RAIRO. RECHERCHE OPÉRATIONNELLE

J. MAUBLANC
A. QUILLIOT

Résolution de programmes linéaires entiers ou mixtes à l'aide de la forme normale de Hermite

RAIRO. Recherche opérationnelle, tome 31, n° 4 (1997), p. 399-427

<http://www.numdam.org/item?id=RO_1997__31_4_399_0>

Recherche opérationnelle/Operations Research
(vol. 31, n° 4, 1997, pp. 399 à 427)

RÉSOLUTION DE PROGRAMMES LINÉAIRES ENTIERS OU MIXTES À L'AIDE DE LA FORME NORMALE DE HERMITE

par J. MAUBLANC ([1]) et A. QUILLIOT ([1])

Communiqué par Philippe CHRÉTIENNE

Résumé. – *Un programme linéaire entier ou mixte s'avère en pratique d'autant plus difficile à résoudre que les solutions de la relaxation entière de ce programme sont susceptibles d'être éloignées des véritables solutions du système. Une telle situation tend à se produire quand les angles coniques définis par les sommets du polyèdre sous-jacent au programme relâché sont très aigus ou plats (proches de 0 modulo* Π*).*

Nous présentons ici une approche pour le traitement de tels programmes qui est basée sur l'utilisation de changements de variables permettant de régulariser localement ce polyèdre et de faire apparaître de façon naturelle des candidats à la solution.

Nous en déduisons différentes interprétations algorithmiques, par génération de coupes et sauts aléatoires, par séparation et évaluation, par décomposition de Benders, dont nous vérifions la convergence et testons l'efficacité.

Mots clés : Recherche opérationnelle, Optimisation combinatoire, Programmation linéaire entière, Forme normale de Hermite.

Abstract. – *Integer Linear Programming (ILP) is especially hard when the solutions of its integral relaxation are far from its true solutions. Such a situation is in some way related to the geometrical properties of the associated polyedron. We present here an approach for ILP which is based upon a systematical use of specific changes of variables. These changes of variables involve Hermite Normal Form decompositions and allow the implementation of specific cutting plane methods and branch and bound methods. We describe and test these methods, and conclude this work by introducing several open problems*

1. INTRODUCTION

Le problème de la programmation linéaire entière est connu pour être central de l'optimisation combinatoire. Beaucoup plus que le problème de satisfaction de clauses booléennes, qui sert de référence en théorie de la complexité [13], il permet une traduction simple et parfois automatisable de nombre de problèmes de décision de la classe NP-Temps. Il joue dès lors le

(*) Reçu en mai 1995.

([1]) Université Blaise Pascal, CUST, BP 206, 63174 Aubière.

Recherche opérationnelle/Operations Research, 0399-0559/97/04/$ 7.00

rôle d'une sorte de problème « universel » à l'intérieur de cette classe, de façon beaucoup plus concrète que les autres problèmes NP-Complets.

Les problèmes que l'on choisit de représenter via ce formalisme sont principalement : ([6, 7, 22]).

– Des problèmes combinatoires formulés grâce à des variables de décision (Stable dans un graphe, coloration de graphes, voyageur de commerce...);

– Des problèmes de consistance en logique propositionnelle ou en logique du premier ordre (Détection de pannes, puzzles,...);

– Des problèmes mixtes (ordonnancements disjonctifs, Flow Shop, localisation, multiflots, gestion de production...).

Ce caractère universel de la programmation entière et mixte explique l'importance des travaux qui lui ont été consacrés, la qualité des bibliothèques construites autour de la résolution de ces programmes (CPLEX, XMP, OSL,...), ainsi que les efforts actuellement déployés afin d'insérer des solveurs en nombres entiers au sein des langages de programmation logique et par contraintes (CHIP, PROLOG III, PECOS...).

Les méthodes les plus classiques pour résoudre ces problèmes sont :

– Les méthodes dérivées de l'algorithme du simplexe et basées sur l'exploitation de spécificités telles que la totale unimodularité de la matrice des contraintes [3, 11, 23, 29];

– Les méthodes de coupes, souvent assez lentes [1, 18, 19, 20];

– Les méthodes de séparation et évaluation, s'appuyant sur une relaxation de la contrainte d'intégrité ou bien une relaxation lagrangienne [9, 15];

– Les méthodes de décomposition de type Benders, utilisant la dualité [29].

De façon quelque peu décevante, peu de méthodes parmi celles proposées savent exploiter la géométrie des domaines considérés, ainsi que les propriétés des nombres entiers décrivant le problème. Si le fameux résultat de Lenstra [26] sur la polynomialité de la programmation linéaire entière à dimension fixée s'appuie sur des considérations relatives à l'épaisseur des polyèdres, il ne donne par contre naissance à aucun algorithme efficace. De même l'utilisation des mises en forme normale de Hermite ou Smith de la matrice des contraintes, qui permet de vérifier la polynomialité des systèmes d'équations linéaires en nombres entiers, ne trouve pas d'autre exploitation systématique en dehors de ce cadre. Nous nous sommes donc intéressés à la possibilité d'introduire des arguments de nature géométrique ou arithmétique pour justifier la conception d'algorithmes pour des programmes linéaires entiers ou mixtes.

L'idée principale que nous développons ici consiste à essayer de modifier certaines des caractéristiques algébriques ou géométriques d'un programme linéaire entier par application d'un changement de variables approprié : plus précisément, nous focalisons ici notre attention sur l'utilisation de changements de variables associés au processus de mise en forme normale de Hermite d'une matrice carrée inversible. Si en effet une telle matrice carrée est représentative d'un sommet d'un certain polyèdre défini par une famille d'inégalités linéaires, alors un tel changement de variable conservera le treillis des vecteurs entiers tout en « arrondissant » localement le polyèdre et permettra de faire apparaître un vecteur entier « candidat » à l'appartenance au polyèdre. Nous montrons alors comment ce type de changements de variables, mis en œuvre de façon systématique, induit la construction de méthodes de résolution par séparation et évaluation ou par coupes successives, dont nous testons l'efficacité, et dont nous prouvons la convergence théorique.

Il arrivera que l'efficacité de ces méthodes soit grèvée par des effets de « surplace », qui feront que la solution fractionnaire du sous-problème courant évoluera très lentement. Nous montrons alors comment il peut être avantageux, d'un point de vue pratique, d'avoir recours à un objectif de contrôle linéaire, dont les paramètres varient aléatoirement, et qui permet d'effectuer des sauts autour du polyèdre des contraintes, c'est-à-dire de passer, d'une itération à l'autre, d'un sommet de ce polyèdre à un sommet qui en est relativement éloigné.

Nous travaillons principalement dans le cadre de ce travail sur des systèmes d'inégalités linéaires diophantiennes. Nous concluons en évoquant la possible adaptation des méthodes qui y sont présentées au contexte de programmes linéaires mixtes incluant une fonction objectif, en en présentant quelques prolongements possibles pour notre étude, concernant notamment la possibilité d'agir par changements de variables sur d'autres caractéristiques géométriques des programmes étudiés.

2. FORME NORMALE DE HERMITE ET SYSTÈMES D'INÉGALITÉS LINÉAIRES DIOPHANTIENNES

Rappel Forme Normale de Hermite.

