MASTER EN SCIENCES INFORMATIQUES

Quasi-optimalité en programmation linéaire simple

Coutisse, Bernard

Award date:
1985

Awarding institution:
Universite de Namur

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR
INSTITUT D'INFORMATIQUE

QUASI-OPTIMALITE EN PROGRAMMATION LINEAIRE SIMPLE

Mémoire présenté par

Bernard COUTISSE

en vue de l'obtention du grade de Licencié et Maître en Informatique.

Année académique 1984-1985

TABLE DES MATIERES

INTRODUCTION

Ce mémoire se base essentiellement sur l'article de J.SISKOS (Operations Research; vol.18, n°4, novembre 1984, p.381 à 401).

Le but de cet article est triple :

1° poser le problème des solutions optimales multiples en programmation linéaire en le formulant dans un contexte plus général, celui de la recherche des solutions quasi-optimales;

2° décrire deux approches représentatives et complémentaires et montrer leurs limites;

3° présenter succintement deux approches heuristiques qui évitent les complications algorithmiques des méthodes exactes.

Dans un premier chapitre, ce travail rappelle les principales notions de programmation linéaire simple en proposant en plus de l'approche algébrique classique, une approche géométrique qui non seulement aide à percevoir plus profondément de nombreux aspects du problème, mais aussi permet de guider les démarches algébriques et même de démontrer naturellement de nombreux théorèmes.

Nous présentons, au deuxième chapitre, le problème des solutions optimales multiples et quasi-optimales. Le problème des solutions optimales multiples est très frquemment rencontré tant dans les applications pratiques que théoriques. On le posera en le plaçant dans un contexte plus général, celui de la recherche des solutions quasi-optimales.

Vient au troisième chapitre, la description de la méthode du simplexe inverse. L'algorithme général évitant les problèmes liés à la dégénérescence y est décrit en détails et est suivi de l'algorithme classique qui ne se préoccupe pas du cyclage éventuel. Nous illustrons la méthode par un exemple puis nous la critiquons.

Au quatrième chapitre, nous abordons les méthodes de labyrinthe. Nous donnons un aperçu de ces méthodes et nous développons l'une d'entre elles : celle de MANAS et NEDOMA représentative des méthodes de pivotage et dont l'efficacité est reconnue.

Le cinquième et dernier chapitre présente le logiciel de programmation linéaire XMP. Ce logiciel est orienté recherche algorithmique et se présente comme un outil bien adapté à notre problème.

Dans l'annexe, nous concevons concrètement l'algorithme de MANAS et NEDOMA en tenant compte des possibiliés FORTRAN-XMP.

CHAPITRE I : RAPPELS DE PROGRAMMATION LINEAIRE SIMPLE.

1.1. Présentation du problème.

Le problème général de programmation linéaire simple se présente de la manière suivante :

$$\text{opt } z = \sum_{j=1}^{r} c_j x_j \quad ,$$

fonction économique ou objective à optimiser,
sous contraintes

$$\sum_{j=1}^{r} a_{ij} x_j \left\{ \begin{matrix} \geqslant \\ = \\ \leqslant \end{matrix} \right\} b_i \qquad (i = 1, \dots, q)$$

$$x_j \geqslant 0 \qquad (j = 1, \dots, p)$$

$$x_j \text{ quelconque} \qquad (j = p+1, \dots, q)$$

et peut toujours se ramener à l'une des deux formes :

forme standard
$\min z = \sum_{j=1}^{n} c_j x_j$
s.c.
$\sum_{j=1}^{n} a_{ij} x_j = b_i$
$x_j \geqslant 0$
$(i = 1, \dots, m)$
$(j = 1, \dots, n)$

forme canonique
$\min z = \sum_{j=1}^{n} c_j x_j$
s.c.
$\sum_{j=1}^{n} a_{ij} x_j \geqslant b_i$
$x_j \geqslant 0$
$(i = 1, \dots, m)$
$(j = 1, \dots, n)$

ou en notations matricielles :

forme standard	forme canonique
$\min z = cx$ s.c. $Ax = b \quad (1)$ $x \geq 0 \quad (2)$	$\min z = cx$ s.c. $Ax \geq b \quad (1)$ $x \geq 0 \quad (2)$

où x est le vecteur colonne $(x_1, \ldots, x_n)'$,
c le vecteur ligne $(c_1, \ldots, c_n)$ $(\neq (0, \ldots, 0))$,
b le vecteur colonne $(b_1, \ldots, b_m)'$,
A la matrice (a_{ij}) ($i = 1, \ldots, m$ et $j = 1, \ldots, n$) dont la i^e colonne est notée P_i
et $\underline{0}$ le vecteur colonne nul à n composantes.

Un vecteur x qui satisfait à (1) est dit solution du PLS (');
de plus si x satisfait à (2), il est dit solution réalisable.
Une solution optimale du PLS est une solution réalisable pour laquelle il n'existe pas de meilleures solutions réalisables (c'est à dire de solutions réalisables qui rendent la valeur de la fonction économique plus petite (respectivement grande) (cas min (respectivement max))).

(') PLS : problème de programmation linéaire simple.

1.2. Interprétation géométrique du PLS (').

Plaçons-nous dans l'espace affin euclidien de dimension n, E_n , muni du repère canonique.

Dans ce repère, l'ensemble des points dont les coordonnées x vérifient Ax = b est une variété affine V de dimension n - rang (A). Les points de V sont en bijection avec les solutions du PLS.

Si on se limite à l'orthant positif, on obtient l'ensemble des points dont les coordonnées satisfont aux contraintes. Cet ensemble, noté X , est appelé le tronçon (") des contraintes (ou polyèdre des contraintes si X est borné).
Les points de X sont en bijection avec l'ensemble des solutions réalisables.

L'ensemble des points dont les coordonnées x dans le repère canonique vérifient $cx = z$ $(z \in R)$ est un hyperplan, noté H, pour lequel le vecteur c de composantes $(c_1, \ldots, c_n)$ dans la base canonique est un vecteur normal, pointant dans le sens opposé à celui d'une minimisation.

Lorsque z varie, on obtient une famille d'hyperplans parallèles et il s'agit de déterminer, si possible, le ou les points du tronçon des contraintes pour lesquel(s) z est optimum.

Un dessin svp

(') On se limitera à la forme standard.

('') tronçon = intersection d'un nombre fini de demi-espaces fermés;
polyèdre = tronçon non vide borné;
tronçons et polyèdres sont convexes.

1.3. Solutions de base et points extrémaux.

Dans la suite, nous supposerons toujours que le système $Ax = b$ est compatible et indépendant, ce qui revient à supposer que rang $(A) = m \leq n$. Cette hypothèse n'est pas restrictive dans la mesure où l'application des méthodes de résolution des PLS nous amène toujours à partir d'une forme standard compatible et indépendante (grâce à d'éventuels artifices).

1.3.1. Solutions de base.

Une base du PLS est un sous-ensemble ordonné de m P_j indépendants. Par extension, on dit aussi que la sous-matrice B ayant pour colonnes ces m P_j est une base du PLS. Les m variables associés à ces m P_j sont dites variables de base, les autres sont dites hors base.

Après renumérotation éventuelle et après avoir posé :

$$A = (B \mid N),$$

$$x = \begin{pmatrix} x_B \\ x_N \end{pmatrix},$$

$$c = (c_B \mid c_N),$$

on a $Ax = b$

$$\iff x_B + B^{-1}Nx_N = B^{-1}b . \qquad (1.1)$$

On pose encore

$$B^{-1}A = (I \mid \bar{P}_{m+1}, \dots, \bar{P}_n) = (\bar{a}_{ij}) ;$$

$$B^{-1}b = \bar{b} ;$$

et on obtient une solution particulière

$$x = \begin{pmatrix} x_B \\ x_N \end{pmatrix} = \begin{pmatrix} \bar{b} \\ \underline{0} \end{pmatrix}$$

appelée solution de base associée à B.

Elle est dite réalisable si $x \geq \underline{0}$ c'est à dire si $\bar{b} \geq \underline{0}$, B étant alors une base réalisable ;

dégénérée si elle est réalisable et non $\bar{b} > \underline{0}$,

optimale si elle est réalisable et si elle donne la valeur optimale à la fonction économique, B étant alors une base optimale.

1.3.2. Point extrémal.

D'un point de vue géométrique, on définit un point extrémal P d'un convexe X comme étant un point qui ne peut appartenir à un segment d'extrémités P_1 et P_2 avec P_1 et P_2 points de X différents de P.

Les points c. et d. du théorème suivant mettent en relation les solutions de base et les points extrémaux.

1.4. Théorème fondamental de la programmation linéaire simple (1).

Etant donné un PLS sous forme standard :

a. S'il existe une solution réalisable, il existe une solution de base réalisable.
b. S'il existe une solution optimale, il existe une solution de base optimale.
c. A une solution de base réalisable correspond un point extrémal du tronçon des contraintes.
d. A tout point extrémal du tronçon des contraintes correspond une solution de base réalisable.

Remarque : Les points c. et d. n'impliquent pas la correspondance biunivoque entre les solutions de base réalisables et les points extrémaux du tronçon; la correspondance n'est bi-univoque qu'en l'absence de dégénérescence.

Par conséquent, si un PLS a une solution optimale, il possède une solution de base optimale qui est un point extrémal du tronçon X.

Il est facile de voir aussi que toute combinaison linéaire convexe de solutions optimales est encore une solution optimale, ce qui signifie d'un point de vue géométrique que l'enveloppe convexe de points extrémaux optimaux est un ensemble de points optimaux.

---

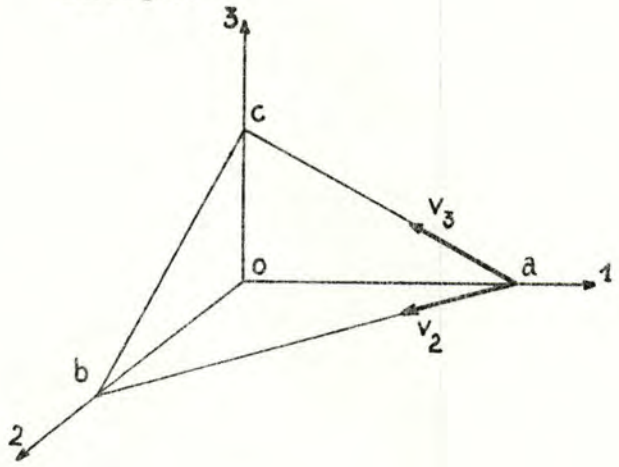
(1) Démonstration : voir [2] p.II.33 à II.37.

On se limitera donc à la recherche de solutions de base optimales.

Un PLS peut admettre 0,1 ou une infinité de solutions optimales.

Géométriquement, le cas d'une infinité de solutions optimales apparaît lorsque l'hyperplan H défini par la fonction économique est parallèle à une variété affine V' contenue dans V (on aura au plus une p-uple infinité de solutions optimales si dim (V')=p avec V' la variété de dimension la plus grande contenue dans V et parallèle à H).

Exemple :



si $z_1 = c_1 x$ avec c_1 ppd v_3
on aura soit une solution b
soit une simple infinité de solutions (segment ac).

si $z_2 = c_2 x$ avec c_2 ppd v_2 et c_2 ppd v_3,
on aura une double infinité de solutions (triangle abc et son intérieur).

1.5. Algorithme primal du simplexe.

1.5.1. Approche algébrique.

L'algorithme primal du simplexe est destiné au PLS sous forme standard pour lesquels on connaît une base réalisable B et son inverse B^{-1}. Il se distingue de la méthode du simplexe destinée à tout PLS sous forme standard.

Rappelons-le brièvement sans démonstration (').

(') L'interprétation géométrique qui suit ne fournit pas une démonstration rigoureuse mais une preuve intuitive. On peut trouver une démonstration dans [2] p.III.5 à III.14.

Se basant sur le théorème fondamental, son but est, partant d'une solution réalisable, d'obtenir, si possible, une solution optimale de base du PLS (et non pas toutes les solutions optimales (de base)).('')

Le PLS est équivalent, vu (1.1) au PLS en (x,z)

$$\min\ z$$

$$\text{s.c.} \begin{cases} -z + c_B x_B + c_N x_N = 0 \\ x_B + B^{-1} N x_N = \bar{b} \\ x \geqslant \underline{0} \end{cases}$$

$$\Longleftrightarrow$$

$$\min\ z$$

$$\text{s.c.} \begin{cases} -z + (c_N - c_B B^{-1} N)\, x_N = -c_B \bar{b} \\ x_B + B^{-1} N x_N = \bar{b} \\ x \geqslant \underline{0} \end{cases}$$

Posons

$$\bar{c} = (\bar{c}_B \mid \bar{c}_N) = c - c_B B^{-1} A = (\underline{0}' \mid c_N - c_B B^{-1} N)$$

$$\bar{z}_o = c_B \bar{b}$$

ou en introduisant $\pi = c_B B^{-1}$ (vecteur qui joue un rôle important)

$$\bar{c} = (\underline{0}' \mid c_N - \pi N)$$

$$\bar{z}_o = \pi b$$

Les $\bar{c}_j$ sont appelés coûts relatifs correspondant à la base B et les π_i les multiplicateurs de simplexe.

$x = (x_B' \mid x_N')' = (\bar{b}' \mid \underline{0}')'$ est la solution de base associée à la base B et donne à la fonction économique la valeur Z.

Le dernier PLS est appelé l'équivalent canonique associé au PLS original et permet de construire le tableau-simplexe de départ :

('') Le problème des solutions multiples sera abordé dans la suite.

	x_1	...	x_m	x_{m+1}	...	x_n
	$-\bar{z}_0$	0 ...	0	$\bar{c}_{m+1}$	...	$\bar{c}_n$
x_1	$\bar{b}_1$	1	0			
.		. .			$\bar{a}_{ij}$	
.						
x_m	$\bar{b}_m$	0	1			

Voici l'algorithme.
Soit un tableau-simplexe.

Si $\bar{c}_N \geq 0$

alors
x = $(\bar{b}' \; \underline{0}')'$ est une solution de base minimale;
si $\bar{c}_N > 0$
alors
cette solution est unique;
sinon
problème des solutions optimales multiples;(')
Fin du programme;
sinon
choisir $\bar{c}_s < 0$, par exemple $\bar{c}_s = \min \{ \bar{c}_j \}$;
C On dit que x_s entre dans la base
si $\bar{P}_s \leq 0$
alors pas de minimum fini;
sinon
calculer $\min \{ \bar{b}_i / \bar{a}_{is} \} = \bar{b}_r / \bar{a}_{rs}$
pour i indice d'une variable de base tel que $\bar{a}_{is} > 0$;

(') On y reviendra.

C ‖ On dit que x_r sort de la base.
‖ Pivoter le tableau-simplexe avec $\bar{a}_{rs}$ pour pivot;
C ‖ On note ce pivotage $(x_s \uparrow x_r \downarrow)$.
‖ Retour au début.

L'algorithme s'arrête après un nombre fini d'itérations si, à chacune d'elles, il n'y a pas dégénérescence. Sinon, il se peut que l'algorithme cycle. Pour éviter un cyclage éventuel, il est toujours possible d'utiliser une méthode, par exemple la méthode lexicographique (').

Dans la suite, on supposera toujours l'utilisation d'une méthode évitant le cyclage. Ainsi, à chaque itération-simplexe on obtient, si le problème admet un optimum fini, une solution de base réalisable non encore obtenue.

Notations : Lorsqu'on se réfère à une itération-simplexe, on note B, la base de laquelle on part et $\bar{B}$ la base à laquelle on arrive.

1.5.2. Approche géométrique ('').

Soit un tableau-simplexe et une base réalisable B.

