programming, a review*". Operations Research 2, pp. 275-288.

[8] BELLMAN R. (1957) "*Dynamic programming and the numerical solution of variational problem*". Operations Research 5, pp. 277-288.

[9] BOWMAN V.J., NEMHAUSER G.L. (1971) "*Deep cuts in integer programming*". Operations Research 8, pp. 89-111.

[10] BRADLEY G.H. (1971) "*Transformation of integer programs to knapsack problems*". Discrete Mathematics 1, pp. 29-45.

[11] BRADLEY G.H. (1971) "*Equivalent integer programs and canonical problems*". Management Science 17, pp. 354-366.

[12] BRADLEY G.H., HAMMER P.L., WOLSEY L. (1974) "*Coefficient reduction for inequalities in 0-1 variables*". Mathematical Programming 7, pp. 263-282.

[13] CHANG S.K., GILL A. (1970)*Algorithmic solution of the change making problem*". Journal of the Association for Computing machinery 17 (1), pp. 113-122.

[14] CHVATAL V., HAMMER P.L. (1977) "*Aggregation of inequalities in integer programming*". Annals of Discrete Mathematics 1, pp. 145-162.

[15] DANTZING G.B. (1957) "*Discret variable extremum problems*". Operations Research 5, pp. 266-288.

[16] FAALAND B. (1973) "*Solution of the value - Independent knapsack problem by partitioning*". Operations Research 21, pp. 332-337.

[17] FAYARD D., PLATEAU G. (1975) "*Resolution of the 0-1 knapsack problem : Comparaison of methods*". Mathematical Programming 8 (3), pp. 272-307.

[18] FAYARD D., PLATEAU G. (1979). Thèse de Docteur-ès-Sciences Mathématiques. Université des Sciences et Techniques de Lille.

[19] FAYARD D., PLATEAU G. (1982) "*An algorithm for the solution of the 0-1 knapsack problem*". Computing 28, pp. 269-287.

[20] FAYARD D., PLATEAU G. (1982) "*An algorithm for the collapsing 0-1 knapsack problem*". Mathematics for Computer Science, pp. 397-406.

[20 bis] GAREY M.R., JOHNSON D.S. (1979) "*Computers and intractability : a Guid to the theory of NP-Completeness*". Freeman, San Francisco

[21] GARFINKEL R.S., NEMHAUSER G.L. (1972) "*Integer programming*". A Wiley-Interscience Publication, John Wiley and Sons.

[22] GEOFFRION A.M., GRAVES G.W. (1974) "*Multicommodity distribution system design by benders decomposition*". Management Science 20, pp. 822-844.

[23] GILMORE P.C., GOMORY R.E. (1965) "*Multistage cutting stock problems of two and more dimensions*". Operations Research 13, pp. 94-120.

[24] GILMORE P.C., GOMORY R.E. (1966) "*The theory and computation of knapsack functions*". Operations Research 14, pp. 1045-1074.

[25] GLOVER F. (1975) "*New results on equivalents integer formulations*". Mathematical Programming 8, pp. 84-90.

[26] GLOVER F., WOOLSEY R.E. (1972) "*Aggregation diophantine constraintes*". Zeitschrift für Operations Research 16, pp. 1-10.

[27] GREENBERG H., HEGERICH R.L. (1970) "*A branch search algorithm for the knapsack problem*". Management Science 16 (5), pp. 327-332.

[28] GROWDER H., JOHNSON E.L., PADBERG M. (1982) "*Solving large-scale zero-one linear programming problems*". IBM Thomas, J. Watson Research.

[29] HAYASHI Y. (1980) "*Reduction methods in the 0-1 knapsack problem*". Graduate School of Management, University of Rochester, NY 14627.

[30] HERLESTAM T. (1978) "*Critical remarks on some public - key cryptosystems*". BIT 18, pp. 493-496.