Une matrice entière B à m lignes et n colonnes $(m \leq n)$ est dite en Forme Normale de Hermite (FNH) si elle est triangulaire inférieure sur ses m premières colonnes, nulle sur les autres colonnes, si tous ses éléments $B_{i,i}$, $i = 1...m$, sont positifs, et si sur chaque ligne $i = 1...m$ les éléments non

diagonaux sont négatifs ou nuls et de valeurs absolues strictement inférieures à l'élément correspondant $B_{i,i}$ situé sur la diagonale.

Il est connu [21, 29] que pour toute matrice entière C à m lignes et n colonnes et de rang $m \leq n$, il existe une matrice carrée entière U de taille n, inversible et de déterminant $+1$ ou -1, telle que $B = CU$ soit en FNH. Il est connu aussi que la recherche de U peut se faire en temps polynomial [24], la principale difficulté pour la mise en œuvre de cette recherche tenant au fait que *la taille des nombres susceptibles d'intervenir au cours du processus peut être assez importante.* Il est en fait indispensable, pour programmer cette décomposition, d'utiliser l'une des méthodes dites par « congruence modulo déterminant » (*voir* [10, 24]).

Il s'en déduit qu'un système d'égalités linéaires entières de la forme

$$\{\text{Trouver } x \text{ dans } Z^n, \text{ tel que } C \,.\, x = b\},$$

où b est un vecteur colonne entier de taille m et C est une matrice entière à m lignes et n colonnes, et de rang m, peut se résoudre en temps polynomial [21], grâce au changement de variable $x = U \,.\, X$ associé à la mise en FNH de C, qui conserve la contrainte d'intégrité sur x et transforme le système en un système à solution immédiate.

Considérons à présent le problème de satisfaction d'inégalités linéaires entières suivant :

Problème (P) :

$$\{\text{Trouver } x \text{ dans } Z^n, \text{ tel que } A\,x \leq b\},$$

où A est une matrice entière à m lignes et n colonnes et de rang n $(m \geq n)$, et b un vecteur colonne entier de taille m. Notons P^* le polyèdre défini par : $P^* = \{x \text{ dans } Q^n \text{ tels que } A \,.\, x \leq b\}$.

Remarque : durant toute la suite, nous ferons, à propos d'une telle instance de (P), l'hypothèse que A est de rang n (de plein rang). Cette hypothèse n'est pas restrictive. Dans le cas où elle n'est pas satisfaite, on peut en effet toujours s'y ramener par le biais d'un changement de variable préalable $x = VX$, où V est une matrice entière de déterminant 1 ou -1 et telle que les colonnes de AV indexées de (Rang $(A) + 1$) à n soient toutes nulles. On travaille alors dans $Z^{\text{Rang}\,(A)}$.

Il est connu qu'un sommet de P^* est alors défini par une famille I d'indices lignes incluse dans $\{1...m\}$ telle que la sous-matrice A^I de A induite par les lignes indexées dans I est carrée, inversible et satisfait :

$$A \,.\, (A^I)^{-1} \,.\, b^I \quad \leq \quad b,$$

où b^I est la trace sur I du vecteur b.

Supposons donnée une telle famille I d'indices lignes.

Nous pouvons appliquer à (P) le changement de variables :

$x = U \,.\, X$. U étant obtenue de telle sorte que le produit $A^I \,.\, U$ soit en FNH.

Ce changement de variables a plusieurs effets :

– Il régularise le polyèdre P^* d'un point de vue géométrique, c'est-à-dire qu'il le rend plus « carré », au voisinage du sommet défini par I. Ce phénomène est par exemple illustré par l'exemple suivant :

$$A = \begin{pmatrix} 1 & 10 \\ 1 & -10 \\ -1 & -10 \\ -1 & 10 \end{pmatrix} \quad b = \begin{pmatrix} 10 \\ 0 \\ -1 \\ 9 \end{pmatrix} \quad I = \{1.2\};$$

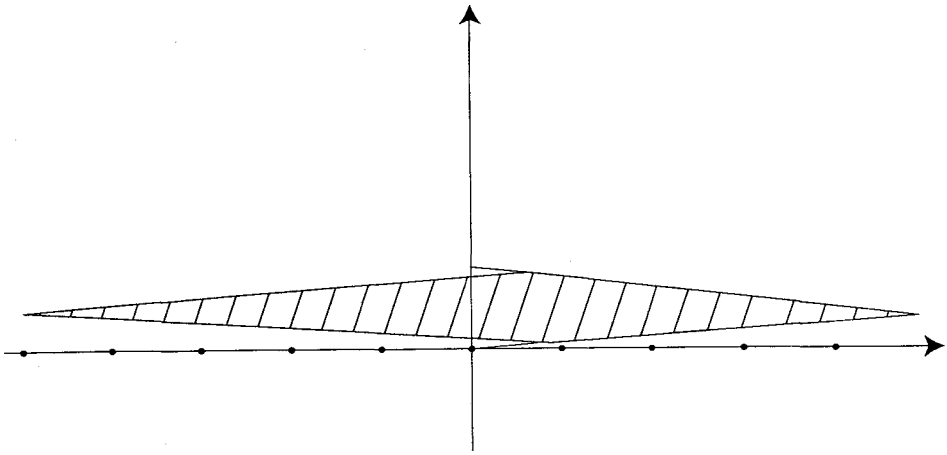
Le sommet de P^* associé à $I = \{1\,.\,2\}$ définit un angle très aigu :

La mise en FNH de A fait apparaître une matrice

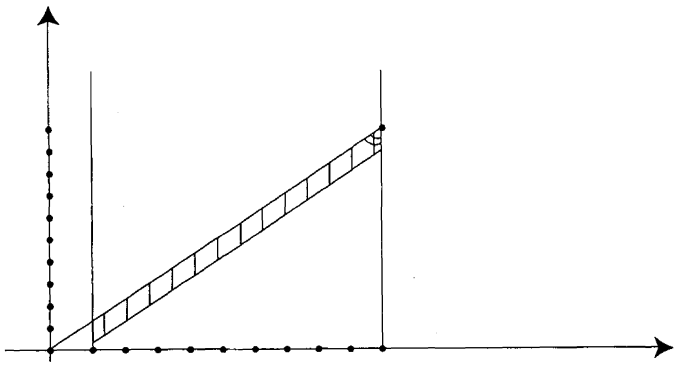
$$\begin{pmatrix} 1 & 0 \\ -19 & 20 \\ -1 & 0 \\ 19 & -20 \end{pmatrix}$$

L'angle autour du sommet défini par I est devenu voisin de 45°.

Représentation graphique du polyèdre P* sous sa forme initiale :



et après changement de variable dû à la mise de A en FNH :



Ceci a des conséquences pratiques, dans la mesure où l'expérience montre que la résolution du problème (*P*) est d'autant plus difficile que les angles coniques définis par les sommets du polyèdre P^* sont proches de 0 modulo Π. Une telle situation tend à se produire en particulier quand l'une des contraintes est de la forme $cx \leq k$, ou k est une quantité que l'on cherche à rendre la plus petite possible, c'est-à-dire quand le problème que l'on traite est en fait un problème d'optimisation en nombres entiers.