La variété définie en (1.2) a pour équation paramétrique, dans le repère canonique $(O;\vec{e}_1,\dots,\vec{e}_n)$: $\begin{pmatrix} x_B \\ x_N \end{pmatrix} = \begin{pmatrix} \bar{b} \\ 0 \end{pmatrix} + \lambda_{m+1}\begin{pmatrix} -\bar{p}_{m+1} \\ e_1 \end{pmatrix} + \dots + \lambda_n \begin{pmatrix} -\bar{p}_n \\ e_{n-m} \end{pmatrix}$.

Posons :

$\bar{P}_o$ le point de coordonnées, dans le repère canonique, $(\bar{b}', \underline{0}')'$,

$\bar{e}_i$, le vecteur de composantes, dans la base canonique, $(a_{i1},\dots a_{in})' \quad (i = 1,\dots,m)$,

$\bar{e}_{m+i}$, le vecteur de composantes, dans la base canonique, $(-\bar{p}_i' \mid \underbrace{0\dots1}_{i}\dots 0)' \quad (i = 1,\dots,n-m)$.

$(\bar{e}_1,\dots,\bar{e}_m)$ forme une base de l'espace vectoriel engendrant une variété orthagonale à V,

$(\bar{e}_{m+1},\dots,\bar{e}_n)$ une base de l'espace vectoriel engendrant V,

et donc $(\bar{P}_o;\bar{e}_1,\dots,\bar{e}_n)$ peut être considéré comme nouveau repère.

('') Elle illustre l'approche algébrique et la complète notamment en posant le problème des solutions quasi-optimales.

(') Voir [2] p.III.19 à III.24; résumé en 1.6.

La matrice de changement de base est :

$$S = \left[\begin{array}{c|c} B' & -B^{-1}N \\ \hline N' & I \end{array}\right]$$

et les coordonnées d'un point dans ces deux repères (soit $\bar{x}$ dans l'ancien et x dans le nouveau repère) sont reliées par la relation suivante :

$$x = (\bar{b}' \mid \underline{0}')' + S\,\bar{x}$$

Le PLS s'exprime facilement dans ce nouveau repère :

les contraintes sont :

$$\begin{cases} \bar{x}_B = \underline{0} \qquad \text{(équation de v)} \\ \begin{pmatrix} \bar{b} \\ 0 \end{pmatrix} + \bar{x}_{m+1} \begin{pmatrix} -\bar{p}_{m+1} \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \ldots + \bar{x}_n \begin{pmatrix} -\bar{p}_n \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \geq 0 \qquad (1.2) \end{cases}$$

(équivalent de $x \geq \underline{0}$)

(en particulier $\bar{x}_N \geq \underline{0}$)

et la fonction économique est :

$$z = (c_B \mid c_N) \begin{pmatrix} \bar{b} \\ \underline{0} \end{pmatrix} + (c_B \mid c_N) \left(\begin{array}{c|c} B' & -B^{-1}N \\ \hline N' & I \end{array}\right) \begin{pmatrix} \bar{x}_B \\ \bar{x}_N \end{pmatrix}$$

(or $\bar{x}_B = \underline{0}$)

$$= c_B\,\bar{b} + (c_N - c_B\,B^{-1}N)\,\bar{x}_N$$

$$= \bar{\pi}\,b + \bar{c}_N\,\bar{x}_N \qquad (1.3)$$

$\bar{P}_o$ est un point extrémal du tronçon X puisqu'il lui correspond une base réalisable et vu la forme de $\bar{e}_{m+i}$ qui a n-m-1 composantes nulles, $\bar{P}_o + \bar{x}_{m+i}\,\bar{e}_{m+i}$ est une arête du tronçon X.

On a aussi $c.\bar{e}_s = c_s - c_B\bar{P}_s = \bar{c}_s$, et donc, vu la signification de c, $\bar{c}_s$ caractérise la possibilité de décroissance de la fonction économique en se déplaçant le long de l'arête $\bar{P}_o + \bar{x}_s\,\bar{e}_s$, arête qu'on appellera arête s. La relation (1.3) confirme ce fait (c'est à dire $\bar{c}_s > 0$ (respectivement $<$, $=$) implique que la fonction économique croît (respectivement décroît, reste constante) en se déplaçant le long de l'arête s.

La relation (1.2) permet de dire dans quelle mesure x peut varier.

Partant de là, l'idée de l'algorithme du simplexe est la suivante :

se trouvant en un point extrémal, essayer de se déplacer le long d'une arête jusqu'à un point extrémal adjacent avec l'espoir de faire décroître la fonction économique.

Plus précisément,
étant en un point extrémal (algébriquement, on a un tableau - simplexe et une solution de base réalisable),

si $\bar{c}_N \geq 0$
alors
le déplacement le long d'une arête quelconque n'entraîne pas une diminution de la valeur de la fonction économique et donc le point extrémal courant est un point minimum;

si $\bar{c}_N > 0$
alors
il est évidemment unique;
sinon
problème des solutions optimales multiples;
ce qui suit n'est pas un algorithme mais seulement quelques remarques :
si $\bar{c}_s$, composante de $\bar{c}_N$, $= 0$
alors
(1.3) montre que le déplacement le long de l'arête s ne modifie pas la valeur de la fonction économique;
si tous les $\bar{a}_{is} \leq 0$,
alors
(1.2) est vérifié pour tout x;
on peut donc se déplacer le long de l'arête s jusqu'à l'infini;
on est en présence d'un problème de solutions optimales multiples non bornées;
sinon

(1.2) montre qu'on ne pourra se déplacer sur l'arête s jusqu'à l'infini et donc, qu'à une distance finie (stance nulle dans le cas d'une dégénérescence), on rencontrera un point extrémal; plus précisément (1.2) montre qu'on devra prendre $\bar{x}_s = \min \{ \bar{b}_i / \bar{a}_{is} \}$ pour i tel que $\bar{a}_{is} > o$;

sinon

choisir $\bar{c}_s < o$, par exemple $\bar{c}_s = \min_j \{ \bar{c}_j \}$;
le déplacement le long de l'arête s entraîne une décroissance stricte de la fonction économique;
si tous les $\bar{a}_{is} \leq o$,

alors

(1.2) est vérifié pour tout x; on peut donc se déplacer le long de l'arête s jusqu'à l'infini;
il n'y a donc pas de minimum fini;
sortir du programme;

sinon

(1.2) montre qu'on ne pourra se déplacer sur l'arête s jusqu'à l'infini et donc, qu'à une distance finie (distance nulle dans le cas d'une dégénérescence), on rencontrera un point extrémal; plus précisément, (1.2) montre qu'on devra prendre $\bar{x}_s = \min \{ \bar{b}_i / \bar{a}_{is} \}$ pour i tel que $\bar{a}_{is} > o$;
écrire le problème dans le nouveau repère, ce qui revient à pivoter le tableau - simplexe avec a pour pivot; (')
retour au début.

Remarque : sans calcul, (1.3) montre que z décroît de $|\bar{c}_s \bar{b}_r / \bar{a}_{rs}|$ (=0 si dégénérescence) et (1.2) montre que $\tilde{P}_o$, le nouveau point extrémal a pour composantes, dans le repère canonique:

(') Il s'agit d'exprimer la matrice de changement de repère $(o; e_1, \ldots, e_n) \longrightarrow (\tilde{P}_o; \tilde{e}_1, \ldots, \tilde{e}_n)$ puis de là, la matrice de changement de repère $(\bar{P}_o; \bar{e}_1, \ldots, \bar{e}_n) \longrightarrow (\tilde{P}_o; \tilde{e}_1, \ldots, \tilde{e}_n)$.

$$\begin{pmatrix} \bar{b} \\ \hline \underline{o} \end{pmatrix} + \frac{\bar{b}_r}{\bar{a}_{rs}} \begin{pmatrix} -\bar{P}_s \\ \hline o \\ \vdots \\ 1 \\ \vdots \\ o \end{pmatrix} .$$

1.6. Méthode lexicographique.(')

Cette méthode ordonne les bases réalisables d'un PLS conformément à un ordre strict dit lexicographique.

On dit que :

- un vecteur v est lexicographiquement positif, et on écrit $v \succ \underline{o}$, si $v \neq \underline{o}$ et si sa première composante non nulle est positive;
- un vecteur v est lexicographiquement supérieur à un vecteur w si $v - w \succ \underline{o}$.

La méthode lexicographique est identique à l'algorithme du simplexe à ceci près que :

- dans le tableau initial, ranger les colonnes de telle façon que toutes les lignes, sauf éventuellement celle correspondant à la fonction économique, constituent des vecteurs lexicographiquement positifs;
- le critère de sortie est remplacé par le suivant : s étant l'indice de la variable entrant dans l'ensemble des variables de base, sélectionner pour variable sortante, celle dont l'indice r correspond au vecteur qui, parmi les vecteurs $(\bar{b}_i / \bar{a}_{is}, \bar{a}_{i1} / \bar{a}_{is}, \ldots, \bar{a}_{in} / \bar{a}_{is})$, avec $\bar{a}_{is} > o$, $i \in \{1, \ldots, m\}$, est lexicographiquement inférieur à tous les autres.

On peut montrer que la ligne correspondant à la fonction économique croît lexicographiquement à chaque itération, ce qui assure le non-cyclage.

(') Pour plus de détails, voir [2] p.III.19 à III.24.

CHAPITRE II : SOLUTIONS OPTIMALES MULTIPLES ET QUASI-OPTIMALES D'UN PLS.

2.1. Position du problème.

Au paragraphe 1.5., on a vu qu'à l'issue de l'algorithme du simplexe se pose le problème des solutions optimales multiples lorsqu'il existe au moins un coût relatif associé à une variable hors base qui est nul. L'approche géométrique (1.5.2.) par exemple, montre clairement qu'une itération-simplexe avec introduction dans la base d'une telle variable fournit une nouvelle solution optimale (bornée ou non).(')

On ne peut pourtant pas déterminer d'avance le nombre de solutions multiples auxquelles on peut ainsi parvenir par une poursuite arbitraire de l'algorithme du simplexe. Aussi va-t-on essayer de présenter des méthodes d'exploration systématique.

On posera toutefois ce problème dans un cadre plus général, celui des solutions quasi-optimales, c'est-à-dire de solutions pour lesquelles la valeur de la fonction économique ne diffère de la valeur optimale que d'une "petite"quantité k.

Ce problème est intéressant d'un point de vue théorique et aussi pratique.

Les domaines théoriques les plus affectés sont :

- la théorie des jeux à deux personnes et à somme nulle où il existe très souvent non pas une seule stratégie optimale mais plusieurs [3] ;

(') Rappel: on a une nouvelle solution optimale, mais celle-ci ne correspond pas nécessairement à un point extrémal non encore obtenu (" déplacement nul" si dégénérescence). De plus, cette solution peut correspondre à un point à l'infini (on parle alors de solution non bornée).

- le "goal programming", domaine nouveau de la PL [4] caractérisé par la recherche de solutions qui doivent être les plus compatibles possible avec un système de contraintes (goals);
- la régression ordinale de type quantitatif et/ou qualitatif [5], où le codage des variables que l'on cherche à ajuster est loin d'être unique ;
- la théorie des graphes (le problème de transport par exemple), bien que bon nombre de ces problèmes se formalisent en termes de programmes linéaires en nombres entiers;
- ...

D'un point de vue pratique, on sait ce qu'il en est, lors de modélisations de problèmes réels, de la validité des contraintes et de la fonction économique : le choix des contraintes, l'estimation des coefficients d'un PL comportent une trop grande part d'arbitraire pour qu'on ne s'attache qu'à la détermination d'une solution optimale.

2.2. Définition du problème.

Soit le PLS

$$\min z = c\,x$$

$$\text{s.c} \begin{cases} A\,x = b \\ x \geq o \end{cases}$$

pour lequel on suppose avoir déterminé une solution optimale x^o pour laquelle $c\,x^o = z^o$.

Le problème des solutions quasi-optimales (') consiste à déterminer toutes ('') les solutions du système suivant :

$$\begin{cases} A\,x = b \\ c\,x \leq z^o + k \\ x \geq o \end{cases}$$

(k: constance $\geq o$) (notons X' l'ensemble de E_n associé).

(') On dira aussi solutions k - optimales.

('') Certaines méthodes ne déterminent que certaines solutions (on y reviendra).

La recherche de toutes les solutions optimales se présente comme un cas particulier de ce problème en prenant k=o.

D'un point de vue géométrique, puisque X' est un œconvexe fermé, il s'agira de déterminer :

- si X' est borné, tous les points extrémaux; l'ensemble recherché est alors l'enveloppe convexe de ces points.
- si X' n'est pas borné, X' est le tronçon X tronqué non borné; il s'agirait de déterminer les points extrémaux et la direction des arêtes s'étendant à l'infini.

CHAPITRE III : LA METHODE DU SIMPLEXE INVERSE.

3.1. Itération-simplexe inverse.

Géométriquement, il est évident que toute itération-simplexe $(x_s \uparrow x_r \downarrow)$ possède une itération-simplexe inverse $(x_r \uparrow x_s \downarrow)$ qui restitue le tableau-simplexe existant avant l'itération $(x_s \uparrow x_r \downarrow)$: il s'agit tout simplement d'effectuer le changement de repère inverse, ou de se déplacer dans le sens inverse sur l'arête qu'on vient de parcourir (remarque : déplacement nul compris).

Précisons algébriquement cette approche.

Soit une itération-simplexe $(x_s \uparrow x_r \downarrow)$

Essayons de réaliser un pivotage inversant le rôle de la variable qui entre dans la base avec celui de celle qui en sort. Essayons donc d'introduire dans la base la variable x_r. Puisque le coût relatif associé au pivotage $(x_s \uparrow x_r \downarrow)$, $\tilde{c}_r = -\bar{c}_s/\bar{a}_{rs}$ est strictement positif, ce pivotage fera donc croître la valeur de la fonction économique.

Pour respecter les contraintes et obtenir une autre solution réalisable, la relation (1.2) (par exemple), écrite dans la base $\tilde{B}$ montre que la variable qui sortira de la base sera celle associée à l'indice j réalisant $\min \{ \tilde{b}_i / \tilde{a}_{ir} \}$ avec i, indice d'une variable de la base $\tilde{B}$ telle que $\tilde{a}_{ir} > o$.

On a :

$$\tilde{b}_s / \tilde{a}_{sr} = (\bar{b}_r / \bar{a}_{rs}) / (1 / \bar{a}_{rs}) = \bar{b}_r$$

et pour $i \neq s$ avec x_i variable de $\tilde{B}$ et $\tilde{a}_{ir} > o$.

$$\tilde{b}_i / \tilde{a}_{ir} = \frac{\bar{b}_i - \bar{a}_{is}\bar{b}_r / \bar{a}_{rs}}{-\bar{a}_{is} / \bar{a}_{rs}} \quad (\text{on a donc aussi } a \quad o)$$

$$= \bar{b}_r - \bar{a}_{rs}\, \bar{b}_i / \bar{a}_{is} \quad \geq \bar{b}_r$$

On peut donc effectuer le pivotage inverse $(x_r \uparrow x_s \downarrow)$ qui, bien sûr (') restitue le tableau-simplexe.

(') Géométriquement, évident; algébriquement, faire le pivotage.

3.2. Idée de la méthode du simplexe inverse.

Partant de la signification géométrique d'une itération-simplexe inverse, l'idée de cette méthode est la suivante : se trouvant en un point extrémal minimum, on va essayer de parcourir des arêtes dans le sens contraire d'une minimisation jusqu'à d'autres points extrémaux "pas trop écartés" (').On fera de même à partir de chaque point obtenu. On rangera ces points par ordre croissant de la valeur de la fonction économique associée.

Algorithmiquement, cette méthode apparaît comme une méthode arborescence qui consiste à rechercher des solutions de base quasi-optimales en les rangeant selon les valeurs croissantes de z.