[31] HOROWITZ E., SAHNI S. (1974) "*Computing partitions with application to the knapsack problems*". Journal of the Association for Computing Machinery 21, pp. 275-292.

[32] HU T. (1969) "*Integer programming and network flow*". Reading, Mass. Addison-Wesley.

[33] INGARGIOLA G.P., KORSH J.F. (1973) "*Reduction algorithm for the zero-one single knapsack problems*". Management Science 20 (4), pp. 460-663.

[34] INGEMARSON I. (1980) "*Knapsack which are not partly solvable after multiplication Modulo q*". IBM Thomas J. Wetson Research Center.

[35] JOHNSON E.L. (1974) "*On the group problem for mixed integer programming*". Mathematical Programming Study 2, pp. 137-179.

[36] KALISZEWSKI I., LIBURA M. (1977) *"Constraints aggregation in integer programming"*. Report MPD 5-77, Systems Research Institue, Polish Academy of Sciences, Varsovie (Pologne).

[37] KANNAN R. (1983) *"Polynomial - time aggregation of integer programming problem"*. I.A.C.M. 30 (1), pp. 133-145.

[38] KARNIN E.D. (1984) *"A parallel algorithm for the knapsack problem"*. IEEE Transaction on Computers C-33 (5), pp. 404-408.

[39] KENDAL K.E., ZIONTS S. (1977) *"Solving integer programming problems by aggregation constraints"*. Operations Research 25 (2), pp. 346-351.

[40] KIANFAR F. (1971) *"Stronger inequalities for 0-1 integer programming using knapsack functions"*. Operations Research 19, pp. 1374-1392.

[41] KIANFAR F. (1976) *"Stronger inequalities for 0-1 integer programming computational refinements"*. ORSA 24 (3), pp. 581-585.

[42] KOLESAR P.J. (1967) *"A branch and bound algorithm for the knapsack problem"*. Management Science 13 (9), pp. 723-735.

[43] KOSTREVA M.M., JOHNSON E.L. (1982) *"Solving 0-1 integer programming problem arising from large scale planning models"*. Research Publication GMR-3990, Mathematics Departement, Warren Michigan.

[44] LAGARIAS J.C., ODLYZKO A.M. (1984) *"Solving low-Density subset sum problems"*. Bell Laboratories Murray Hill, New Jersey 07974.

[45] LAURIERE N. (1978) *"An algorithm for the 0-1 knapsack problem"*. Mathematical Programming 14, pp. 1-10.

[46] LAURIERE J.L. (1979) *"Elements de programmation dynamique"*. Collection Programmation. Gauthier - Villars.

[47] LAWLER E.L. (1979) *"Fast approximation algorithms for knapsack problems"*. Mathematics of Operation Research 4, pp. 339-356.

[48] MAGAZINE M.J., OGUZ O. (1981) "*A fully polynomial approximate algorithm for the 0-1 knapsack problem*". European Journal of Operations Research 8 (3), pp. 270-273.

[49] MARTELLO S., TOTH P. (1977) "*An upper bound for the zero-one knapsack problem and branch and bound algorithm*". European Journal of Operational Research 1, pp. 169-175.

[50] MARTELLO S., TOTH P. (1979) "*The 0-1 knapsack problem in : combinatorial optimisation*". "Christofides N., Mingozzi A., Sandi C. Toth P., eds), London : J. Wiley.

[51] MARTELLO S., TOTH P. (1982) "*A mixed algorithm for the subset - sum problem*". EURO V. TIMS XXV, Lausanne.

[52] MERKLE R.C., HELLMAN M.E. (1978) "*Hiding information and signatures in trapdoor knapsack*". IEEE Transaction on Information Theory 24, pp. 525-530.

[53] MEYER R.R. (1979) "*Equivalent constraits for discrete sets*". Discrete Applied Mathematics 1 (1), pp. 31-50.

[54] MINOUX M. (1983) "*Programmation mathématique*". Théorie et algorithmes. Tome 2, Edition DUNOD.