Remarque : Cet effet de régularisation s'explique assez bien d'un point de vue mathématique : si B est une matrice inversible de taille n, le sinus de l'angle défini par le cône $\{B \,.\, x \leq d\ x$ dans $Q^n\}$, est égal au quotient de la valeur absolue du déterminant de B par le produit des normes euclidiennes des vecteurs colonnes de B. Si l'on applique à ce cône un changement de variables $x = V \,.\, X$ préservant la contrainte d'intégrité (et donc tel que le déterminant de V soit 1 ou -1), le déterminant de B est conservé, et l'on voit que l'angle conique défini par notre cône sera d'autant plus droit (proche de 90°) que le produit des normes euclidiennes des vecteurs colonnes de la matrice $B \,.\, V$ sera proche du produit des valeurs propres de $B \,.\, V$. Les propriétés de la Forme Normale de Hermite sont telles que l'on tendra vers une telle situation si $B \,.\, V$ est la Forme Normale de Hermite de B. Si en particulier $B \,.\, V$ est diagonale, alors l'angle conique obtenu sera parfaitement droit. Il convient de noter toutefois que le changement de variables associé à la mise en Forme Normale de Hermite de la matrice B n'est pas forcément, parmis tous les changements de variables possibles qui conservent la contrainte d'intégrité, celui qui maximise le sinus de l'angle conique associé au système $B \,.\, x \leq d$.

Le changement de variable évoqué ci-dessus permet de déduire des « coupes », que nous noterons « *Coupes de Hermite associées à I* », et un vecteur entier particulier, que nous nommerons « *Candidat associé à I* » et que nous noterons $\beta(I)$:

Si on pose en effet $B = AU$ et que l'on renumérote les indices lignes de B de telle sorte que $I = 1...n$, alors on voit que l'on peut définir :

$\beta_1 = \lceil b_1/B_{1,1} \rceil$ = *Partie entière inférieure de* $b1/B\,1,1$;

...

$\beta_n = \lceil (b_n - B_{n,1} \cdot \beta_1 - \ldots - B_{n\,.\,n-1} \cdot \beta_{n-1})/B_{n,n} \rceil$;

et déduire les nouvelles contraintes sur X :

$X_i \leq \beta_i$, i dans $1...n$,

ainsi qu'un vecteur entier particulier $\beta(I) = (\beta_1 ... \beta_n)$.

Ce sont ces nouvelles contraintes qui sont nommées « Coupes de Hermite associées à I », tandis que le vecteur $\beta(I)$, repéré par ses coordonnées après changement de variables, peut être vu comme une solution potentielle du système et est donc nommé « Candidat associé à I ».

Ces coupes de Hermite ne sont pas les coupes de Gomory : on peut par exemple vérifier qu'elles ne s'expriment pas forcément à partir de A et b sous la forme :

t.A.x $\leq \lceil b \rceil$, où t est un vecteur fractionnaire positif tel que t.A est entier;

alors qu'il est connu [29] que les coupes de Gomory peuvent s'exprimer de la sorte.

3. UN PREMIER ALGORITHME ET SA CONVERGENCE

De l'ensemble des remarques ci-dessus, nous déduisons un processus de résolution du problème (P), calqué sur le principe des méthodes de résolution par coupes successives : à chaque étape au cours de ce processus, on recherche un sommet fractionnaire du polyèdre courant P^*. Si un tel sommet n'existe pas, alors le problème est bien sûr résolu, sinon il lui est associée une certaine famille I d'indices lignes de A et l'on applique le changement de variables correspondant à la mise en FNH de la sous-matrice de A définie par les lignes de I. Si le sommet Candidat associé à I est solution de (P) est résolu, sinon on insère à la fin du tableau $(A\,b)$ les coupes de Hermite associées à I et on recommence.

Ce processus se formalise alors selon l'algorithme HERMI1 ci-dessous :

```
HERMI1
Données : le système {A.x ≤ b, x dans Z^n représentatif
du problème (P);
Résultats : une solution de ce système s'il en existe;
Début
Not Stop :
Tant que Not stop faire
   Début
   (1) : Déterminer I inclus dans {1...m}, définissant un
   sommet de P*
   et minimal pour l'ordre lexicographique :
   (Commentaire : cette instruction sera détaillée
   au paragraphe IV
   Si I n'existe pas alors
      Début
      Stop; Succès := Faux;
      Fin
   sinon
      Début
      (2) : Appliquer à partir de I le changement de
      variable de Hermite, déduire les coupes de Hermite,
      les insérer à la définition de (P) en fin du tableau
      (A b) :
      Soit α le premier élément de I;
      (3) : Pour i < α, remplacer b_i par b_i - 1 :
      (Commentaire : voir explication ci-dessous)
      (4) : Calculer le candidat β(I) associé à I :
      (5) : Si β(I) est solution de (P) alors
         Début
         Stop : Succès;
         Fin
      Fin
   Fin
Fin.
```

Explication des instructions (1) *et* (3) : La référence à l'ordre lexicographique dans les instructions (1) et (3) permet d'enrichir le système

de ce que l'on appellera les coupes « lexicographiques ». Pour $i < \alpha$ ci-dessus, on ne peut en effet, compte tenu de ce qui a été supposé sur I, trouver un vecteur entier x (repéré par rapport à la base courante) qui soit solution de (P) et tel que $A^i . x \geq b_i$.

Remarque : L'utilisation du sommet candidat $\beta(I)$ possède ici un double intérêt : elle permet d'une part d'accélérer la convergence de la méthode, puisqu'il n'est plus nécessaire d'attendre (comme on le faisait pour les coupes de Gomery) que le sommet courant défini par I soit entier, et d'autre part, elle permet de programmer la méthode presque uniquement en nombres entiers et donc de ne pas avoir à gérer des problèmes d'arrondis. En effet, l'utilisation de l'algorithme du simplexe pour la résolution du système courant relâché de sa contrainte d'intégrité n'a pour but que la mise en évidence de la famille d'indices I et peut s'effectuer à partir d'une copie du tableau $(A\,b)$ courant.

Exemple d'exécution :

Soit le système :

$$\begin{aligned} 3x + 4y &\leq 15; \\ x - 4y &\leq 0; \\ -x + y &\leq -1; \end{aligned}$$

L'instruction (1) fait apparaître $I = \{1, 2\}$, et le changement de variable $X = 3x + 4y$, $Y = x + y$ permet d'obtenir le système :

$$\begin{aligned} X &\leq 15; \\ -5X + 16Y &\leq 0; \\ 2X - 7Y &\leq -1; \end{aligned}$$

la coupe : $Y \leq 4$, et le sommet candidat $\beta(I)$ de coordonnée $x = 1$, $y = 3$, qui n'est pas solution;

Une deuxième itération pourra alors s'appuyer sur $I = \{2, 3\}$ et induit un changement de variables :

$$X1 = x - 4y\,;\; Y1 = -y.$$

Le système réécrit par rapport à ce changement de variables devient :

$$\begin{aligned} 3X1 - 16Y1 &\leq 15; \\ X1 &\leq 0; \\ -X1 + 3Y1 &\leq -1; \\ X1 - 5Y1 &\leq 4. \end{aligned}$$

Une nouvelle coupe de Hermite apparaît alors qui est $Y1 \leq -1$ soit encore $y \geq 1$, ainsi qu'un sommet Candidat défini par $x = 4$ et $y = 1$ et qui n'est pas solution;

L'instruction (3) permet dans le même temps de remplacer la première contrainte $3x + 4y \leq 15$ par $3x + 4y \leq 14$.