Elle cherche donc à effectuer des pivotages inverses appropriés en allant progressivement de z^o à $z^o + k$.

Pour ne pas s'éloigner de plus de k de z^o, il convient d'introduire une contrainte supplémentaire:

$$c\,x \leq z^o + k$$

ou pour se placer dans les conditions d'application de l'algorithme du simplexe de définir aussi une variable d'écart, y :

$$c\,x + y = z^o + k$$
$$y \geq o$$

(remarque : en fait $o \leq y \leq k$, $y = k$ correspond à une solution optimale du PLS de départ,
$y = o$ correspond aux solutions les plus écartées).

(') On appellera H, l'hyperplan d'équation $c\,x = z^o$
et H', l'hyperplan d'équation $c\,x = z^o + k$.
On dira d'un point extrémal qu'il n'est "pas trop écarté" s'il est situé entre H et H'.

On doit donc compléter le tableau-simplexe optimal par une ligne correspondant à l'adjonction de la nouvelle contrainte et par une colonne nécessaire à la nouvelle variable y.

Plus précisément :

si B est une base optimale,

$\begin{pmatrix} B & \vdots & \underline{0} \\ \cdots & \cdots & \cdots \\ \underline{0}' & \vdots & 1 \end{pmatrix}$ est une base optimale du PLS complété dont la solution de base optimale associée est $(x'_B , x'_N , y)' = (\bar{b}' , \underline{0}' , k)'$:

et dans cette base, la contrainte supplémentaire

$c_B x_B + c_N x_N + y = z^o + k$

s'écrit

$c_B(\bar{b} - B^{-1} N x_N) + c_N x_N + y = c_B \bar{b} + k$

$(c_N - c_B B^{-1} N) x_N + y = k$

$\bar{c}_N x_N + y = k$

et le tableau-simplexe devient :

		x_1 ... x_m	x_{m+1} ... x_n	y
$-z$	z^o	0 ... 0	$\bar{c}_{m+1}$ ... $\bar{c}_n$	0
x_1	$\bar{b}_1$	1 ... 0		0
.	.	. . .		.
.	.	. . .	$\bar{a}_{ij}$	.
.	.	. . .		.
x_m	$\bar{b}_m$	0 ... 1		0
y	k	0 ... 0	$\bar{c}_{m+1}$ ... $\bar{c}_n$	1

Au paragraphe 3.3., on travaillera sur le PLS complété dont le tableau est présenté ci-dessus. On rebaptisera les $\bar{b}$ (resp. $\bar{a}$) qui désigneront donc aussi bien $\bar{b}$ (resp. $\bar{a}$) relatifs aux x ou k (resp. $\bar{c}$) relatif à y. On posera $y = x_{n+1}$ et on parlera de solutions de base et de points extrémaux relatifs au PLS complété (sauf mention du contraire). Logiquement il n'y a aucune raison de distinguer les points extrémaux compris strictement entre H et H'et ceux appartenant à H'et de là, de faire jouer à y un autre rôle qu'aux x (').

(') Dans l'article [illegible], on distingue y des autres variables x, ce qui amène des complications et des oublis (minimums pouvant ne pas exister, problème des doubles non correctement traités).

3.3. La méthode du simplexe inverse.

3.3.1. Notations.

$\bar{c}_s^j$, $\bar{a}_{rs}^j$, $\bar{b}_r^j$ sont les $\bar{c}_s$, $\bar{a}_{rs}$, $\bar{b}_r$ relatifs au tableau-simplexe complété numéro j.

B^j est la base réalisable relative au point extrémal P_j.

3.3.2. Spécifications.

Arguments : un PLS
un tableau-simplexe optimal
un réel $k \geqslant 0$

pré : $\bar{c}_N > 0$

résultat : SOL, la liste de toutes les solutions de base (donc finies) k - optimales.

3.3.3. Remarques.

1. La condition d'applicabilité est donc :
le PLS a une solution optimale unique finie x^0.

2. Vu la condition d'applicabilité, cette méthode ne peut servir à déterminer toutes les solutions optimales éventuelles d'un PLS.

3. Si malgré tout, on l'applique à un PLS ayant des solutions optimales multiples, on n'obtiendra qu'un certain nombre de solutions quasi-optimales, celles qu'on peut atteindre à partir de la solution optimale considérée en faisant des itérations-simplexe inverses augmentant strictement la fonction économique (').

(') Comme on le verra lors de la présentation de l'algorithme.

3.3.4. Définition des variables.

k_j : valeur de y pour le tableau-simplexe numéro j
($o < k_j \leq k$).

k_{ls} : valeur de y, si à partir de P_l , on se déplace sur l'arête s jusqu'au point extrémal adjacent.

K : liste (') des k_{ls} courants :
l : indice d'un point extrémal déjà visité.
s: indice d'une variable telle que $\bar{c}_s > o$ et l'arête s correspondante issue de P_l non encore parcourue.

j,p : numéro de tableau-simplexe.

On veut que les tableaux correspondent à des points extrémaux différents.

Dans le cas où l'on néglige la dégénérescence, il est facile d'ordonner les tableaux-simplexe :
soit le tableau n° o : tableau optimum, solution optimale de base B^o: (x^o, k), point extrémal P_o , $Z = Z^o$;
étant donné le tableau n°j : solution de base réalisable $B^j : (x^j, k)$, point extrémal P_j, $z = z^j$ $(= z^o + k_j)$;
On parlera du tableau n° j + 1 : solution de base réalisable $B^{j+1} (\neq B^j) : (x^{j+1}, k)$, point extrémal P_{j+1}, $z = z^{j+1}$ $(= z^o + k_{j+1})$;
pour lequel $z^j \leq z^{j+1} < z^o + k$, (on ne numérote plus les tableaux pour lesquels $z = z^o + k$ puisqu'il n'auront pas

(') Liste quelconque avec répétitions éventuelles pour laquelle on se permet d'utiliser des notations ensemblistes du genre " $\{a\} \cup \{a\} = \{a, a\}$ ".

de suivant (')).

Si on obtient un point déjà obtenu, on n'a pas besoin de retenir le tableau-simplexe associé puisqu'il est déjà repris dans les j premiers;
ceci ne serait pas vrai si le point extrémal déjà obtenu était dégénéré, car il se pourrait qu'il corresponde à une base différente.

Si on veut tenir compte de la dégénérescence, le problème se complique.
Dans l'algorithme du simplexe, on risquait le cyclage lorsqu'on atteignait un point extrémal dégénéré : les itérations-simplexe pouvaient fournir, tout en restant au même point, une base déjà obtenue.
Ceci peut donc apparaître également lors d'itérations-simplexe inverses.
Mais dans la méthode simplexe inverse, le risque est doublé puisque non seulement on peut étant en un point extrémal dégénéré retomber sur une base obtenue, mais aussi atteindre un point extrémal dégénéré à partir de plusieurs autres points. Quel(s) tableau(x) retenir?
Pour éviter le premier genre de risques, il suffit de penser à une méthode lexicographique et à ne retenir que le dernier tableau-simplexe obtenu.
Naturellement, pour éviter le second genre de problèmes, lorsqu'on atteint un point extrémal déjà obtenu, on retient le tableau-simplexe pour lequel la ligne de la fonction économique est lexicographiquement minimale (et non maximale car $\bar{c}_s > o$).
Ainsi, on évite le cyclage puisqu'on l'évite pour chaque point de même valeur de z et qu'on interdit la communication entre points de même valeur de z (pas d'itération pour $\bar{c}_s = o$).

(') Les seules itérations inverses possibles sont celles pour lesquelles $\bar{c}_s = o$; elles donneraient un point extrémal de H' qu'on obtiendra de toute façon par une autre arête.

SOL : liste des solutions quasi-optimales, dont les éléments ont la forme suivante :

(num , z , $\{ (i_1, x_1), \ldots , (i_{m+1}, x_{m+1})\}$) ,

où num est le numéro de la solution,

z la valeur de la fonction économique et

$\{(i_1, x_1) , \ldots , (i_{m+1} , x_{m+1})\}$ l'ensemble des indices des variables de base et valeurs correspondantes de ces variables.

3.3.5. Algorithme abstrait.

Etape 0

C Initialisations

Mise au point du tableau-simplexe complété;

C Ce tableau doit avoir toutes ses lignes sauf éventuellement

C celle de la fonction économique lexicographiquement positives.

$k_o = k$;

$K = \phi$;

$j = o$;

SOL = $\{(o, z^o, \{(i_1, x_1), \ldots, (i_{m+1}, x_{m+1})\})\}$

$p = o$;

Cette solution est donnée par le tableau optimal qu'on numérote 0.

Fin de l'étape 0.

Etape 1.

C A partir du tableau-simplexe p correspondant au $(p+1)^{e}$ point

C extrémal, on détermine les arêtes issues de P_p selon lesquel-

C les on peut se déplacer en faisant croître la fonction écono-

C mique et on recherche de combien on doit se déplacer (dépasse-

C ment nul compris) sur chacune d'elle, pour atteindre le sommet

C adjacent.

Créer une branche "ps" pour chaque variable x_s hors base pour laquelle $\bar{c}_s^p > o$;

Pour chacune d'elle ,

déterminer minimum lexico $\{(\bar{b}_i^p / \bar{a}_{is}^p , \bar{a}_{i1}^p / \bar{a}_{is}^p , \ldots , \bar{a}_{in+1}^p / \bar{a}_{is}^p)$

sur $\{ i : \bar{a}_{is}^p > o \}$;

C Ce minimum est toujours réalisé puisqu'il existe toujours un

C $\bar{a}_{is}^p > o$ (ne fût-ce $\bar{c}_s^p$).

C Supposons-le réalisé pour i = r.

C

C Pour chaque point extrémal adjacent à P_p , déterminer la

C valeur de y associée à ce point, c'est à dire k_{ps} .

C

Si $r = n + 1$,

alors

C y sort de la base

$k_{ps} = o$;

Si $r \neq n + 1$,

alors

$k_{ps} = k_p - \bar{c}_s^p \, \bar{b}_{rs}^p / \bar{a}_{rs}^p$;

C $o \leq k_{ps} < k$

$K = K \cup \{k_{ps}\}$;

Etape 2.

C On y détermine si possible la solution de z minimum parmi l'

C ensemble des solutions accessibles.

Si $K = \phi$

alors

C il n'y a plus de sommet à visiter.

aller à l'étape 3.

Si $K \neq \phi$

alors

C il reste des sommets à visiter. On va visiter celui de z

C minimum ou de k_{ls} maximum.

C Cela consiste à déterminer les indices l (correspondant à un
C tableau-simplexe) et s (correspondant à une variable entrante
C relative à ce tableau) définis comme suit :
C soit P_k (k = o, ... ,j) un point extrémal déjà obtenu et P_{ks}
C le point extrémal adjacent à P_k correspondant à l'introduction
C dans la base du tableau k de la variable x_s ;
C il s'agit de déterminer parmi les P_{ks} , le point P_{ls} cor-
C respondant à la plus petite valeur de la fonction économique.

| max K = k_{ls} ;

C Si plusieurs k_{ls} réalisent max K, choix d'un k_{ls} quelcon-
C que, par exemple celui de l, le plus grand.
C On met K à jour.

| K = K \ $\{k_{ls}\}$;

C Ayant déterminé les indices l et s de P_{ls} , on voudrait voir
C si P_{ls} a déjà été visité et s'il ne l'a pas été, le déterminer.
C On pourrait penser à comparer les bases, mais il se pourrait
C qu'on atteigne un point extrémal dégénéré correspondant à des
C bases différentes.
C On doit donc calculer les coordonnées de P_{ls} .

| $x_s = \bar{b}^l_{r_s} / \bar{a}^l_{r_s s}$;
| $x_i = \bar{b}^l_i - x_s \bar{a}^l_{is}$ pour i indice d'une variable de base;
| $x_i = o$ pour i indice d'une variable hors base;

C Regardons si ce point a déjà été visité.
C Il suffit de parcourir, en commençant par la fin, la liste SOL
C et de comparer avec les solutions de $z = z^{ls}$.

| Si P_{ls} non encore visité
| alors
| si P_{ls} appartient à H' ($k_{ls} = o$)
| alors
| ajouter cette solution à SOL;
| retour à l'étape 2;

Il est inutile de poursuivre la recherche à partir de P_{ls} .

| | sinon

Il est nécessaire d'effectuer le pivotage pour pouvoir pour-
suivre la recherche à partir de P_{ls} .

pivoter le tableau l avec $\bar{a}^{l}_{r_s s}$ pour pivot;
numéroter ce tableau : numéro j + 1;
ajouter cette solution à SOL;
$k_{j+1} = k_{ls}$;
j = j + 1;
aller à l'étape 1 avec p = j ;

sinon

C P_{ls} a déjà été obtenu ($P_{ls} = P_q$)

Si les ensembles des indices de base de ces points sont les mêmes,

alors

retour à l'étape 2;

sinon

C On retient le tableau correspondant à la ligne économique
C lexico-minimale.

calcul de $\bar{c}^{ls}$;
min lexico $\{\bar{c}^{ls}, \bar{c}^{q}\} = \bar{c}$;
si $\bar{c} = \bar{c}^{q}$

alors

retour à l'étape 2;

C On peut oublier B^{ls}

Si $\bar{c} = \bar{c}^{ls}$

alors

pivoter le tableau l avec $\bar{a}^{l}_{r_s s}$ pour pivot;
tableau simplexe q = tableau-simplexe l;
$K = K \setminus (\bigcup_i \{k_{qi}\})$;
aller à l'étape 1 avec p = q ;

C
C On a toutes les solutions k optimales.
C

Etape 3.

Imprimer SOL.

C
C Fin.
C

En général, on ne soulève pas le problème du cyclage.
L'algorithme se simplifie.
A l'étape 1, on remplace "min lexico..." par "min $\{ \bar{b}_i^p / \bar{a}_{is}^p \}$".
A l'étape 2, on supprime la fin à partir de "Si P_{ls} déjà obtenu".

3.3.6. Exemple.

Considérons le PLS.

$$\min z = -3x_1 - 4x_2 - 5x_3 - 6x_4$$

$$\text{s.c.} \begin{cases} x_1 + x_2 + x_3 + x_4 + x_5 = 18 \\ 2x_3 + 3x_4 + x_6 = 6 \\ x_i \geq 0 \quad (i = 1, \ldots, 6) \end{cases}$$

L'algorithme du simplexe fournit les tableaux suivants :

	1	x_1	x_2	x_3	x_4	x_5	x_6
-z	0	-3	-4	-5	-6	0	0
x_5	18	1	1	1	1	1	0
x_6	6	0	0	2	3	0	1
-z	12	-3	-4	-1	0	0	2
x_5	16	1	1	1/3	0	1	-1/3
x_4	2	0	0	2/3	1	0	1/3
-z	76	1	0	1/3	0	4	2/3
x_2	16	1	1	1/3	0	1	-1/3
x_4	2	0	0	2/3	1	0	1/3

Le dernier tableau donne la solution optimale unique $(0, 16, 0, 2, 0,0)'$ de $z = -76$.
La précondition de l'algorithme de la méthode du simplexe inverse est vérifiée et on peut résoudre, par cette méthode et pour k = 20 par exemple, le problème des solutions quasi-optimales.

Etape 0

Le tableau optimal complété est le suivant :

	1	x_1	x_2	x_3	x_4	x_5	x_6	$y=x_7$
- z	76	1	o	1/3	o	4	2/3	o
x_2	16	1	1	1/3	o	1	-1/3	o
x_4	2	o	o	2/3	1	o	1/3	o
$y=x_7$	20	1	o	1/3	o	4	2/3	1

j = o ;

k_o = 20 ;

K = ϕ ;

SOL = $\{(o, -76, \{(2,16), (4,2), (7,20)\})\}$;

aller à l'étape 1 avec p = o ;

Etape 1.