[55] MORIN T.L., MARSTEN R.E. (1976) "*Branch and bound strategies for dynamic programming*". Operations Research 24, pp. 611-627.

[56] MORIN T.L., MARSTEN R.E. (1976) "*An algorithm for non linear knapsack problems*". Management Science 22 (10), pp. 1147-1158.

[57] MORIN T.L., MARSTEN R.E. (1978) "*Hybrid approach to discret mathematical programming*". Mathematical Programming 14, pp. 21-40.

[58] NAUSS R.M. (1976) "*An efficient algorithm for the 0-1 knapsack problem*". Management Science 23 (1), pp. 27-31.

[59] NEMHAUSER G.L., ULLMAN (1969) *"Discret dynamic programming and capital allocation"*. Management Science 15 (9), pp. 494-505.

[60] PADBERG M.W. (1972) *"Equivalent knapsack type formulations of bounded integer linear programs : an alternative approach"*. Naval Research Logistic Quarterly 19, pp. 699-708.

[61] PLATEAU G., GUERCH M.T. (1984) *"Aggregation of equalities in integer programming"*. Lecture Notes in Control and Information Science, Proceeding of the 11th IFIP (conference Copenhagen Springer-Verlag), pp. 183-192.

[62] PETERSEN C.C. (1967) *"Computational experience with variants of the balas algorithm applied to the selection of R & D projets"*. Management Science 13 (9), pp. 736-750.

[63] PROSCHAN F., BRAY T.A. (1965) *"Optimal redundancy under multiple constraints"*. Operations Research 13, pp. 800-814.

[64] RIBEIRO C. (1983) Thèse de Docteur Ingénieur en Informatique. Ecole Nationale Supérieure des Télécommunications, Paris.

[65] ROSENBERG I.G. (1974) *"Aggregation of equations in integer programming"*. Discrete Mathematics 10, pp. 325-341.

[66] SALKIN H.M. (1975) *"Integer programming"*. Addison - Wesley Publishing Company.

[67] SHAPIRO J.F., WAGNER H.M. (1967) *"A finite renewal algorithm for the knapsack and turnpike models"*. Operations Research 15, pp. 319-341.

[68] SHAPIRO J.F. (1968) *"Dynamic programming algorithms for the integer programming problem viewed as a knapsack type problem"*. Operation Research 16, pp. 103-121.

[69] SHAPIRO J.F. (1971) *"Generalised Lagrange multiplier in integer programming"*. Operations Research 19, pp. 69-76.

[70] TOTH P. (1980) *"Dynamic programming algorithms for the zero-one knapsack problem"*. Computing 25, pp. 29-45.

[71] WALUKIEWICZ S., KALISZEWSKI I. (1976) *"Tighter equivalent formulations of integer programming problems"*. A. Prekopa. Ed. Survey of Mathematical Programming Proceeding of the IX International Mathematical Programming Symposium Budapest.

[72] WEINGARTNER H.M., NESS D. (1967) *"Method for the solution of the Multi-dimensional 0-1 knapsack problem"*. Operations Research 15 (1) pp. 83-103.

[73] WILLIAMS H.P. (1974) *"Experiments in the formulation of integer programming problems"*. Mathematical Programming Study 2, pp. 180-190.

[74] WILSON (1983) *"A method of reducing coefficients in integer linear inequalities"*. Naval Research Logistic Quarterly 30, pp. 49-57.

[75] WOLSEY L.A. (1973) *"Generalised dynamic programming methods integer programming"*. Mathematical Programming 4, pp. 222-232.

[76] WRIGHT J.W. (1975) *"The chang-making problem"*. Journal of the Association for Computing Machinery 22 (1), pp. 125-128.

[77] ZOLTNERS A.A. (1978) *"A direct descent binary knapsack problem"*. Journal of the Association for Computing Machinery 25 (2), pp. 304-311.