Une troisième itération amène $I = \{3, 4\}$. On travaille alors sur les 2 contraintes :

$$-X1 + 3Y1 \leq -1;$$
$$X1 - 5Y1 \leq 4;$$

ou encore

$$-x + y \leq -1;$$
$$x - y \leq 4.$$

Le changement de variables $X = -x + y$, $Y = x + 3y$ produit alors le sommet candidat $x = 2$, $y = 1$ qui est solution.

THÉORÈME 1: *Si le polyèdre P^* est borné, alors l'algorithme HERMI1 converge, c'est-à-dire produit le résultat de (P) en un nombre fini d'itérations.*

Démonstration : On procède ici de la même façon que pour démontrer la convergence de l'algorithme associé aux coupes de Gomory. Supposons qu'une certaine exécution de HERMI1 boucle et procédons par récurrence sur le nombre n de variables afin d'obtenir une contradiction. A partir d'un certain rang, l'indice α utilisé dans l'algorithme devient constant et égal à 1. Si cela n'était pas le cas, le coefficient b décroîtrait à l'infini (du fait de l'instruction (3) relative à l'ordre lexicographique) et le polyèdre P^* ne pourrait être borné. De ce fait, il existe un moment en cours d'exécution de l'algorithme à partir duquel la contrainte numérotée α dans (P) se stabilise sous la forme : $A_{\alpha,1} . X_1 \leq b_\alpha$, b_α étant divisible par $A_{\alpha,1}$.

A partir de ce moment, un rapide calcul permet de voir que l'algorithme se comporte exactement comme s'il ne travaillait plus que sur les variables $X_2, ..., X_n$, la variable X_1 étant assujettie à la valeur $b_\alpha / A_{\alpha,1}$. On conclut alors, du fait de l'hypothèse d'induction. FIN.

Remarque : Si P^* n'est pas borné, il est connu [21, 26] que si une solution entière existe, elle peut être choisie ayant toutes ses coordonnées de valeur absolue au plus égale à $(2n!).\ 2^M$, où M est la taille de codage du tableau (A, b) et donc qu'il est possible de rajouter des contraintes à (P) de façon à rendre P^* borné.

4. IMPLÉMENTATION

Implémentation du Simplexe.

Nous utilisons ici une version de l'algorithme du simplexe qui nous permet d'exécuter l'instruction (1) de l'algorithme HERMI1 sans introduire de variables d'écart, c'est-à-dire en ne travaillant, de façon primale, que sur un système de la forme {Trouver x dans Q^n tel que $A \,.\, x \leq b$}. Cette version, selon laquelle la notion de pivotage se traduit en termes de multiplication à droite de la matrice de contraintes, et donc de changements de variables, est intéressante car peu coûteuse en place, et en harmonie avec l'ensemble du programme. Elle se résume comme suit : à chaque étape au cours du processus de résolution, on disposera (en supposant de plein rang le polyèdre défini par le système $\{A \,.\, x \leq b,\ x \text{ dans } Q^n\}$ que l'on cherche à résoudre), d'une famille I d'indices lignes de A telle que la sous-matrice A^I définie par I soit inversible, et donc d'un vecteur associé $x(I) = (A^I)^{-1} \,.\, b^I$ susceptible de constituer une solution du système. Si $x(I)$ n'est pas solution de ce système, on choisira l'une des contraintes du système qui ne sont pas satisfaites par $x(I)$ et qui aura donc la forme $A^i \,.\, x \leq b_i$; on s'efforcera de modifier I de façon à diminuer la quantité $A^i \,.\, x$ tout en maintenant satisfaites l'ensemble des contraintes du système qui sont vérifiées par $x(I)$, cela jusqu'à permettre le passage de $A^i \,.\, x$ en dessous de b_i ou bien conclure à un échec. L'algorithme correspondant sera donc structuré autour de 2 boucles, l'une (boucle interne) gérant cette phase de minimisation de la quantité cible courante $A^i \,.\, x$, et l'autre permettant d'absorber les contraintes les unes après les autres jusqu'à ce que toutes soient satisfaites. Ces deux boucles pourront alors se détailler suivant les schémas a) et b) ci-dessous :

a) *Supposons que l'on ait à traiter un système (S) :*

$$\{\text{Min}\, z = c \,.\, x \quad \text{pour } x \text{ tel que } A \,.\, x \leq b \quad \text{et} \quad x \text{ dans } Q^n\}$$

où A est une matrice à m lignes et n colonnes pour laquelle on dispose d'une famille I de lignes, telle que la sous-matrice A^I de A définie par les lignes de I soit une matrice de permutation (matrice identité permutée) et b soit un vecteur colonne de taille m tel que chaque coordonnée b_i, pour i dans I, soit nulle.

Remarque : Il est pertinent de supposer que la restriction de la matrice A aux lignes de I est la matrice identité (ou pour le moins une permutation de la matrice identité) dans la mesure où l'on peut toujours, dans l'hypothèse où le rang de A est n et grâce à une élimination de Gauss préalablement

effectuée en travaillant sur les colonnes (et donc un premier changement de variables), se ramener à ce cas. De la même façon, le fait que la trace de b sur I puisse être rendue alors nulle s'obtient par simple translation sur x.

Si $c \leq 0$ alors I définit un sommet optimal pour (S);

S'il existe β dans $1...n$ tel que $c_\beta > 0$ et la colonne A_β est ≥ 0, alors (S) n'est pas borné;

Remarque : Chaque colonne de A contient au moins un élément égal à 1 et donc le cas où une colonne A serait entièrement négative ou nulle n'a pas à être envisagé;

Si aucun des deux cas ci-dessus n'intervient, alors on choisit β dans $1...n$ tel que $c_\beta > 0$ et α dans $1...m$, tel que $A_{\alpha,\beta} < 0$, et minimisant avec cette propriété le quotient $b_\alpha/(-A_{\alpha.\beta})$ (et, afin de contourner les problèmes de dégénérescence, le plus petit possible). On transforme alors le triplet A, b, I de la façon suivante :

$u := A_{\alpha.\beta}$;
$A_\beta := A_\beta/u$:
Pour j différent de β faire
  Début
  $v := A_{\alpha,j}$;
  $A_j := A_j - v.A_\beta$;
  Fin
$w := b_\alpha$;
$b := b - w.A_\beta$;
$I := I - \{$l'élément i de I tel que $A_{i,\beta} = 1\} + \{\alpha\}$.
(Cette transformation sera nommée *Pivotage en α et i*) ;

et on recommence jusqu'à ce que un des 2 cas d'arrêt décrit ci-dessus se présente.