"01" : pivot = $\bar{a}^{o}_{21}$; k_{01} = 4 ;

"03" : pivot = $\bar{a}^{o}_{43}$; k_{03} = 19 ;

"05" : pivot = $\bar{a}^{o}_{75}$; k_{05} = 0 ;

" 06": pivot = $\bar{a}^{o}_{46}$; k_{06} = 16 ;

K = $\{k_{01}, k_{03}, k_{05}, k_{06}\}$;

Etape 2.

K $\neq \phi$;

max K = k_{03} = 19 ;

K = $\{k_{01}, k_{05}, k_{06}\}$;

P_{03} : (0,15, 3, 0, 0, 0) ;

P_{03} : non encore visité car $z^{03} \neq z^{o}$;

P_{03} n'appartient pas à H' car $k_{03} \neq 0$;

Pivoter le tableau o avec $\bar{a}^{o}_{43}$ pour pivot :

Tableau 1.

	1	x_1	x_2	x_3	x_4	x_5	x_6	$y=x_7$
- z	75	1	o	o	-1/2	4	1/2	o
x_2	15	1	1	0	-1/2	1	-1/2	o
x_3	3	o	o	1	3/2	o	1/2	o
$y=x_7$	19	1	o	o	-1/2	4	1/2	1

SOL = SOL $\cup \{ (1, -75, \{(2,15), (3,3), (7,19)\}) \}$;

$k_1 = 19$;

j = 1 ;

aller à l'étape 1 avec p = 1 ;

Etape 1.

"11" : pivot = $\bar{a}^1_{21}$; $k_{11} = 4$;

"15" : pivot = $\bar{a}^1_{75}$; $k_{15} = o$;

"16" : pivot = $\bar{a}^1_{36}$; $k_{16} = 16$;

$K = \{ k_{01}, k_{05}, k_{06}, k_{11}, k_{15}, k_{16} \}$;

Etape 2.

$K \neq \phi$;

max $K = k_{06} = k_{16} = 16$;

k_{16} choisi ;

$K = \{ k_{01}, k_{05}, k_{06}, k_{11}, k_{15} \}$;

P_{16} a pour coordonnées (0,18, 0,0,0,6) ;

P_{16} non encore visité car $z^{16} \neq z^1$;

P_{16} n'appartient pas à H' car $k_{16} \neq 0$;

pivotage du tableau 1 avec $\bar{a}^1_{36}$ pour pivot ;

Tableau 2.

	1	x_1	x_2	x_3	x_4	x_5	x_6	$y=x_7$
- z	72	1	o	-1	-2	4	o	o
x_2	18	1	1	1	1	1	o	o
x_6	6	o	o	2	3	o	1	o
$y=x_7$	16	1	o	-1	-2	4	o	1

SOL = SOL $\cup \{ (2, -72, \{ (2,18), (6,6), (7,16) \}) \}$;

$k_2 = 16$;

$j = 2$;

retour à l'étape 1 avec p = 2 ;

Etape 1.

"21" : pivot = $\bar{a}^2_{71}$; $k_{21} = o$;

"25" : pivot = $\bar{a}^2_{75}$; $k_{25} = o$;

$K = \{ k_{01}, k_{05}, k_{06}, k_{11}, k_{15}, k_{21}, k_{25} \}$;

Etape 2.

$K \neq \phi$;

max $K = k_{06} = 16$

$K = \{ k_{01}, k_{05}, k_{11}, k_{15}, k_{21}, k_{25} \}$;

P_{06} a pour coordonnées (0, 18, 0, 0, 6) ;

P_{06} déjà visité ;

$B^{06} = B^2$;

retour à l'étape 2 ;

Etape 2.

$K \neq \phi$;

max $K = k_{01} = k_{11} = 4$

k_{11} choisi ;

$K = \{ k_{01}, k_{05}, k_{15}, k_{21}, k_{25} \}$;

P_{11} a pour coordonnées (15, 0, 3, 0, 0, 0) ;

P_{11} non encore visité ;

P_{11} n'appartient pas à H' car $k_{11} \neq 0$;

Pivoter le tableau 1 avec $\bar{a}^1_{21}$ pour pivot ;

Tableau 3.

	1	x_1	x_2	x_3	x_4	x_5	x_6	$y=x_7$
- z	60	0	-1	0	0	3	1	0
x_1	15	1	1	0	-1/2	1	-1/2	0
x_3	3	0	0	1	3/2	0	1/2	0
x_7	4	0	-1	0	0	3	1	0

SOL = SOL U $\{(3, -60, \{(1,15),(3,3),(7,4)\})\}$;

$k_3 = 4$;

j = 3 ;

aller à l'étape 1 avec p = 3 ;

Etape 1.

"35" : pivot = $\bar{a}^3_{75}$; $k_{35} = 0$;

"36" : pivot = $\bar{a}^3_{76}$; $k_{36} = 0$;

$K = \{ k_{01}, k_{05}, k_{15}, k_{21}, k_{25}, k_{35}, k_{36} \}$;

Etape 2.

$K \neq \phi$;

max $K = k_{01} = 4$;

$K = \{ k_{05}, k_{15}, k_{21}, k_{25}, k_{35}, k_{36} \}$;

P_{01} non encore visité ;

P_{01} n'appartient pas à H' car $k_{01} \neq 0$;

pivotage du tableau 0 avec $\bar{a}^0_{21}$, pour pivot ;

Tableau 4.

	1	x_1	x_2	x_3	x_4	x_5	x_6	$y=x_7$
- z	60	0	-1	0	0	3	1	0
x_1	16	1	1	1/3	0	1	-1/3	0
x_4	2	0	0	2/3	1	0	1/3	0
$y=x_7$	4	0	-1	0	0	3	1	1

SOL = SOL $\cup$ {(4, -60, {(1,16),(4,2),(?,4)})} ;

$k_4 = 4$;

j = 4 ;

aller à l'étape 1 avec p = 4 ;

Etape 1.

"45" : pivot = $\bar{a}^4_{75}$; $k_{45} = 0$;

"46" : pivot = $\bar{a}^4_{76}$; $k_{46} = 0$;

$K = \{ k_{05}, k_{15}, k_{21}, k_{35}, k_{36}, k_{45}, k_{46} \}$;

Etape 2.

$K \neq \phi$;

max K = $k_{**} = 0$;

k_{46} choisi ;

$K = \{ k_{05}, k_{15}, k_{21}, k_{35}, k_{36}, k_{45} \}$;

p_{46} a pour coordonnées (52/3 , 0 , 0 , 2/3 , 0 , 4) ;

P_{46} non encore visité ;

P_{46} appartient à H' car $k_{46} = 0$;

SOL = SOL $\cup$ { (5 , -56 , { (1, 52/3) , (4, 2/3) , (6, 4) }) } ;

retour à l'étape 2 ;

etc.

SOL final :

num	z	i_1	b_1	i_2	b_2	i_3	b_3
0	-76	2	16	4	2	7	20
1	-75	2	15	3	1	7	19
2	-72	2	18	6	6	7	16
3	-60	1	15	3	3	7	4
4	-60	1	16	4	2	7	4
5	-56	1	52/3	4	2/3	6	4
6	-56	2	41/4	3	3	5	19/4
7	-56	1	16	2	2	6	6
8	-56	2	14	5	4	6	6
9	-56	1	44/3	4	2	5	4/3
10	-56	2	11	4	2	5	5
11	-56	1	44/3	3	3	5	4/3
12	-56	1	4/3	3	1	6	4

3.3.7. Critique de la méthode.

Avantages.

La rec rche des solutions quasi-optimales se fait au fur et à mesure que z croît. Il est dès lors facile d'arrêter la recherche à n'importe quel moment.

Désavantages.

La méthode ne répond pas exactement au problème posé :

- elle n'est applicable que s'il existe une seule solution optimale et par conséquent la recherche de solutions optimales multiples ne peut être un cas particulier de cette méthode;
- si on l'applique à des problèmes ayant plusieurs solutions optimales, on n'obtient qu'un sous-ensemble de solutions quasi-optimales, celles qui sont accessibles par itérations -simplexe inverses avec $\bar{c}_s > o$.

La méthode exige le stockage en mémoire de nombreuses bases. Il est impensable de les garder toutes en mémoire centrale. De là, le temps consacré aux entrées - sorties risque d'être important et la taille des problèmes résolubles ne peut être très grande.

CHAPITRE IV : METHODES DE LABYRINTHE

4.1. Quasi-optimalité et labyrinthes.

On peut remarquer que le problème du traitement de solutions optimales multiples se ramène à un cas particulier du problème classique des labyrinthes en théorie des graphes. Il suffit de remarquer que tout ensemble polyédrique convexe peut être représenté par un graphe convexe (V.U), V étant l'ensemble de ses sommets et U l'ensemble des arcs de voisinage des sommets qui s'explicitent par des pivotages-simplexe. Ainsi, deux sommets sont voisins s'ils correspondent à des bases-simplexe différant d'une seule variable; chaque arc peut donc être traversé dans les deux sens du fait que chaque pivotage-simplexe est réversible.

La recherche de tous les sommets de l'hyperpolyèdre définissant le PLS quasi optimum se ramène à l'exploration de tous les sommets du graphe (V, U) dont, à l'exception des arcs de voisinage, la forme explicite n'est pas connue à priori. Il s'agit là par conséquent de la recherche d'un parcours dans un labyrinthe où les carrefours sont représentés par les sommets et les allées par les arcs de (V, U).

Pour formaliser un peu ces idées, nous devons donner quelques définitions :

- V : ensemble des sommets du graphe composé des m-uples d'entiers, indices de base-simplexe, $v = \{ i_1, \dots, i_m \}$;
- Deux sommets différents $v_1 = \{ i_1, \dots, i_m \}$ et $v_2 = \{ k_1, \dots, k_m \}$ sont à distance $d \leq m$ l'un de l'autre si ces sommets diffèrent par d composantes exactement; ces sommets sont voisins si $d = 1$;

- un arc (v_1, v_2) appartient à U si et seulement si v_1 et v_2 sont voisins (les sommets voisins de v_i forment un ensemble noté $\Gamma(v_i)$).

Le problème de la recherche de tous les sommets d'un graphe (V, U) se formule maintenant différemment : trouver un chemin qui passe par tous les sommets du graphe en ne gardant en mémoire qu'un seul tableau-simplexe pour générer les nouveaux sommets.

1.2. Un aperçu des méthodes de labyrinthe [8]

Aujourd'hui, le problème de la recherche explicite des sommets d'un ensemble polyédrique convexe présente une bibliographie très abondante. De plus, ce domaine de recherche n'est pas démuni d'applications; en voici quelques-unes :

- visualiser, à l'aide de micro-ordinateurs, des hyper-polyèdres à cinq dimensions;
- détecter les contraintes redondantes d'un système d'inégalités linéaires;
- ...

Les méthodes d'énumération des sommets de polyèdre peuvent être divisées en deux classes :

- les méthodes dites de pivotage, par exemple, celles de Balinski [6], Manas et Nedoma [7] (*), Mattheiss [8], Omar [10],... ;
- les méthodes géométriques ou sans pivotage-simplexe, par exemple, celle de Chernikova [9].

Des études comparatives de ces méthodes ont fourni les résultats suivants (**) :

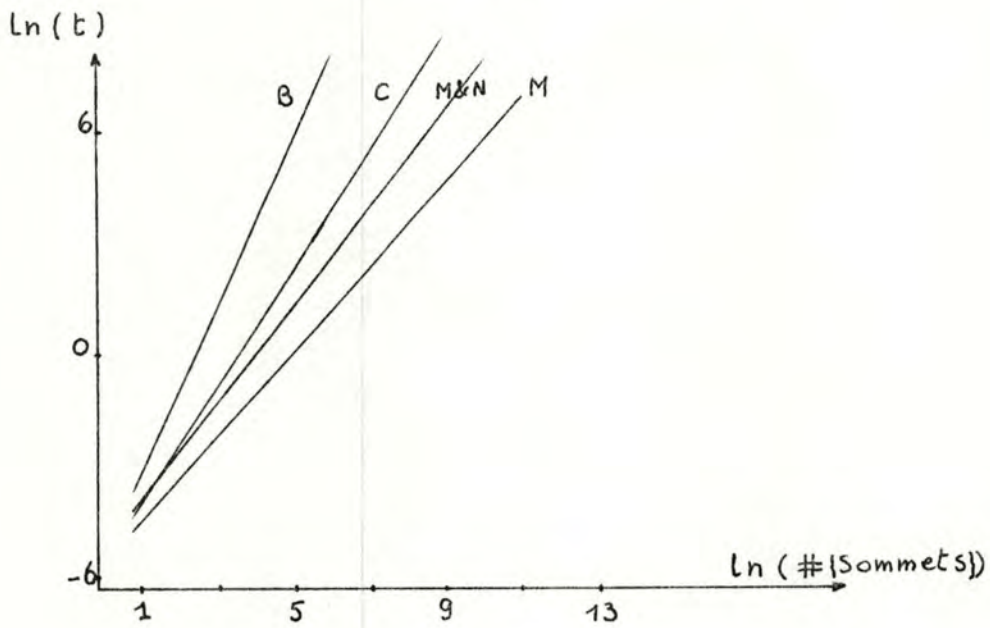
(*) Nous la développerons dans la suite.

(**) Pour plus de détails, voir [8] p.175 à 178.

- taille maximum (m x n) des problèmes résolus par la méthode de :
  . Balinski : 20 x 8
  . Chernikova : 25 x 8
  . Manas et Nedoma : 23 x 14
  . Mattheiss : 29 x 20

pour une mémoire virtuelle de 512 k.

- temps pour résoudre un problème :
  ln (temps en secondes) = $b_0 + b_1$ ln (nombre de sommets)
  . Balinski : $b_0 = -5{,}345$
  $b_1 = 2{,}272$
  . Chernikova : $b_0 = -5{,}589$
  $b_1 = 1{,}418$
  . Manas et Nedoma : $b_0 = -5{,}046$
  $b_1 = 1{,}379.$
  . Mattheiss : $b_0 = -5{,}386$
  $b_1 = 1{,}146.$



Dans la suite, nous présenterons une des méthodes de pivotage, celle de Manas et Nedoma, dont l'efficacité a été reconnue (voir ci-dessus).

4.3. Méthode de Manas et Nedoma.

4.3.1. Idée de la méthode de Manas et Nedoma.

En (4.1), nous avons montré que la recherche des solutions quasi-optimales peut se formuler comme un problème classique de labyrinthe en théorie des graphes.

Appelons G le graphe (V, U) défini en (4.1). Il est symétrique et simplement connexe.

L'algorithme de Manas et Nedoma permet de déterminer un squelette de G en n'enregistrant en mémoire qu'un seul tableau-simplexe.

Naturellement, on pense à des algorithmes d'exploration en profondeur ou largeur d'abord, lesquels nécessitent cependant une gestion d'ensembles assez importante ou l'emploi de la récursivité, choses désagréables à programmer en Fortran.

L'algorithme de Manas et Nedoma, assez semblable à un algorithme d'exploration en profondeur se veut d'éviter la récursivité et d'utiliser des structures de données relativement simples. Pour atteindre cet objectif, il considère comme sommets accessibles à partir du sommet courant, non seulement ses voisins mais aussi les voisins de tout sommet déjà visité (') . Pour diminuer le nombre de retours et donc de pivotages, il conviendra d'ordonner l'ensemble des sommets candidats en sous-ensembles de sommets à même distance du sommet courant et de choisir comme sommet à visiter un sommet à distance minimale.

L'algorithme part du tableau-simplexe optimal et construit itérativement deux ensembles R et W contenant respectivement d'une part l'ensemble des sommets déjà rencontrés et,

(') Différence par rapport aux algorithmes à retour classiques.

d'autre part, l'ensemble des sommets non rencontrés accessibles à partir d'au moins un sommet de R par un seul pivotage-simplexe. Il se termine lorsque W est vide à savoir lorsqu'il ne reste plus de sommets à atteindre à partir de R.

Le nombre d'itérations est compris entre $\#V$ et $\#V \times m$, le type de polyèdre où le nombre d'itérations serait $\#V$ exactement (chemin hamiltonien dans le graphe) n'est pas encore prescrit par les théoriciens.

4.3.2. L'algorithme abstrait de Manas et Nedoma.

Par algorithme abstrait, nous entendons concevoir un algorithme pouvant servir de base à toute conception concrète.

Les structures de données sont décrites sommairement ainsi que les actions considérées comme élémentaires.

Elles restent inchangées quel que soit le langage utilisé dans la suite.

4.3.2.1. Spécification du problème.

Arguments : un PLS
le tableau optimal de ce PLS ($\bar{c}_N \geq \underline{0}$)
un réel $k > 0$.

Résultats : liste de tous les sommets du polyèdre associé au problème de recherche des solutions k - optimales (éventuellement l'ensemble des points extrémaux) (').

Cette méthode, contrairement à la méthode simplexe inverse permet de déterminer toutes les solutions optimales finies.

(') - Deux sommets différents peuvent correspondre à un même point extrémal.

- Le cyclage est automatiquement géré.

4.3.2.2. Définition des variables.

j : entier indiçant les sommets déjà visités.
v_j : j^e sommet visité.
$\Gamma(v)$: ensemble des sommets voisins d'un sommet v.
R : ensemble des sommets déjà visités.
W : ensemble des sommets non encore visités, accessibles à partir d'au moins un sommet de R par un seul pivotage simplexe.

4.3.2.3. Algorithme abstrait.

C Initialisations.
Mise à jour du tableau simplexe complété.
$j = 1$
v_1 = sommet optimal.
$R = \{ v_1 \}$
déterminer $\Gamma(v_1)$
$W = \Gamma(v_1)$

C Corps.
tant que $W \neq \phi$, répéter
déterminer v_{j+1} appartenant à W et à distance minimale de v_j.
$j = j + 1$
$R = R \cup \{ v_j \}$
Déterminer $\Gamma(v_j)$
$W = W \cup \Gamma(v_j) - R$

C Fin de la boucle.
C Facultatif.
Trier R d'après la valeur de la fonction économique.
Eliminer les sommets correspondant à un même point extrémal.
C Fin de l'algorithme.

4.3.2.4. Démonstration.

Soit G = (V, U) le graphe représentant le polyèdre.

Vu la connexité, il est toujours possible de trouver un chemin entre deux sommets quelconques. L'action "déterminer v_{j+1}..." est par conséquent toujours réalisable.

A chaque passage dans la boucle, R et W représentent bien respectivement l'ensemble des sommets déjà visités et l'ensemble des sommets non rencontrés accessibles à partir d'au moins un sommet de R par un seul pivotage simplexe.

Il est évident qu'après un nombre fini d'itérations, W sera vide et donc on sortira de la boucle.

Il reste à démontrer que W = 0 entraîne R = V. Vu la connexité, lorsqu'il n'y a plus de sommets accessibles à partir de R, on a visité tous les sommets et R = V.

4.3.2.5. Remarque.

L'algorithme concret est présenté dans l'annexe mais avant il est nécessaire de parler du logiciel de programmation linéaire, X M P , utilisé par l'algorithme.

CHAPITRE V : LE LOGICIEL DE PROGRAMMATION LINEAIRE X M P (').

5.1. Remarques préliminaires.

Les deux méthodes présentées nécessitent bien entendu l'utilisation de l'algorithme du simplexe pour la résolution du PLS et par la suite, pour la recherche des points extrémaux quasi-optimaux, l'application d'itérations simplexes particulières (itérations simplexe inverse).

le travail proposé demandait, dans la mesure du possible, d'utiliser le logiciel de programmation linéaire LAMPS, présent sur le DEC - 20 de l'institut, comme outil de base.

Ce logiciel, de par son aspect modulaire, offre de nombreuses possibilités quant à la résolution de PL et des efforts ont été entrepris pour rendre les interfaces agréables.

Cependant, l'emploi d'un logiciel comme programme utilitaire présente souvent des inconvénients :

- toute modularité, aussi bonne et bien structurée soit-elle, ne rend pas toujours aisées certaines opérations particulières d'un niveau plus bas que le dernier niveau de découpe présent.
  Par exemple, le module principal d'optimisation (PRIMAL) effectue un grand nombre d'itérations simplexe en choisissant le pivot adéquat mais il n'est pas possible de demander de réaliser une seule itération simplexe avec un élément donné pour pivot.

- une structure et une gestion de données efficaces pour des problèmes courants peuvent nécessiter des lourdes opérations lorsqu'on emploie le logiciel dans des situations inhabituelles.

(') Documentation détaillée disponible au centre de calcul de l'institut.

Il ne s'agit nullement de reproches à l'égard de LAMPS qui est en fait un logiciel commercial très bon pour des problèmes courants mais inadaptés à la recherche algorithmique.

Après plusieurs tentatives, nous concluons que LAMPS ne peut être utilisé efficacement pour notre genre de problèmes.

Pensez seulement, par exemple, qu'il est impossible de réaliser les deux opérations suivantes souvent présentes dans les algorithmes : forcer un pivotage particulier, lire B^{-1}.

On s'est dès lors tourner vers X M P, un logiciel de programmation linéaire plus orienté recherche algorithmique.

5.2. Caractéristiques générales du logiciel XMP.

Le logiciel de programmation linéaire X M P se présente sous la forme d'une librairie et a pour but de faciliter la recherche algorithmique.

Il a été conçu comme outil expérimental destiné à un large spectre de chercheurs et notamment à ceux désireux d'utiliser la méthode simplexe ou une partie de celle-ci comme sous-programme dans un algorithme plus général.

De là, il n'a nullement l'intention de rivaliser avec les logiciels commerciaux (tels MPSX / MIP d'IBM, LAMPS,...) et il n'est pas aussi performant que ceux-ci point de vue taille maximum des problèmes à résoudre, temps d'exécution et possibilités d'emploi offertes directement.

Voici présentées succintement par ordre d'importance décroissant les principales caractéristiques.

- Manière de l'employer.

La librairie consiste entièrement à des sous-programmes FORTRAN qui peuvent être appelées par des programmes utilisateurs.

En particulier, il est possible d'accéder à tous les résultats intermédiaires.

- Lisibilité.

Le logiciel est écrit en FORTRAN et non en langage assembleur et est bien documenté si bien qu'il soit possible de le comprendre et de là, le modifier ou d'écrire des alternatives.

- Modularité et structure hiérarchique.

Leurs avantages sont bien connus et les programmes utilisateurs peuvent appeler des routines de tout niveau. Il n'y a dès lors pas de variables globales; les arguments sont explicites et de là très nombreux.

- Modules abstraits de données.

Les modules de données (données du PLS et inverse de la base) sont visibles par les opérateurs définis sur eux.

- Portabilité.

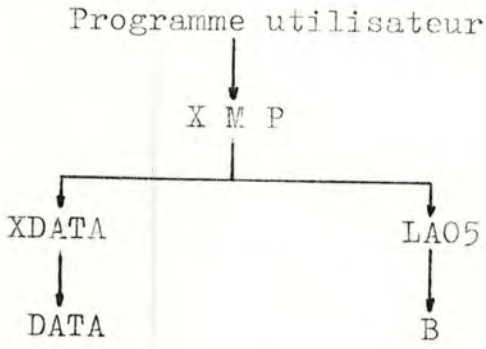
Il est écrit en FORTRAN classique et tourne sur IBM, CDC, DEC.

- Taille des problèmes.

On peut résoudre des problèmes comportant jusqu'à 1.500 contraintes.

- L'extensibilité et la modifiabilité sont aussi des caractéristiques avantageuses de ce logiciel et découlent de ce qui précède.

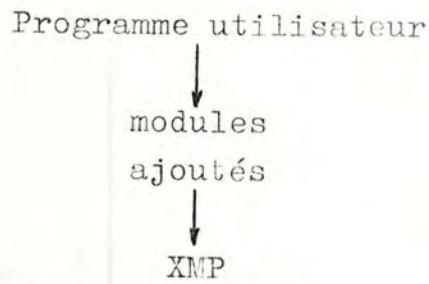
La librairie XMP consiste en 38 sous-programmes auxquels s'ajoutent deux modules données permettant la gestion des données du problème (XDATA*) et de l'inverse de la base ou plus précisément des facteurs triangulaires LU de la base (LA05*).



Il y a trois façons d'utiliser X M P.

- Un programme principal, écrit par nous, lit des données, génère la matrice du PL, appelle XMP et prend en charge les outputs de XMP.
- Le programme XDEMO, écrit pour nous, se charge de tout.
- Le programme MPDEMO fait de même, la seule différence étant le format des inputs (forme MPS-standard pour MPDEMO et forme propre à lui pour XDEMO).

Dans ce travail, nous nous tiendrons à la philosophie de XMP et notre tâche consiste en fait à rajouter un ou plusieurs modules de niveau supérieur :



et non à faire un logiciel interactif de programmation linéaire quasi-optimale.

5.3. Principaux sous-programmes.

Il y a six catégories de sous-programmes dans la librairie XMP :

1. Sous-programmes implémentant la logique de la méthode du simplexe;
2. sous-programmes servant d'interface entre la méthode du simplexe et les données du problème;
3. sous-programmes servant d'interface entre la méthode du simplexe et la représentation de l'inverse de la base;
4. sous-programmes gérant directement les données du problème;
5. sous-programmes gérant directement la représentation de l'inverse de la base;
6. sous-programmes fournissant des services variés.

Voici une description rapide et par catégorie des sous-programmes que nous utilisons.

1. XBCOMP : calcule la valeur courante des variables de base et de la fonction objectif.

   XCHUZR : détermine la variable quittant la base lors d'un pivotage simplexe primal.

   XDOT : calcule le produit d'un vecteur ligne par une colonne compactée de la matrice.

   XPIVOT : réalise un pivotage simplexe primal.

   XPRIML : sous-programme du premier niveau réalisant la méthode du simplexe.

2. XADDAJ : ajoute une colonne au PL.

   XADDUB : ajoute des limites pour une variable.

   XGETAJ : lit une colonne du PL.

   XGETUB : lit les limites d'une variable.

3. XBTRAN : calcule le produit d'un vecteur ligne par B^{-1}.

   XFACT : réfactorise la base courante.

6. XSTOP : fournit un traitement centralisé de toutes les erreurs fatales et arrête le programme.

5.4. Variables X M P.

Nous décrirons rapidement les variables XMP que nous utilisons; pour plus de détails, voir dictionnaire des variables dans la documentation XMP.

B : RHS.

BASCB : tableau contenant $\bar{c}_B$.

BASIS : liste des variables de base; BASIS(i)= j signifie que la i^e variable de base est j.

BASLB : tableau des bornes inférieures des variables de base.

BASUB : tableau des bornes supérieures des variables de base.

BIG : valeur utilisée pour représenter + infini.

BNDTYP : 1 : toute variable non libre a des bornes égales à 0 et + infini;
2 : toute variable structurelle non libre a des bornes égales à 0 et BOUND;
3 : la limite inférieure de toute variable non libre est 0;
4 : les bornes sont générales.

BOUND : limite supérieure commune dans le cas BNDTYP = 2.

CAND : liste des variables hors base susceptibles d'y entrer.

CANDA : table contenant les coefficients non nuls des colonnes qui appartiennent à CAND.

CANDCJ : liste contenant les coefficients de la fonction objectif pour chaque colonne de CAND.

CAND : table contenant les numéros de ligne correspondant à un coefficient non nul d'une colonne de CAND.

CANDL : liste contenant le nombre de coefficients non nuls dans chaque colonne de CAND.

CJ : coefficient de la fonction objectif.

COLA : liste des coefficients non nuls d'une colonne de la matrice.

COLI : liste des numéros de ligne correspondant aux coefficients non nuls d'une colonne de la matrice.

COLLEN : nombre de coefficients non nuls dans une colonne de la matrice.

COLMAX : nombre maximum de coefficients non nuls dans toute colonne de la matrice.

DQ : profit relatif d'une variable entrante.

FACTIT : nombre d'itérations effectuées depuis la dernière factorisation.

FACTOR : fréquence de la refactorisation.

INTYP : + 1 : la variable entrante croît à partir de sa borne inférieure;
- 1 : la variable entrante décroît depuis sa borne supérieure.

IOERR : unité d'I-O.

IOLOG : unité d'I-O.

J : variable.

LEAVE : variable quittant la base.

LENMA : longueur du tableau MAPA.

LENMI : longueur du tableau MAPI.

LENMY : longueur du tableau MEMORY.

LJ : bonne inférieure d'une variable.

LQ : bonne inférieure de la variable entrante.

M : nombre de contraintes.

MAPA : pointeurs vers le tableau MEMORY (partie données du problème).

MAPI : pointeurs vers le tableau MEMORY (partie inverse de la base).

MAXM : nombre maximum de contraintes permises.

MAXN : nombre maximum de variables permises.

MEMORY : tableau principal contenant tous les tableaux représentant les données du problème et l'inverse de la base.

N : nombre de variables.

OUTTYP : + 1 : la variable sortante va vers sa limite inférieure.
- 1 : la variable sortante va vers sa limite supérieure.
0 : les variables entrante et sortante sont les mêmes.

PIVOT : pivot.

Q : indice de la variable entrante.

R : position de la variable sortante, l'indice de la variable sortante est BASIS (R).

ROWTYP : tableau des types de lignes :
+ 2 : contrainte du type "A <= ... <= B";
+ 1 : contrainte du type <= ;
0 : contrainte du type = ;
- 1 : contrainte du type >= ;
- 2 : ligne libre (fonctionnelle).

STATUS : tableau indicateur de chaque variable :

- 0 : variable hors de la base à sa limite inférieure;
- k : k^e variable de base ;
- - 1 : variable hors de la base à sa limite supérieure;
- - 2 : variable libre (entrée dans la base, elle y reste) ;
- - 3 : variable artificielle (sortie de la base, elle n'y rentre plus ;
- - 4 : variable bloquée hors de la base à sa limite inférieure ;
- - 5 : variable bloquée hors de la base à sa limite supérieure ;

remarque : variables libres et artificielles ont toujours STATUS = -2 ou -3 même si elles sont dans une base.

UJ : borne supérieure d'une variable.

UQ : borne supérieure de la variable entrante.

UZERO : tableau des $\bar{c}_B$.

XBZERO : tableau des $\bar{b}$.

YQ : colonne non compactée.

Z : ...eur de la fonction objectif.

ZLC : on considère comme nuls les nombres plus petits en valeur absolue que ZLC.

5.5. Paramètres des sous-programmes utilisés.

XADDAJ : CJ, COLA, COLI, COLLEN, COLMAX, IOERR, J, LENMA, LENMY, MAPA, MEMORY, N.

XADDUB : BNDTYP, IOERR, J, LENMA, LENMY, LJ, MAPA, MEMORY,UJ.

XBCOMP : B, BASCB, BNDTYP, BOUND, COLA, COLI, COLMAX, IOERR, LENMA, LENMI, LENMY, M, MAPA, MAPI, MAXM, MAXN, MEMORY, N, STATUS, XBZERO, Z.

XBTRAN : IOERR, LENMI, LENMY, M, MAPI, MEMORY, ROW.

XCHUZR : BASIS, BASLB, BASUB, INTYP, IOERR, LQ, M, MAXM, MAXN, N, OUTTYP, PIVOT, Q, R, STATUS, THETA, UNBDD, UQ, XBZERO, YQ.