Remarque : Procéder de la sorte revient en fait à travailler sur la forme duale du programme initial à l'aide de la version duale du Simplexe.

b) Supposons maintenant que le système soit du type :

$$\{\text{Trouver } x \text{ dans } Q^n, \text{ tel que } A.x \leq b\},$$

où A est une matrice à m lignes et n colonnes et de rang n, pour laquelle existe I inclus dans $\{1...m\}$ tel que la sous-matrice A^I de A définie par les lignes de I soit une matrice de permutation et b est un vecteur colonne de taille m.

On procède alors de la façon suivante :

```
SIMP-INEG
Données : A, b, I tels que ci-dessus;
Résultats : Échec ou une famille I d'indices lignes définissant
un sommet du polyèdre associé à A et b;

Début
Not Stop;
Tant que Not Stop faire
Début
   Soit x(I) la solution du système A^I . x = b^I ;
   Soit J := {i dans 1...m tels que A^i . x(I) ≤ b_i}
   Si J = {1...m} alors Stop
   sinon
   Début
      Choisir i_0 dans {1...m} − J; Poser c := A^{i0};
      Résoudre (en utilisant l'algorithme présenté en (a)) le
système;
      {Z min = c . x, A^i . x ≤ b_i pour i dans J, −c . x ≤ −b_{i0}};
      Si l'optimum trouvé est plus grand que b_i, alors Stop
et Échec;
   Fin
Fin
Fin.
```

c) Obtention de la minimalité pour l'ordre lexicographique.

Rappel : I et J étant deux sous ensembles de $\{1...m\}$, de cardinalité n et dont les éléments sont rangés par ordre croissant, I sera dit plus petit que J pour l'ordre lexicographique si il existe k dans $1...n$ tel que les $k-1$ premiers éléments de I et J soient identiques et tel que le k^0 élément de I soit plus petit que son homologue de J.

Si l'on souhaite résoudre un système du type posé en *b*) ci-dessus tout en obtenant une solution I maximale pour l'ordre lexicographique, il suffit de compléter le programme SIMP-INEG ci-dessus par les instructions suivantes :

```
Not Stop 1
Tant que Not Stop 1 faire
   Début
   Chercher α dans 1...n, i dans 1...m, tels que le pivotage
   en (α, i) soit réalisable (laisse b positif et tels
   que : α < i;
   Si α et i n'existent pas alors Stop 1 sinon Appliquer
ce pivotage;
   Fin.
```

Implémentation de la mise en FNH.

L'une des principales difficultés liées à la mise en Forme Normale de Hermite d'une matrice réside dans la possibilité de voir apparaître de très grands nombres. Ce phénomène, qui induira des dépassements de capacité sur les types entiers ou entiers longs classiques, peut se produire très vite, pour des petites dimensions (à partir de $n = 5$). Une façon de contourner cet écueil consiste à programmer la mise en FNH modulo le déterminant de la matrice carrée considérée (algorithme de Damich, Kannan et Trotter [10]). C'est ce que nous avons fait. Cette mesure ne suffira pas à effacer complètement la difficulté dans le cas de problèmes générés de façon complètement aléatoire et dont la matrice de contraintes est susceptible de contenir de très grands déterminants, comme nous le verrons au paragraphe suivant. Mais elle permettra par contre d'attaquer des problèmes exhibant une certaine forme de régularité, et pour lesquels on sera capable *a priori* de majorer les sous-déterminants de la matrice de contraintes.

5. INTRODUCTION D'UN CONTRÔLE ALÉATOIRE : RÉSULTATS NUMÉRIQUES

Il peut arriver que la convergence de HERMI1, à l'instar de ce qui se produit pour l'algorithme de Gomory, soit relativement lente. De façon schématique, il peut se faire en effet que le sommet associé à I soit très peu modifié quand on passe d'une itération de la boucle principale à l'autre, à l'issue de la prise en compte des coupes, le « Candidat » $\beta(I)$ demeurant très loin de P^*. Une fois encore, c'est la forme du polyèdre au voisinage du sommet considéré (le fait que l'angle conique associé soit plus ou moins proche de 0 Modulo II), qui pourra être à l'origine d'une telle situation. On peut pallier cet inconvénient en utilisant un objectif de contrôle linéaire, dont les paramètres varient aléatoirement, et qui permet, à l'issue de chaque itération de la boucle principale de l'algorithme, de s'éloigner sensiblement du sommet de P^* courant sur lequel on était préalablement positionné. Le

processus HERMI2 défini ci-dessous et qui vise à résoudre un système du type de (*P*), va alors fonctionner de la façon suivante : à chaque entrée dans la boucle principale, on tirera aléatoirement un vecteur (ligne) c de taille n, et la recherche d'un sommet fractionnaire x de P^* se fera de façon à minimiser la quantité $c.x$. Dans le cas où un tel sommet existe, on génèrera les coupes de Hermite et on effectuera le test sur le sommet Candidat associé, puis on recommencera.

Nous obtiendrons donc, en écriture résumée de ce processus :

```
HERMI2
Données : le système {A.x ≤ b, x dans Z^n};
Résultats : une solution de ce système s'il en existe;
Début
Not Stop :
Tant que Not stop faire
   Début
   Générer aléatoirement un vecteur c dans Z^n;
   Déterminer I inclus dans {1...m} et correspondant à un
   sommet de P* qui maximise c.x;
   Si I n'existe pas alors Stop (Échec)
   sinon
      Début
      Appliquer à partir de I le changement de variable
      de Hermite, déduire les coupes de Hermite, les
      insérer à la définition de (P) et calculer enfin le
      « Candidat associé... » β(I);
      Si β(I) est solution de P alors Stop (Succès);
      Fin
   Fin
Fin.
```

Tests.

L'impact de l'introduction d'un tel contrôle est difficile à évaluer d'un point de vue théorique. Nous avons réalisé des expériences sur des exemples de problèmes de type (*P*) « hors sémantique », et donc dénués *a priori* de toute spécificité, générés de la façon suivante :

Catégorie 1 : Les exemples sont générés par la procédure GEN(n, m, u, k, l) de telle sorte que :

– n, m, l, u sont entiers ≥ 0, k est entier relatif, $k \leq 1$;

– Les n variables $x_1 ... x_n$ sont bornées en valeurs absolues par 10;

– Les coefficients de A sont tirés aléatoirement entre $-u$ et u;

– Les coefficients de b sont tirés aléatoirement entre $k \,.\, n \,.\, u$ et $l \,.\, n \,.\, u$;

(si $k \leq 0$, on est sûr que (P) admet la solution $x = 0$).

Catégorie 2 : on ajoute à un exemple de la catégorie 1, tiré avec $k = 0$, une contrainte $c \,.\, x \leq [V] + 1$, c étant tiré aléatoirement selon les mêmes lois que les autres coefficients de A et V constituant la valeur optimale du programme :

$$\{Z \min = c \,.\, x, \quad \text{pour } x \text{ tels que } A\,x \leq b \text{ et } x \text{ dans } Q^n\}.$$

La catégorie 2 correspond donc à la génération de problèmes (P) plus difficiles, associés à des polyèdres P^* de faibles volumes et présentant des angles assez plats ou aigus autour de certains sommets.