XDOT : COLA, COLI, COLLEN, COLMAX, MAXM, ROW, ZDOT.

XFACT : BASCB, BASIS, BASLB, BASUB, COLA, COLI, COLMAX, FCODE, IOERR, LENMA, LENMI, LENMY, M, MAPA, MAPI, MAXM, MEMORY.

XGETAJ : CJ, COLA, COLI, COLLEN, COLMAX, IOERR, J, LENMA, LENMY, MAPA, MEMORY.

XGETUB : BNDTYP, IOERR, J, LENMA, LENMY, LJ, MAPA, MEMORY,UJ.

XPIVOT : BASIS, BASLB, BASUB, DQ, INTYP, IOERR, LEAVE, LENMI, LENMY, LQ, M, MAPI, MAXM, MAXN, MEMORY, N, OUTTYP, PCODE, PIVOT, Q, R, STATUS, THETA, UNBDD, UQ, XBZERO, YQ, Z.

XPRIML : B, BASCB, BASIS, BASLB, BASUB, BNDTYP, BOUND, CAND, CANDA, CANDCJ, CANDI, CANDL, COLI, COLA, COLMAX, FACTOR, IOERR, IOLOG, ITER1, ITER2, LENMA, LENMI, LENMY, LOOK, M, MAPA, MAPI, MAXM, MAXN, MEMORY, N, NTYPE2, PICK, PRINT, STATUS, TERMIN, UNBDDQ, UZERO, XBZERO, YQ, Z.

CHAPITRE VI : DEUX HEURISTIQUES ET CONCLUSION.

Le modeste tour d'horizon que nous avons entrepris sur les méthodes exactes de traitement des solutions quasi-optimales en PL n'a fait que nous décourager de les appliquer à des problèmes de taille courante.

Le recours à des techniques heuristiques est une bonne solution pour sortir de l'impasse, d'autant plus que l'on s'intéresse le plus souvent non pas à connaître des milliers de solutions-sommets mais à un tout autre type d'informations comme, par exemple : la stabilité d'une solution dégénérée, le choix d'un ensemble de telles solutions qui soient aussi représentatives que possible de l'hyperpolyèdre, la corrélation éventuelle entre variables, etc...

Siskos [5] propose un type particulier d'analyse où le système des solutions recherchées comprend seulement celles qui maximisent et/ou minimisent chaque fois une ou plusieurs variables.

Winkels [11] propose une heuristique consistant à explorer l'hyperpolyèdre de façon plus régulière sans se limiter forcément à ses sommets ni se préoccuper des propriétés à priori des solutions recherchées. C'est une méthode de discrétisation du polyèdre dont le principe repose sur le choix d'un pas de discrétisation pour chacune des variables.

En conclusion, soulignons encore une fois l'importance du problème de solutions multiples en programmation linéaire, problème que beaucoup d'auteurs sous-estiment et dont les codes informatiques, malheureusement, ne tiennent pas compte.

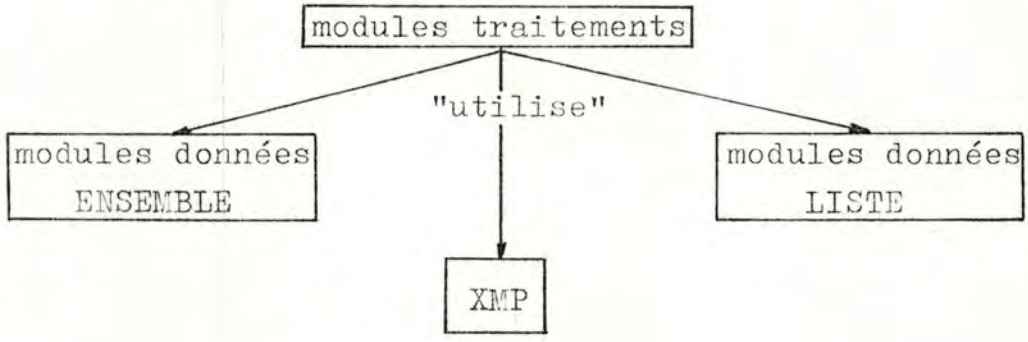
ANNEXE : CONCEPTION CONCRETE DE L'ALGORITHME DE MANAS ET NEDOMA.

Tout en ne parlant pas encore Fortran ou XMP, on tient compte des possibilités offertes par Fortran et des spécifications des modules XMP.

A partir de l'algorithme abstrait, des structures de données abstraites, et des possibilités Fortran-XMP, on précise les actions élémentaires, les modules traitements et les modules données.

Les modules se divisent en trois classes :
- les modules traitements correspondant à des aggrégations-désaggrégations d'actions abstraites élémentaires;
- les modules données correspondant à la représentation de certains ensembles et des opérateurs associés ;
- les modules données correspondant à la représentation de certaines listes et des opérateurs éssociés .

Ils s[illegible] hiérarchisés de la façon suivante :



Définition des variables.

R : ensemble d'éléments v représentant les sommets déjà visités.
Un élément v de R est de la forme :
.**z** : réel, valeur de la fonction économique
.$\{(i_1, b_1), \ldots, (i_m, b_m)\}$: ensemble des indices de base et valeurs correspondantes des variables.
Cet ensemble est visible par ses interfaces :
INIT-R : initialisation de R au vide;
ADD-R : ajout à R d'un élément n'appartenant pas à R;
v-IN-R : test d'appartenance de v à R.

R' : ensemble d'éléments v représentant le sommet courant et le prédécesseur de chaque sommet de R', ce jusqu'au sommet optimal, ainsi que le pivotage décrivant l'arc entre le prédécesseur d'un sommet et ce sommet.
Un élément v de R' a la même forme qu'un élément de R et on ajoute un couple d'entiers (R,S) représentant le pivotage ($x_s \uparrow x_r \downarrow$).
Cet ensemble se structure en une pile et est visible par ses interfaces :
INIT-R' : initialisation de R' au vide;
ADD-R' : ajout au sommet de la pile d'un élément n'appartenant pas à R';
SUBTR-R' : suppression de l'élément sommet de la pile;
GET-R' : accès à l'élément sommet de R'.
CARD-R' : entier, cardinal de R'.

W : l'ensemble des sommets non encore visités accessibles à partir d'un sommet de R par un seul pivotage simplexe est aussi, vu le mode d'exploration des sommets en profondeur d'abord, l'ensemble des sommets non encore visités dont un prédécesseur est dans R'(')

(')Si un sommet non encore visité a deux (ou plus) prédécesseurs dans R', le prédécesseur auquel il sera rattaché est celui le plus récemment visité; cela pour réaliser le moins possible de pivotages pour l'atteindre à partir du sommet courant.

W est donc structuré en une liste de sous-ensembles W^1, ..., $W^{card\text{-}R'}$ de sommets non encore visités de même prédécesseur dans R'; certains W^k pouvant être vides (description d'un sous-ensemble : voir ci-dessus).

W est visible par ses interfaces :

INIT-W : initialisation de la liste W au vide;

W-VIDE : test de W vide;

ADD-W : ajout en fin de liste W d'un élément n'appartenant pas à W;

SUBTR-W : suppression du dernier élément de la liste W.

MAJ-W : mise à jour du k^e élément de W ;

GET-W : accès au k^e élément de W.

W^k : ensemble d'éléments w représentant des sommets non encore visités de même prédécesseur dans R'.

Cet ensemble est identifié par $\{i_1,\ldots,i_m\}$ l'ensemble des indices de base du prédécesseur de R'(').

Un élément w de W^k est de la forme

(r,) : couple d'entrées représentant le pivotage (x_s entre, x_r sort) entre le prédécesseur et w.

W^k est visible par ses interfaces :

INIT-W' : initialisation de l'ensemble W^k au vide;

W'-VIDE : test de W^k vide;

w-IN-W' : test d'appartenance de w à W^k;

ADD-W' : ajout à W^k d'un élément n'appartenant pas à W^k;

SUBTR-W': suppression d'un élément de W^k ;

GET-W' : accès à un élément quelconque de W^k.

(') Redondance contrôlée.

Modules données - type ENSEMBLE.

Nous devons tenir compte des structures de données fournies par Fortran, de la structure des ensembles manipulés et des opérations à réaliser. Il est inutile d'envisager des ensembles de grande taille, aux structures complexes et sur lesquelles on peut réaliser toutes les opérations ensemblistes alors que certains des ensembles ont une taille constante et petite et que les opérations sont simples.

Nous localisons dans ces modules les dépendances dues à la représentation des ensembles, ce qui garantit la modificabilité.

Remarques :
- nous prenons MAXM, le nombre maximum de contraintes, ≤ 32 . Il est inutile de prendre plus, des tests (') ayant montré que les problèmes les plus grands qu'on peut efficacement résoudre ont une taille donnée par N = 15 et M = 25.
- ce n'est pas un hasard si nous employons dans la suite les mêmes identificateurs de variables que XMP; heureusement avec la même signification!

(') Voir [8] p. 175

ENSEMBLE $\{ i_1, \ldots, i_m \}$.

I - BASE.

(représentation de $\{ i_1, \ldots, i_m \}$)

Spécifications.

arg : M :entier
MAXM : entier
BASIS : tableau de MAXM entiers

pré : $0 < M \leq MAXM \leq 32$
$0 < BASIS(k) \leq 32 \quad (k = 1, \ldots, M)$

rés : ENS-I : entier de 32 bits

post : (j^e bit, à partir de la gauche, de ENS-I = vrai)
ssi ($\exists$ i $\in$ [1 : M] : BASIS (i) = j)

Variable locale.

i : entier parcourant BASIS.

Algorithme.

ENS-I = 0
Pour i de 1 à M faire ENS-I = ENS-I + $2^{BASIS(i)-1}$

MAJ - ENS-I.

($\{ i_1, \ldots, i_m \} = \{ i_1, \ldots, i_m \} \cup \{q\} \setminus \{r\}$)

Spécifications.

arg : ENS-I : entier de 32 bits
R : entier
Q : entier

pré : ENS-I représente E = $\{ i_1, \ldots, i_m \}$
R appartient à E
Q appartient à [1 : 32] et non à E

rés : ENS : entier de 32 bits

post : ENS représente E U $\{ Q \}$ - $\{ R \}$

Algorithme.

ENS = ENS - I $-2^{R-1} + 2^{Q-1}$

ENSEMBLE W'.

INIT - W'.

(initialisation d'un ensemble W^k au vide).

Spécif ations.

arg : M : entier
MAXM : entier
ENS-I : entier de 32 bits

pré : $0 < M \leq MAXM \leq 32$
ENS-I représente $\{ i_1, \ldots, i_m \}$

rés : ENS-W' : tableau de MAXM + 2 entiers

post : ENS-W' représente un ensemble W^k vide dont $\{i_1, \ldots i_m\}$ est le prédécesseur.

Remarque.

ENS-W' (MAXM + 1) : ensemble des variables de base du prédécesseur.

ENS-W' (MAXM + 2) : $\# W^k$

ENS-W' (S) (S = 1,...,M) = R : représente $(x_s \uparrow x_R \downarrow)$
= 0 : non significatif, c'est-à-dire pas de pivotage$(x_s \uparrow * \downarrow)$

ENS-W' (S) (S = M + 1,...,MAXM): non significatif.

Variable locale.

k : entier parcourant ENS-W'.

Algorithme.

Pour k de 1 à M faire ENS-W' (k) = 0
ENS-W' (MAXM + 1) = ENS-I
ENS-W' (MAXM + 2) = 0

W' - VIDE.

(fonction booléenne testant si un W^k est vide).

Spécifications.

arg : MAXM : entier
ENS-W' : tableau de MAXM + 2 entiers.

pré : $0 < MAXM \leq 32$
ENS-W' représente un ensemble W^k.

rés : W'-VIDE : booléen

post : W'-VIDE = vrai si W^k est vide, faux sinon.

Algorithme.

W'-VIDE = (ENS-W' (MAXM +2) = 0)

W - IN - W'.

(test d'appartenance de w à un W^k).

Spécifications.

arg : MAXM : entier
ENS-I : entier de 32 bits
ENS-W' : tableau de MAXM + 2 entiers.

pré : $0 < MAXM \leq 32$
ENS-I : représente $\{i_1,\ldots,i_m\}$
ENS-W' représente W^k.

rés : W-IN : booléen
R : entier
S : entier.

post : (W-IN = vrai) et (ENS-I correspond à un élément de ENS-W') et ((R,S) représente le pivotage ($x_S \uparrow x_R \downarrow$) pour passer du prédécesseur de cet élément à cet élément)
ou
(W-IN = faux) et (ENS-I ne correspond pas à un élément de ENS-W') et ((R,S) non significatif).

Variables.

k : entier
E : entier de 32 bits
E-I : entier de 32 bits
i : entier, nombre de bits dans ENS-I et non dans E
j : " " " " " E et non dans ENS-I

Algorithme.

E = ENS-W' (MAXM + 1)
E-I = ENS-I
W-IN ≠ faux
k = 1
i = 0
j = 0
tant que (((E ≠ 0) ou (E-I ≠ 0)) et ((i < 2) et (j < 2))).

On sort de la boucle quand :
- soit on a parcouru ENS-I et ENS-W' (MAXM + 2)
- soit il y a plus de 2 bits différents dans les entiers représentant ces deux ensembles.

```
répéter
    si (mod (E, 2) - mod (E-I,2)) 1,2,3.
1:  i = i + 1
    R = k
    go to 2
3:  j = j + 1
    S = k
2:  E = E div 2
    E-I = E-I div 2
    k = k + 1
```

si (i = 1 et j = 1) alors si ENS-W'(S) = R alors W-IN = vrai.

ADD-W'.

(ajout à W^k d'un élément n'appartenant pas à W^k.)

Spécifications.

arg : MAXM : entier
ENS-W' : tableau de MAXM + 2 entiers
R : entier
S : entier

pré : $0 < MAXM \leq 32$
ENS-W' représente W^k
ENS-W' (S) = 0

rés : ENS-W' : tableau de MAXM + 2 entiers

post : ENS-W' représente W^k donné auquel on a ajouté l'élément représenté par (R, S).

Algorithme.

ENS-W' (S) = R
ENS-W' (MA XM + 2) = ENS-W' (MAXM + 2) + 1.

SUBTR - W'

(suppression d'un élément de W^k)

Spécif ations.

arg : MAXM : entier
ENS-W' : tableau de MAXM + 2 entiers
R : entier
S : entier

pré : $0 < MAXM \leq 32$
ENS-W' représente W^k
ENS-W' (S) = R

rés : ENS-W' : tableau de MAXM + 2 entiers

post : ENS-W' représente W^k donné auquel on a retiré l'élément représenté par (R, S).

Algorithme.

ENS-W' (S) = 0
ENS-W' (MAXM + 2) = ENS-W' (MAXM + 2) - 1.

G E T - W'

(accès à un élément quelconque de W^k)

Spécifications.

arg : M : entier
ENS-W' : tableau de MAXM + 2 entiers.

pré : $0 < M \leq 32$
ENS-W' :représente W^k non vide.

rés : R : entier
S : entier.

post : ENS-W' (S) = R

Variable locale.

i : entier parcourant ENS-W'.

Algorithme.

```
i = 1
tant que i ≤ M répéter
                si (ENS-W' (i) ≠ 0) alors go to 1
                i = i + 1
1: S = i
   R = ENS-W' (S)
```

Modules données - type LISTE.

Vu la pauvreté des structures de données FORTRAN, il est difficile d'isoler la représentation des listes de la représentation des éléments les composant. Par exemple, on aimerait parler de tableaux d'éléments en faisant abstraction de la structure des éléments mais hélas, bien souvent, c'est impossible. On est donc obligé de mélanger les représentations des listes et des éléments de ces listes.

LISTE - R.

R est une liste d'un nombre variable d'éléments de même type et de la forme :

Z : réel

ENS-I: entier de 32 bits représentant l'ensemble des variables de base;

BASIS: tableau de MAXM entiers; (') (BASIS (i)$\leq$ 32).

XBZERO: tableau de MAXM réels (').

On doit pouvoir réaliser efficacement sur R les deux opérations suivantes :

- ajout d'un élément;
- test d'appartenance d'un élément, deux éléments étant considérés comme égaux si leur champ ENS-I est le même. Pour cela, on va organiser R en une sorte de fichier à accès calculé, la clé d'accès étant le champ ENS-I. Plus précisément, R sera représenté par trois tableaux :

  RI : tableau de L x C x (MAXM+1) entiers (BASIS et ENS-I).

  RR : tableau de L x C x(MAXM+1) réels (XBZERO et Z)

  IR : tableau de C entiers, nombre d'éléments dans la C^e colonne de RI ou RR.

On considère une fonction f d'adressage ou de randomisation qui à tout entier ENS-I associé un entier compris entre 1 et C-2 indiquant le numéro de la colonne dans laquelle doit

(') Variable XMP.

se trouver l'élément de valeur de clé ENS-I.

Dans une même colonne les éléments sont rangés séquentiellement.

Les deux dernières colonnes sont des zones de débordement utilisées lorsqu'une des C-2 premières dans laquelle on doit ajouter un élément est remplie.

La fonction f est définie par :

f (ENS-I) = ((ENS-I mod 257) mod (C-2)) + 1 (').

Voici les opérateurs définis sur R.

INIT - R.

(initialisation de R au vide).

Spécifications.

arg : MAXM : entier
L, C : entier

pré : $0 < MAXM \leq 32$
$0 < L, C$

rés : RI, RR : tableau de L x C x (MAXM+1) entiers, réels.
IR : tableau de C entiers.

post : (RI, RR, IR) représente une liste R vide.

Variable locale.

k : entier parcourant IR.

Algorithme.

Pour k de 1 à C faire IR (k) = 0

(') Mod 257 : 257 est un nombre premier et des tests ont montré que ça ne marchait pas trop mal ainsi.

ADD - R.

(ajout à R d'un élément n'appartenant pas à R).

Spécifications.

arg : M : entier
MAXM : entier.
L, C, RI, RR, IR : liste R
Z, ENS-I, BASIS, XBZERO : élément r de même type que ceux de R.

pré : $0 < M \leq MAXM \leq 32$
r non dans R.

rés : L, C, RI, RR, IR : liste R
LL, CC : entier
OK-R : booléen.

post : (OK-R = vrai) et ((L, C, RI, RR, IR) représente la liste R donnée à laquelle on a ajouté r) et (LL: numéro de la ligne où se trouve r) et (CC : numéro de la colonne où se trouve r)
ou
(OK-R = faux) et ((L, C, RI, RR, IR) représente la liste R donnée inchangée (pas de place pour ajouter r) et (LL, CC non significatifs).

Variable locale.

k : entier parcourant RI et RR.

Algorithme.

```
OK-R = vrai
CC = mod (mod (ENS-I, 257), C-2) + 1
LL = IR (CC)
si (LL - L)  1, 2, 3.
1:  | LL = LL + 1
    | IR (CC) = LL
    | Pour k de 1 à M faire | RI (LL, CC, k) = BASIS (k)
    |                       | RR (LL, CC, k ) = XBZERO(k).
```

```
        RI (LL, CC, MAXM+1) = ENS-I
        RR (LL, CC, MAXM+1) = Z
        go to 3
2:  si (IR (C-1) - L) 4, 5, 3.
4:  LL = IR (C-1) + 1
    CC = C - 1
    IR (C-1) = LL
    Pour k de 1 à M faire | RI (LL, CC, k) = BASIS(k)
                          | RR (LL, CC, k) = XBZERO(k)
    RI (LL, CC, MAXM+1) = ENS-I.
    RR (LL, CC, MAXM+1) = Z
    go to 3
5:  si (IR (C) - L)  6, 7, 3.
6:  LL =IR (C) + 1
    CC = C
    IR (C) = LL
    Pour k de 1 à M faire | RI (LL, CC, k) = BASIS (k)
                          | RR (LL, CC, k) = XBZERO (k)
    RI (LL, CC, MAXM+1) = ENS-I.
    RR (LL, CC, MAXM+1) = Z
    go to 3
7:  OK - = faux.
3:  CONTINUE.
```

V - IN - R.

(Fonction booléenne testant l'appartenance de v à R).

Spécifications.

arg : M, MAXM : entier
L, C, RI, RR, IR : liste R
ENS : entier de 32 bits.

pré : $0 < M \leq MAXM \leq 32$
ENS représente $\{ i_1, \ldots, i_m \}$

rés : V - IN-R : booléen.

post : (V- IN - R = vrai) et (il existe un élément de R de champ ENS-I = ENS)
ou
(V- IN - R = faux) et (il existe pas un élément de R de champ ENS-I = ENS)

Variables locales.

V-IN : booléen
k : entier
LL, CC : entier.

Algorithme.

```
V - IN = faux
CC = m... (mod (ENS-I, 257), C - 2) + 1
LL = IR (CC)
k = 1
tant que ((k ≤ LL) et non V - IN) répéter
        si RI (k, CC, MAXM+1) = ENS alors V-IN = vrai
        k = k + 1
si (non V-IN et LL = L) alors
        k = 1
        LL = IR (C-1)
        tant que ((k ≤ LL) et non V-IN) répéter
                si RI (k, C-1, MAXM+1)= ENS alors
                                        V-IN = vrai
                k = k+1
si (non V-IN et LL = L) alors
        k = 1
        LL = IR (C)
        tant que ((k ≤ LL) et non V-IN) répéter
                si RI (k, C, MAXM+1)= ENS alors
                                        V-IN = vrai
                k = k+1
V-IN-R = V-IN
```

LISTE - R'.

R' est une liste de CARD - R' éléments de R s'organisant en une pile. Pour ne pas recopier les éléments de R, on représente R' par un tableau de pointeurs vers les éléments de R.
Plus précisément, R' sera représenté par les variables suivantes :

CARD-R' : entier, # R' ;
L : entier, nombre de lignes de R;
C : entier, nombre de colonnes de R;
I-L : tableau de L x C entiers; I-L(i) est le numéro de le ligne de R du i^e élément de R';
I-C : tableau de L x C entiers; I-C(i) est le numéro de la colonne de R du i^e élément de R';
I-R : tableau de L x C entiers; I-R(1) non significatif; I-R (i > 1) : variable sortante lors du passage du $(i-1)^e$ élément de R' au i^e;
I-S : tableau de L x C entiers; I-S (1) non significatif; I-S (i 1) : variable entrante lors du passage du $(i-1)^e$ élément de R' au i^e.

Voici les opérateurs définis sur R'.

INIT - R'.

(Initialisation de R' au vide).

Spécifications.

arg : --
rés : CARD - R', L, C, I-L, I-C, I-R, I-S : liste R'
post : R' vide.

Algorithme.

CARD - R' = 0

ADD - R'.

(Ajout au sommet de R' d'un élément (non déjà dans R')).

Spécifications.

arg : CARD-R', L, C, I-L, I-C, I-R, I-S : liste R'
LL, CC : entier
R , S : entier.

pré : $1 \leq LL \leq L$
$1 \leq CC \leq C$
$1 \leq R, S \leq M$

rés : CARD-R', L, C, I-L, I-C, I-R, I-S : liste R'

post : R' est R' donnée au sommet de laquelle on a ajouté l'élément (LL, CC,...) de R atteint à partir de l'avant dernier élément de R' par le pivotage ($x_S \uparrow x_R \downarrow$)

Algorithme.

CARD - R' = CARD-R' + 1
I-L (CARD-R') = LL
I-C (CARD-R') = CC
I-R (CARD-R') = R
I-S (CARD-R') = S

SUBTR - R'.

(suppression de l'élément sommet de R')

Spécifications.

arg : CARD-R', L, C, I-L, I-C, I-R, I-S : liste R'

pré : R' non vide

rés : CARD-R', L, C, I-L, I-C, I-R, I-S : liste R'

post : R' est R' donnée de laquelle on a retiré l'élément sommet.

Algorithme.

CARD-R' = CARD-R' - 1

G E T - R'

(accès à l'élément sommet de R')

Spécifications.

arg : CARD-R', L, C, I-L, I-C, I-R, I-S : liste R'

pré : R' non vide

rés : LL, CC, R, S : entier.

post : (LL, CC, R, S) caractérise l'élément sommet de R'.

Algorithme.

LL = I-L (CARD-R')
CC = I-C (CARD-R')
R = I-R (CARD-R')
S = I-S (CARD-R')

LISTE - W

W est une liste de CARD-W (= CARD-R') éléments de type W^k de la forme ENS-W' : tableau de MAXM + 2 entiers.
W est représenté par les variables suivantes :

CARD-W : entier, # W;
L : entier, nombre de lignes de R;
C : entier, nombre de colonnes de R;
W : tableau de (L * C) x (MAXM+2) entiers.

Voici les opérateurs définis sur W :

INIT - W.

(initialisation de W au vide)

Spécifications.

arg : --
rés : CARD-W, L, C, W : liste W
post : W vide.

Algorithme.

CARD-W = 0

W - VIDE.

(fonction testant si W est vide)

Spécifications.

arg : CARD-W, L, C, W : liste W
rés : W- VIDE : booléen
post : W-VIDE = vrai si W est vide, faux sinon.

Algorithme.

W - VIDE = (CARD-W = 0).

A D D - W.

(ajout en fin de liste W d'un élément n'appartenant pas à W)

Spécifications.

arg : M, MAXM : entier
CARD-W, L, C, W : liste W
ENS-W' : tableau de MAXM + 2 entiers

pré : $0 < M \leq MAXM \leq 32$
ENS-W' représente un élément de type W^k

rés : CARD-W, L, C, W : liste W

post : W est W donnée à laquelle on a ajouté ENS-W' comme dernier élément.

Variable locale.

k : entier parcourant W et ENS-W'.

Algorithme.

```
CARD-W = CARD-W + 1
Pour k de 1 à M faire W (CARD-W,k) = ENS-W' (k)
W (CARD-W, MAXM + 1) = ENS-W' (MAXM + 1)
W (CARD-W, MAXM + 2) = ENS-W' (MAXM + 2)
```

G E T - W

(accès au k^e élément de W)

Spécifications.

arg : M, MAXM : entier
CARD-W, L, C, W : liste W
K : entier.

pré : $0 < M \leq MAXM \leq 32$
$0 < K \leq$ CARD-W

rés : ENS-W' : tableau de MAXM + 2 entiers
post : ENS-W' est le k^e élément de W.

Variable locale.

i : entier parcourant W et ENS-W'

Algorithme.

Pour i de 1 à M faire ENS-W' (i) = W (K,i)
ENS-W' (MAXM+1) = W (K, MAXM+1)
ENS-W' (MAXM+2) = W (K, MAXM+2).

SUBTR - W

(suppression du dernier élément de W)

Spécifications.

arg : CARD -W, L, C, W : liste W
pré : W non vide
rés : CARD-W, L, C, W : liste W
post : W est W donnée de laquelle on a retiré le dernier élément.

Algorithme.

CARD-W = CARD-W - 1

M A J - W

(mise à jour du K^e élément de W).

Spécification.

arg : M, MAXM : entier
CARD-W, L, C, W : liste W
K : entier
ENS-W' : tableau de MAXM + 2 entiers

pré : $0 < M \leq MAXM \leq 32$
$0 < K \leq CARD\text{-}W$

rés : CARD-W, L, C, W : liste W

post : W est W donnée où le K^e élément a été remplacé par ENS-W'.

Variable locale.

i : entier parcourant W et ENS-W'.

Algorithme.

Pour i de 1 à M faire W (K, i) = ENS-W' (i)
W (K, MAXM + 1) = ENS-W' (MAXM + 1)
W (K, MAXM+2) = ENS-W' (MAXM + 2)

Modules traitements.

A D D - RR'

Spécifications.

arg : M, MAXM : entier
L, C, RI, RR, IR : liste R
CARD-R', I-L, I-C, I-R, I-S : liste R'
BASIS, XBZERO, Z : sommet
R, S : entier.

pré : $0 < M \leq MAXM < 32$
$0 \leq R, S < M$
(Z, *, BASIS, XBZERO) non dans R (notons v cet élément)

rés : L, C, RI, RR, IR : liste R
CARD-R', I-L, I-C, I-R, I-S :liste R'
OK : booléen.

post : (OK = vrai) et (R est R donnée à laquelle on a ajouté v) et (R' est R' donnée au sommet de laquelle on a ajouté v atteint à partir de l'avant-dernier élément de R' par le pivotage ($x_S \uparrow x_R \downarrow$)
ou
(OK = faux) et (R, R' inchangés).

Variables locales.

ENS-I : entier de 32 bits représentant l'ensemble des variables de base
LL, CC : entier

Algorithme. (')

```
      Call I-BASE (BASIS, ENS-I, M, MAXM).
C     ENS-I est la représentation de l'ensemble des variables
C     de base donné par BASIS
C
      Call ADD-R (BASIS, C, CC, ENS-I, IR, L, LL, M, MAXM, OK,
      RI, RR. XBZERO, Z).
      Si OK alors CALL ADD-R' (C, CARD-R', CC, I-C, I-L, I-R,
      I-S, L, LL, R, S).
```

M A J - WW'

Spécifications .

arg : M, MAXM : entier
N, MAXM : entier
L, C, RI, RR, IR: liste R
CARD-W, W : liste W.
(arguments XMP)
BASIS, BASLB, BASUB, BNDTYP, COLMAX, IOERR, LENMA, LENMY, MAPA, MEMORY, NTYPE 2, STATUS.

(') On vérifie aisément que "pré" des sous-programmes appelés sont satisfaites.

pré : $0 < M \leq MAXM \leq 32$
$0 < N \leq MAXN \leq 32$
+ préconditions dues à la signification des variables XMP.

rés : CARD-W, L, C, W : liste W.

post : (posons v le sommet courant déterminé par BASIS et Γ (v) ses voisins)
W est W donnée à laquelle on a ajouté Γ (v) et retiré R et où chaque sommet est dans un sous-ensemble à distance minimale du sommet v.