En fait, il apparaît que les mises en FNH induisent des phénomènes de grands nombres (et donc de dépassement de capacité dans les types entiers longs) et tendent à produire des matrices mal conditionnées dès lors que les déterminants des sous-matrices carrées de A sont trop grands. Nous nous sommes donc limités pour nos tests à ceux des exemples générés dans les catégories 1 et 2 qui ne font apparaître aucun tel sous-déterminant de valeur absolue plus grande que 10^5.

Nous n'avons pas retenu le temps CPU comme critère d'évaluation : ce temps dépend trop du langage et du mode de programmation utilisé. Nous n'avons pas non plus focalisé notre attention sur le nombre d'exécutions de l'algorithme du simplexe ou sur le nombre de mises en FNH, dans la mesure où le coût de ces opérations dépend largement de la capacité que l'on a à programmer de façon incrémentale et à établir une certaine continuité entre les différentes étapes de calcul (point que nous n'avons pas cherché à optimiser). Pour chacun des exemples ainsi générés, nous nous sommes donc simplement intéressés aux quantités suivantes :

– Nombre d'itérations de la boucle principale de HERMI1 et HERMI2 en cas de non existence d'une solution :

– Nombre d'itérations de la boucle principale de HERMI1 et HERMI2 en cas d'existence d'une solution;

et nous avons procédé de façon comparative, en prenant pour références, selon les mêmes critères, l'algorithme classique de Gomory.

Les résultats peuvent alors se résumer selon les tableaux I et II suivants :

Catégorie 1. (Moyennes obtenues pour n entre 10 et 40 m entre 20 et 100, chaque quantité correspondant à 20 essais).

TABLEAU I

Comparaison entre les nombres de coupes engendrées par les algorithmes GOMORY, HERMI1 et HERMI2 pour des problèmes de la catégorie 1, générés aléatoirement. Sur chaque ligne du tableau sont séparés les cas où le système admet une solution et les cas où il n'en admet pas.

n		GOMORY	HERMI1	HERMI2
10	Solution	27	11	3,4
	Non solution	36	15	5,1
20	Solution	52	26	8,2
	Non solution	78	35	13
30	Solution	97	38	16
	Non solution	158	58	29
40	Solution	201	80	32
	Non solution	311	120	47

Catégorie 2.

TABLEAU II

Comparaison entre les nombres de coupes engendrées par les algorithmes GOMORY, HERMI1 et HERMI2 pour des problèmes de la catégorie 2 générés aléatoirement. Sur chaque ligne du tableau sont séparés les cas où le système admet une solution et les cas où il n'en admet pas.

n		GOMORY	HERMI1	HERMI2
10	Solution	37	18	6,2
	Non solution	48	24	8,2
20	Solution	80	40	13
	Non solution	94	50	17
30	Solution	196	75	25
	Non solution	245	84	33
40	Solution	410	165	56
	Non solution	450	180	65

Commentaires : On constate l'impact très positif de l'introduction du contrôle aléatoire. L'amélioration obtenue en ce qui concerne le passage de Gomory à HERMI2 doit être pondérée par le fait que chaque itération de HERMI2 implique une mise en FNH d'une sous-matrice carrée de la matrice courante de contraintes, et donc un calcul supplémentaire. Le fait que nous ayons dû nous restreindre quant à la taille des sous-déterminants de la matrice A suggère que les applications de ce type de méthodes se

trouveront plutôt autour de problèmes définis par des matrices construites à partir d'un noyau totalement unimodulaire par addition de lignes (contraintes) ou colonnes (variables) qui rompent cette propriété de totale unimodularité tout en permettant de majorer *a priori* les sous-déterminants de la matrice A.

6. UN PROCESSUS DE SÉPARATION/TEST ASSOCIÉ AUX COUPES DE HERMITE

Nous venons de voir qu'une exploitation algorithmique des propriétés des changements de variables associés aux coupes de Hermite consistait en une génération de coupes successives, couplée avec des tests sur un certain sommet « Candidat » associé à ces coupes.

Une autre exploitation possible consiste en l'application d'un processus d'exploration arborescente (Try/test) filtré de la même façon par insertion de coupes de Hermite et test sur le sommet Candidat associé à ces coupes.

Le principe en est simple :

– Supposons le problème (P) donné sous la forme :

$$\{\text{Trouver } x \text{ dans } Z^n \text{ tel que } A \,.\, x \leq b\},$$

A étant une matrice à m lignes et n colonnes, de rang n et entière, b étant un vecteur colonne entier de taille m.

Le changement de variable $x = U \,.\, X$ associé à la mise en FNH de la sous-matrice A^I définie par un sommet du polyèdre P^* permet de faire apparaître une première coupe $X_1 \leq \beta_1$ (*cf.* notations précédentes). A cette coupe correspond naturellement, si l'on se place dans la perspective de la résolution de (P) via une exploration arborescente, une séparation en 2 alternatives :

$$X_1 = \beta_1 \qquad \text{et} \qquad X_1 \leq \beta_1 - 1;$$

La première de ces deux options offre l'avantage d'être d'une part la plus intuitivement plausible et d'autre part de ramener le problème à un sous-problème de dimension inférieure.

Le processus appliqué sera alors le suivant :

(1) Générer aléatoirement c dans Z^n;
Résoudre le programme : $\{A \,.\, y \leq b,\ y \text{ dans } Q^n \,.\, z \min = c \,.\, y\}$;
Si ce programme n'a pas de solution alors Échec et Stop
sinon Continuer en (2) en notant I le sous-ensemble de $\{1...m\}$ qui définit un sommet solution de ce programme :

(2) Générer le changement de variable $x = U.X$ associé à la mise en FNH de A^I;

Par rapport à ce changement de variable, générer les coupes $X_i \leq \beta_i$, $i = 1...n$, et le sommet candidat $\beta(I)$ associé à I défini par : $\beta(I) = \beta_1...\beta_n$;

Si $A.\beta \leq b$ alors Succès et Stop sinon Continuer en (3);

(3) Résoudre récursivement le problème (*P*), reformulé par rapport à X et enrichi des coupes $X_i \leq \beta_i$, $i = 2...n$, et de la contrainte $X_1 = \beta_1$;

Si une solution apparaît, alors cette solution est (à changement de variable près), la solution de (*P*), sinon résoudre récursivement le problème (*P*), formulé par rapport à X et enrichi des coupes $X_i \leq \beta_i$, $i = 2...n$ et de la contrainte $X_1 \leq \beta_1 - 1$.

Commentaires : Le processus décrit ci-dessus est muni d'un double filtre : d'une part, la résolution du sous-problème courant relâché de sa contrainte d'intégrité permet, de façon classique, d'anticiper un échec, et d'autre part le test sur le point candidat courant permet à la fois d'anticiper sur l'échec et d'éviter d'avoir à gérer des problèmes d'arrondis. La méthode peut être en effet, pour sa partie principale, programmée entièrement en nombres entiers, la résolution du sous-problème relâché pouvant s'effectuer sur une copie du système courant et n'ayant pour but que de mettre en évidence une famille d'indices lignes particulière.

L'implémentation effective de ce processus exige, si l'on veut éviter de réaliser des empilements de matrices, qu'à chaque retour arrière on soit en mesure de remonter la chaîne des changements de variables qui ont été effectués. La matrice A de contraintes et le vecteur b sont gérés en variables globales; le système est contrôlé à l'aide d'une pile PILE, chaque étage de la pile utilisée pour la gestion du bactracking étant susceptible de contenir :

– Un coefficient ÉTAT permettant à chaque instant de connaître le type d'action à effectuer;

– La famille I d'indices lignes définissant le sommet d'appui pour la résolution du sous-problème courant;

– L'indice Col de A qui correspond à la dimension du sous-problème courant;

– L'indice Lig qui indique le nombre de contraintes dans le sous-problème courant;

– Le nombre CoeffCol qui permet de mettre à jour le vecteur b chaque fois qu'un essai d'affectation d'une des variables du problème courant est effectué;

– Une liste L-PIVOT permettant, à chaque retour arrière, de revenir sur les changements de variables qui ont été effectués et de reconstituer la matrice A;

L'algorithme SEP-HERMI correspondant fonctionnera alors comme suit :

– La pile PILE sera initialisée de façon à décrire le système dans son état initial, la variable ÉTAT étant mise à 0;

– L'algorithme prendra fin quand une solution aura été trouvée ou bien quand PILE sera vide;

– A chaque entrée dans la boucle principale, on regardera la variable ÉTAT contenue au sommet de la pile : Cette variable, dont les valeurs possibles iront de 0 à 5 et détermineront le contenu de l'action réalisée, sera alors incrémentée. Puis :

– Si ÉTAT = 0, on résoudra la relaxation entière du sous-problème courant décrit sur la pile par Col et Lig, et on dépilera en cas d'échec, en restituant le système défini à l'étage inférieur grâce à L-PIVOT; (en cas de succès, ÉTAT sera alors mis à 1)

– Si ÉTAT = 1, on appliquera à ce sous-problème le changement de variables correspondant à la mise en FNH de la sous-matrice carrée associée à la solution ainsi trouvée et on mémorisera les opérations décrivant ce changement de variables;

– Si ÉTAT = 2, on déduira du changement de variables précédent les coupes de Hermite et le sommet Candidat; Si le sommet Candidat est solution du système, alors on aura fini, et il faudra reconstituer la solution compte tenu des changements de variables effectués, sinon on insèrera les coupes de Hermite en fin du tableau A;

– Si ÉTAT = 3, on empilera les données descriptives du sous-problème obtenu en contraignant la coordonnée du vecteur inconnu dont l'indice est fourni par la variable Col à être égale à sa plus grande valeur possible, contenue dans la variable Coeffcol;

– Si ÉTAT = 4, on empilera les données descriptives du sous-problème obtenu en contraignant la coordonnée du vecteur inconnu dont l'indice est fourni par la variable Col à être au plus égale à Coeffcol–1;

– Si ÉTAT = 5, on dépilera.

Un schéma d'implémentation plus détaillée est proposé ci-dessous : (nous identifions dans ce schéma la pile PILE et son sommet)

SEP-HERMI

Données : Le problème (P), ainsi que I inclus dans $\{1...m\}$ tel que la restriction A^I de A à I soit une matrice identité;

Résultats : Une solution x de (P) s'il en existe une et un message Échec sinon;

Début

Not Stop: Undefined(Sol); PILE:=Vide;

Empiler $[0, I.m, 1, \text{Nil}, \text{Nil}]$;

Tant que Not Stop et Not Vide(PILE) faire

Début

Case of:

PILE^ÉTAT=0:

Début

(O1): Générer aléatoirement un vecteur c indexé de 1 à n;

Copier la restriction A^* de A aux indices colonnes allant de PILE^Col à n, et déterminer à partir de cette copie une famille d'indices lignes J définissant un sommet du problème :

{Trouver x indexé de PILE^Col à n et à valeurs dans Q tel que $A^*x \leq b$ et maximisant $c.x$} :

Si J n'existe pas alors Dépiler

sinon

Début

PILE^I:=J; PILE^ÉTAT:=1;

Fin

Fin

PILE^ÉTAT=1 :

Début

Mettre en FNH la restriction de A aux indices colonnes allant de PILE^Col à n et aux indices lignes de J;

Déduire une liste PILE^L-PIVOT représentative des changements de variables effectués et appliquer ces changements de variables à l'ensemble de la matrice A;

PILE^ÉTAT:=2;

Fin

PILE^ÉTAT=2 :

Début

Calculer CAND, le candidat associé à I pour le problème $\{A^*.X \leq b,\ X$ entier, indexé de PILE^Col à n, où A^* est la restriction de A aux indices colonnes allant de PILE^Col à $n\}$;

PILE^Coeff-Col:= $\text{CAND}_{\text{PILE}^{\wedge}|\text{col}}$;

Si A. $CAND \leq b$ alors

Début

Stop; Reconstituer SOL grâce aux listes PILE^L-PIVOT;

Fin

sinon

Début

PILE^État:=3;

Insérer les nouvelles coupes de Hermite en fin de Tableau;

Incrémenter PILE^Lig et faire PILE^I:={indices associés à ces coupes};

Fin

Fin

PILE^ÉTAT=3 :

Début

b:=b-PILE^Coeff-Col. $A_{\text{PILE}^{\wedge}\text{Col}}$;

PILE^ÉTAT:=4;

Empiler {0, Queue(PILE^I), PILE^Lig, PILE^Col+1, Nil, Nil};

Fin

PILE^ÉTAT=4 :

Début

PILE^ÉTAT:=5;

b:=b+PILE^.Coeff-Col. $A_{\text{PILE}^{\wedge}\text{Col}}$;

Empiler {0, I, PILE^Lig+1, PILE^Col, Nil, Nil};

Ajouter à $(A\,b)$ la ligne (0.....010..0) $(\text{CAND}_{\text{PILE}^{\wedge}\text{Col}}-1)$;

PILE^Col

Fin

```
        PILE^ÉTAT=5 :
           Début
           Reconstruire A grâce à PILE^L-Pivot:
           Dépiler;
           Fin
        Fin
     Fin
  Si Undefined(Sol) alors Échec:
  Fin.
```

Tests numériques sur SEP-HERMI.

Nous avons procédé exactement comme pour l'évaluation de HERMI2, en testant pour les catégories de problèmes 1 et 2 définies au paragraphe IV :

– Le nombre de nœuds de l'arbre d'exploitation visités suivant que l'on est dans une configuration d'échec ou de succès;

– La même quantité dès lors que l'on renonce à l'instruction (O1) concernant l'objectif de contrôle c généré aléatoirement : Option SEP-HERMI-PASSIF.

Nous avons comparé avec une méthode par séparation filtrage, nommée SEPAR-SIMPLE basée sur une simple relaxation de la contrainte d'intégrité.

Les résultats sont alors résumés dans les tableaux III et IV suivants :

Catégorie 1.