Variables locales.

```
i, ix : entier
rr, ss : entier
ENS, ENS-I : entier, ensemble de variables de base
ENS-W', ENS-W" : tableau de MAXM + 2 entiers
W-IN : booléen.
(variables XMP).
COLA, COLI, COLLEN
Q, CQ, LQ, UQ, YQ
INTYP, OUTTYP, PIVOT, R, THETA, UNBDD.
```

Algorithme.

```
      CALL I-BASE (BASIS, ENS-I, M, MAXM)
C     ENS-I représente l'ensemble des variables de base du
C     sommet courant.
      CALL INIT-W'( ENS-I, ENS-W", M, MAXM)
C     ENS-W" représente le sous-ensemble des suivants non
C     encore visités du sommet courant, qui, pour l'instant
C     a été initialisé au vide.
C
C     Pour chaque variable Q hors base, on va déterminer, si
C     possible, la variable sortante et voir si le sommet
C     ainsi obtenu a déjà été visité. Dans le cas où il ne l'a
C     pas encore été, on le place dans un sous-ensemble W^k de W de
C     telle sorte qu'il soit à distance minimale du sommet
C     courant.
```

```
      Pour Q de 1 à N faire :
C     si Q est une variable hors base,...
        si  (STATUS (Q) = 0 ou STATUS (Q) = - 1 alors
C     obtenir la colonne Q sous forme voulue;
          CALL XGETAJ (CQ, ..., Q,...)
          Pour i de 1 à M faire YQ (i) = 0
          Pour i de 1 à COLIEN faire ix = COLI (i)
                                     YQ(ix) = COLA(i)

C     Obteni  les bornes de Q;
          go to (1, 2, 3, 3), BNDTYP
1:        LQ = 0
          UQ = BIG
          go to 4
2:        LQ = 0
          UQ = BOUND
          si (Q > NTYPE 2) alors UQ = BIG
          go to 4
3:        CALL XGETUB (..., Q,..., LQ,..., UQ)
4:        CONTINUE
C     mise à jour de la colonne YQ (YQ = B^-1 YQ) nécessaire
C     avant XCHUZR;
          CALL XFTRAN (YQ,...)
          CALL XCHUZR (..., LQ, ..., Q, ..., UQ, ...,YQ)
C     si une condition de dépassement de limites a été détectée,
C     laisser tomber la variable Q;
          si UNBDD alors go to 6
C     représenter l'ensemble des variables du nouveau sommet;
          CALL MAJ-ENS-I (BASIS (R), ENS, ENS-I, Q )
C     ENS représente l'ensemble des variables de base du nou-
C     veau sommet;
C     voir si le nouveau sommet
C        - a déjà été visité (élément de R ou non)
C        - est voisin d'un sommet déjà visité (élément d'un W^k
C          ou non);
          si v-IN-R (C, ENS, IR, L, M, MAXM, RI, RR) alors go to 6
```

```
C   pour voir si le sommet représenté par ENS est un élément
C   d'un W^k, on parcourt W à partir de la fin;
        pour i de CARD-W à 1 faire
                    | CALL GET-W (C, CARD-W, ENS-W', i, L,
                    | M, MAXM, W) ;
C   ENS-W' est le i^e élément de W;
                    | CALL W-IN-W' (ENS, ENS-W', MAXM, rr,
                    | ss, W-IN)
C   W-IN = vrai si ENS correspond à un élément de ENS-W'
C   faux sinon;     |
                    | si W-IN alors
                    |     | CALL SUBTR-W' (ENS-W',MAXM,rr,ss)
                    |     | go to 5
C   si ENS correspond à un élément d'un W^k , on retire cet
C   élément et on le replace dans le sous-ensemble des sui-
C   vants du sommet courant; cela pour que tout sommet acces-
C   sible à partir de R' soit à distance minimale d'un de
C   ses prédécesseurs;
C
5:          CONTINUE
            CALL ADD-W' (MAXM ENS-W", BASIS (R), Q)
6:      CONTINUE
C   Il reste à ajouter ENS-W" à W.
    CALL ADD-W (C, CARD-W, ENS-W",L, M, MAXM, W)
```

SUIVANT - V.

Spécifications.

```
arg     : M, MAXM : entier
          N, MAXM : entier
          CARD-R', L, C, I-L, I-C, I-R, I-S : liste R'
          CARD-W, W : liste W.
```

(arguments XMP)
B, BASCB, BASIS, BASLB, BASUB, BNDTYP, COLMAX, FACTIT, FACTOR, IOERR, LENMA, LENMI, LENMY, MAPA, MAPI, MEMORY, NTYPE 2, STATUS, UZERO, XBZERO, Z.

pré : $0 < M \leq MAXM \leq 32$
$0 < N \leq MAXN \leq 32$
il existe W' élément de W avec W' non vide
+ préconditions XMP
(notons v le sommet courant donné par BASIS, XBZERO, Z)

rés : CARD-R', L, C, I-L, I-C, I-R, I-S : liste R'
CARD-W, W : liste W

BASCB, BASIS, BASLB, BASUB, STATUS, UZERO, XBZERO, Z.

post : (notons W^k l'élément de W, non vide d'indice maximum)
(BASCB, ..., Z) représente un sommet, à distance minimum de v, dont le prédécesseur est le k^e élément de R'
R' est R' donnée limitée au k premiers éléments
W' est W' " " " " " "

Variables locales.

CARD, i, j, jx : entier
ll, cc : entier
ENS-W' : tableau de MAXM + 2 entiers
R, S : entier, variable
(variables XMP)
CR, CS, LR, LS, UR, US, YR, YS, DR, DS, UZAJ
COLA, COLI, COLLEN
FCODE, INTYP, OUTTYP, LEAVE, PCODE, PIVOT, THETA, UNBDD.

Algorithme.

```
C   A partir du sommet courant, remonter la liste R' jusqu'à
C   un sommet ayant un sous-ensemble de sommets voisins non
C   encore visités non vide.
C
    CARD = CARD-R'
    Pour i de CARD à 2 faire
    CALL GET-W (C, CARD-W, ENS-W',i, L, M, MAXM, W)
```

```
C   ENS-W' est le i^e élément de W;
        si W'-vide (ENS-W', MAXM) alors
C   remonter au sommet i-1 par le pivotage (x_R↑ x_S↓ ) donné
C   par le dernier élément de R';
        CALL GET-R'(C, CC, CARD-R', I-C, I-L, I-R, I-S, L,
                    LL, R, S )
C   Obtenir la colonne relative à la variable entrante R
C   sous forme voulue;
        CALL XGETAJ (CR, ..., R,...)
        Pour j de 1 à m faire| YR (j) = 0
        Pour j de 1 à COLLEN faire| jx = COLI (j)
                                  | YR(jx) = COLA(j)
C   Obtenir les bornes de R;
        go to (1, 2 , 3, 3 ), BNDTYP
1:      LR = 0
        UR = BIG
        go to 4
2:      LR = 0
        UR = BOUND
        si R > NTYPE 2 alors UR = BIG
        go to 4
3:      CALL XGETUB (..., R,..., LR,..., UR)
4:      CONTINUE
C   initialisation de DR et INTYP, paramètres de XPIVOT ;
        CALL XDOT (..., UZAJ)
        DR = CR - UZAJ
        si |DR| < ZLC alors DR = 0
        INTYP = 1
        si DR < 0 alors INTYP = - 1
C   Pivoter R dans la base,
        CALL XPIVOT (..., DR,..., LR,..., R, S,...,UR, XBZERO,
                     YR, Z)
C   ref actorisation de la base si nécessaire;
        FACTIT = FACTIT + 1
        si FACTIT > FACTOR alors
                |CALL XFACT (...)
                |FACTIT = 0
                |CALL XBCOMP (...)
                |
```

```
C   fin de la ref actorisation;
C   mise à jour des variables duales;
        Pour k de 1 à M faire| UZERO (k) = BASCB (k)
        CALL XBTRAN (...)
C   mise à jour des listes R' et W, le sommet courant est
C   maintenant le (i-1)^e;
        CALL SUBTR-W (C,CARD-W, L, W)
        CALL SUBTR-R' (C, CARD-R', I-C, I-L, I-R, I-S, L)

    si non W'-VIDE (ENS-W', MAXM) alors go to 5

C   fin de la boucle "Pour i de CARD à R...";
C   le sommet courant a un sous-ensemble de sommets voisins
C   non encore visités non vide;
C   il reste à déterminer un sommet de ce sous-ensemble;
5:  CALL GET-W' ( ENS-W', MAXM, R, S)
C   effectuer le pivotage (x_S ↑ x_R ↓ ); d'où la même séquence
C   que ci-dessus;
    CALL XGETAJ (CS,..., S,...)
    Pour j de 1 à M faire| YS (j) = 0
    Pour j de 1 à COLLEN faire| jx = COLI (j)
                              | YS(jx) = COLA (j)
    go to (6, 7, 8, 8); BNDTYP.
6:  LS = 0
    US = BIG
    go to 9
7:  LS = 0
    US = BOUND
    si S > NTYPE2 alors US = BIG
    go to 9
8:  CALL XGETUB (..., S,..., LS,..., US)
9:  CONTINUE
    CALL XDOT (...)
    DS = CS-UZAJ
    si |DS| < ZLC alors DS = 0
    INTYP = 1
```

```
si DS < 0 alors INTYP = - 1
CALL XPIVOT (..., DS,...,LS,..,S,R,...,US, XBZERO, YS,Z)
FACTIT = FACTIT + 1
si FACTIT > FACTOR alors
    | CALL XFACT (...)
    | FACTIT = 0
    | CALL XBCOMP (...)
Pour k de 1 à M faire UZERO (k) = BASCB (k)
CALL XBTRAN (...)
C   XPIVOT met à jour la solution courante et les représenta-
C   tions de B^-1.
C   Les arguments de ADD-RR' et MAJ-WW' sont ainsi initialisés.
```

M A J - PLS.

Spécifications.

arg : K : réel (on recherche les solutions K-optimales)
M, MAXM : entier
N, NSTRUC : entier

(arguments XMP)

B, BASCB, BASIS, BASLB, BASUB, BNDTYP, COLMAX, IOERR, LENMA, LENMY, MAPA, MEMORY, ROWTYP, STATUS, UZERO, XBZERO, ;

pré : $0 < K$
$0 < M \leq MAXM \leq 32$
$0 < NSTRUC \leq N \leq 32$
$BNDTYP \neq 2$

+ préconditions XMP.

rés : M, N, NSTRUC : entier
B, BASCB, BASIS, BASLB, BASUB, ROWTYP, STATUS, UZERO, XBZERO.

Post : toutes ces variables représentent le PLS donné auquel on a ajouté la contrainte.

$$\bar{C}_{HB}\, x_{HB} + x_{NSTRUC+1} = K;$$

la base a aussi été mise à jour.

Variables locales.

(variables XMP)

CJ, COLA, COLJ, COLLEN, DJ, ERROR, J, LJ, UJ, ZDOT.

Algorithme.

```
      M = M + 1
      si M > MAXM alors | ERROR = 0
                        | CALL XSTOP (...)
C     la M^e constante est une égalité;
      ROWTYP (M) = 0
C     le terme indépendant est K;
      B (M) = K
C     ajout de la nouvelle variable x_(NSTRUC + 1);
      NSTRUC = NSTRUC + 1
      COLLEN = 1
      COLI (1) = 1
      COLA (1) = 1
      CJ = 0
      CALL XADDAJ (...)
C     ajout des bornes LJ et UJ;
      si BNDTYP = 1 alors | LJ = 0
                          | UJ = BIG
                          | go to 3
      si BNDTYP = 2 alors | ERROR = 3
                          | CALL XSTOP (...)
      si (BNDTYP = 3 ou BNDTYP = 4) alors | LJ = 0
                                          | UJ = K
3:    CALL XADDUB (...)
C     ajout des coefficients non nuls A (M,i);
      COLMAX = COLMAX + 1
      Pour j de 1 à N faire
      |  si (STATUS (j) = 0, -1, -4, -5) alors
C     |  calcul de c̄_j;
      |     CALL XGETAJ (...)
      |     CALL XDOT(..., ZDOT)
      |     DJ = CJ - ZDOT
```

```
        si | DJ | ⩾ ZLC  alors
                COLLEN = COLLEN + 1
                COLI (COLLEN) = DJ
                CALL XADDAJ (..., N - 1)
C       mise à jour de la solution et de la base;
        BASIS (M) = NSTRUC
        STATUS (NSTRUC) = M
        BASCB (M) = 0
        BASLB (M) = LJ
        BASUB (M) = UJ
        UZERO (M) = 0
        XBZERO (M) = K
```

MANAS.

Spécifications.

arg : K : réel (on recherche les solutions K-optimales)
M, MAXM : entier
N, NSTRUC, MAXN : entier
(arguments XMP)

B, BASCB, BASIS, BASLB, BASUB, BNDTYP, COLMAX, FACTIT, FACTOR, IOERR, LENMA, LENMI, LENMY, MAPA, MAPI, MEMORY, NTYPE 2, ROWTYP, STATUS, UZERO, XBZERO, Z.

pré : $0 < K$
$0 < M \leq MAXM \leq 32$
$0 < NSTRUC \leq N \leq MAXN \leq 32$
$BNDTYP \neq 2$

+ préconditions XMP

rés : RI, RR, IR : liste R
OK : booléen.

post : (OK = vrai) et R est la liste de tous les sommets K optimaux)
ou
(OK = faux) et R est la liste d'un certain nombre de sommets K optimaux).

Variables locales.

```
j, R, S : entier
(L,C)CARD-R', I-L, I-C, I-R, I-S : liste R'
CARD-W, W  : liste W
UNION-VIDE  : booléen
ENS-W'  : entier.
```

Algorithme.

```
C   mise à jour du tableau complété;
    CALL MAJ-PLS (...)
C   initialisation de R et R';
    CALL INIT-R (...)
    CALL INIT-R' (...)
C   initialisation de I-R (1) et I-S (1) à 0 via R et S;
    R = 0
    S = 0
C   ajout du sommet optimum à R et R';
    CALL ADD-RR' (...)
C   initialiser W;
    CALL INIT-W (...)
C   mise à jour de W et W', suivants du sommet courant;
    CALL MAJ-WW' (...)
C   tant que l'union des éléments de W est non vide,...
1:  UNION - VIDE = vrai
    j = CARD-W
    tant que ((j ⩾ 1) et UNION-VIDE) faire
        CALL GET-W (C, CARD-W, ENS-W', j, L, M, MAXM, W)
        si W'-VIDE (ENS-W', MAXM) alors j = j - 1
                                  sinon UNION-VIDE = faux
    si non UNION-VIDE alors
C   on détermine le suivant du sommet courant à distance
    minimale;
        CALL SUIVANT-V (...)
C   on en fait le sommet courant;
        CALL ADD-RR' (...)
```

```
C     si on n'a pas pu le rajouter à R, arrêter les recherches;
          | si non OK alors | écrire ("recherches arrêtées-R ")
          |                 | go to 2
C     on met à jour la liste des suivants non encore visités du
      sommet courant
          | CALL MAJ-WW' (...)
C     On reoommence;
          | go to 1
2:    CONTINUE
C     fin.
```

Remarques.

- Pour utiliser l'algorithme de MANAS, il suffit d'écrire un programme principal du genre XDEMO qui en plus, lit la variable K et initialise les variables L et C, dimensions de R,
- On peut aussi rajouter un module de tri et/ou d'impression.

BIBLIOGRAPHIE

1. J. SISKOS, Le traitement des solutions quasi-optimales en programmation linéaire continue, Operations Research, vol.18,n°4,novembre 84, p.381-401.

2. J. FICHEFET, Eléments de programmation linéaire, FNDP, Institut d'informatique, septembre 78.

3. H.W. KUHN, Contribution to the theory of games, vol 2, Princeton University Press,Princeton, New Jersey, 1953.

4. J.P. IGN IO, Goals programming and extensions, Lexington Books, D.C. Heath and Company, Massachusetts, 1976.

5. J. SISKOS, Comment modéliser les préférences au moyen de fonctions d'utilité additives, Recherche Opérationnelle, vol.14,n°1,1980, p.53-82.

6. M.L. BALINSKI, An algorithm for finding all vertices of convex polyhedral sets, Appl. Math.,vol.9, n°1, 1961, p.72-88.

7. M. MANAS, Finding all vertices of a convex polyhedron, Numerische Mathematik, vol.14, 1968, p.226-229.

8. T.H.MATTHEISS, A survey and comparaison of méthods for finding all vertices of convex polyhedral sets, Mathematics of Operations Research, vol.5, 1980, p.167-185.

9. N.V.CHERNIKOVA, Algorithm for finding a general formula... Computational Math. and Math. Physics, vol. V, 1965, p.228-233.

10. A. OMAR, Finding all extreme points...
Ekonomiko-Matematicky, Obzor, vol.3, 1977, p.331-342.

11. H.M. WINKELS, A flexible decision aid méthod for linear multicritéria systems, Collaborative Proceedings Séries CP-82-812, Laxenburg (Austria), 1982, p 377-410.