TABLEAU III

Comparaison entre les nombres de séparations engendrées par les algorithmes SEPAR-SIMPLE, SEP-HERMI et SEP-HERMI-PASSIF pour des problèmes de la catégorie 1 générés aléatoirement. Sur chaque ligne du tableau sont séparés les cas où le système admet une solution et les cas où il n'en admet pas.

n		SEPAR-SIMPLE	SEP-HERMI	SEP-HERMI-PASSIF
10	Solution Non solution	13 14	3,5 5,7	5,2 8
20	Solution Non solution	32 31	8 11	13 16
30	Solution Non solution	58 61	17 20	25 29
40	Solution Non solution	105 102	33 39	45 52

Catégorie 2.

TABLEAU IV

Comparaison entre les nombres de SEPAR-SIMPLE, SEP-HERMI et SEP-HERMI-PASSIF pour des problèmes de la catégorie 2 générés aléatoirement. Sur chaque ligne du tableau, sont séparés les cas où le système admet une solution et les cas où il n'en admet pas.

n		SEPAR-SIMPLE	SEP-HERMI	SEP-HERMI-PASSIF
10	Solution	21	7	8,5
	Non solution	23	10	11
20	Solution	45	13	16
	Non solution	50	18	20
30	Solution	102	27	35
	Non solution	96	35	42
40	Solution	202	55	74
	Non solution	194	64	76

Commentaires : On constate à nouveau une amélioration sensible due à l'utilisation de l'instruction relative au contrôle aléatoire. On retrouve aussi le fait que en cas de succès, l'émergence à chaque étape d'un sommet « candidat » permet d'abréger très fortement la recherche arborescente. Les problèmes liés au conditionnement de certaines matrices intermédiaires rendent toutefois la méthode SEP-HERMI difficile à implémenter et tester et obligent à envisager une limitation des champs d'application. De ce fait, nous avons, reprenant ce qui a été suggéré à la fin du paragraphe concernant les tests pour HERMI1 et HERMI2, testé SEP-HERMI sur une troisième catégorie de problèmes, construits comme suit :

Catégorie 3 : Les systèmes $\{A\,x \leq b,\ x \text{ dans } Z^n\}$ considérés sont générés par la procédure GENBIS(n, m, nl, ml, u, k, l, t) de telle sorte que :

– n, m, u, k, et l ont la même signification que dans la procédure GEN appliquée pour obtenir les exemples des deux premières catégories, à la différence cependant que ceux des coefficients de A qui sont tirés aléatoirement entre u et v concernent seulement les colonnes allant de $n\,1 + 1$ à n et les lignes allant de $m\,1 + 1$ à m;

– $n\,1 \leq n$ et $m\,1 \leq m$ sont tels que la sous-matrice de A définie par les lignes $1...m\,1$ et les colonnes $1...m\,1$ est une matrice graphique, c'est-à-dire une matrice d'incidence arcs/chemins d'une famille de chemins élémentaires d'un arbre dont les arêtes ont été arbitrairement orientées, perturbée par insertion de t coefficients 1 ou -1 additionnels;

Si $m - m1$ et $n - n1$ sont assez petits, il devient alors en pratique possible de contrôler la taille des sous-déterminants de la matrice A. Pour par exemple $n = 20$, $n1 = 15$, $m = 30$, $m1 = 25$, $u = -3$, $v = 3$, $t = 20$, les problèmes posés par l'apparition de sous-déterminants trop grands disparaissent complètement et l'on constate alors un très bon comportement de l'algorithme SEP-HERMI, comme le montre le tableau V suivant :

TABLEAU V

Comparaison entre les nombres de séparations effectuées par SEP-HERMI et SEPAR-SIMPLE pour des exemples de catégorie 3, créés avec $n = 20$, $m = 30$, $n1 = 15$, $m1 = 25$, $u = -3$, $v = 3$, $t = 20$. Les exemples sont séparés suivant que le système considéré admet ou non une solution.

	SEPAR-SIMPLE	SEP-HERMI
Solution	14	4,5
Non solution	16	6,8

Commentaire : Beaucoup de problèmes réels s'articulent ainsi autour d'un noyau totalement unimodulaire augmenté de quelques contraintes et variables induisant une perte de cette propriété de totale unimodularité. Il en est ainsi les problèmes de multiflots entiers ou de couplages généralisés. Les résultats fournis alors par ces dernières expérimentations indiquent une certaine efficacité des méthodes que nous venons de présenter dans des contextes rendant possible une anticipation sur la taille des sous-déterminants de la matrice de contraintes du problème, et donc sur la taille des nombres susceptibles d'apparaître au fil du processus de résolution.

7. CONCLUSION : EXTENSIONS ET PERSPECTIVES

Les résultats numériques enregistrés sur les deux méthodes par coupes de Hermite et par séparations de Hermite et filtrage nous paraissent intéressants, en ce sens qu'il font apparaître un gain de performance dû à l'utilisation du sommet candidat et à un effet de régularisation locale du polyèdre de contraintes. L'exploitation systématique de ces méthodes ou de méthodes dérivées impliquera cependant d'une part de savoir mettre en œuvre une certaine forme de lien entre les calculs successifs de FNH et les applications du simplexe, et d'autre part de pouvoir mieux cerner les domaines d'application concernés (systèmes à sous-déterminants bornés). En terme de conclusion, nous pouvons faire apparaître un certain nombre de problèmes, qui se situent dans le prolongement de ce que nous venons de présenter et qui nous semblent pouvoir intéresser d'autres chercheurs.

1. *Problème : L'étude de l'extension de nos méthodes au cas d'un problème de la forme :*

(POPT) : { Trouver x dans Z^n, tel que $A \,.\, x \leq b$ et minimisant $d \,.\, x$},

A étant une matrice entière à m lignes et n colonnes, b étant un vecteur colonne entier de taille m, d étant un vecteur ligne entier de taille n;

Afin de résoudre un tel problème, on peut :

– appliquer HERMI2 en substituant au choix d'un objectif aléatoire c (instruction (*)), le choix de d (réécrit compte tenu du changement de variables courant) comme objectif déterministe.

– appliquer SEP-HERMI en appliquant la même substitution, et en filtrant le procesus d'exploitation en arbre par l'utilisation de la borne fournie par l'instruction (O1) de SEP-HERMI (Branch/Bound);

– travailler en résolvant une succession de problèmes de la forme :

$$\{\text{Trouver } x \text{ dans } Z^n \text{ tel que } A\,x \leq b,\, d \,.\, x \leq k\},$$

le paramètre k étant alors rendu progressivement le plus petit possible.

La dernière de ces 3 démarches, peu conventionnelle, permettrait ici de tirer parti des remarques que nous avons faites et qui concernent l'introduction d'un objectif de contrôle évoluant aléatoirement; il serait dès lors intéressant de la comparer dans ce contexte aux approches classiques.

2. *Problème : L'étude de l'extension de nos méthodes au contexte de la programmation linéaire en nombres mixtes de la forme* :

(PMIX) : {Trouver x dans Z^n, y dans Q^p tels que $A\,x + B\,y \leq b$},

A et B étant deux matrices entières avec m lignes et respectivement n et p colonnes, et b étant un vecteur colonne entier de taille m.

Les méthodes évoquées plus haut ne s'exploitent pas ici directement. Si I est une famille d'indices lignes qui définit un sommet du polyèdre défini dans Q^{n+p} par le tableau $(A\,B\,b)$, un changement de variable sur (x, y) conservant l'intégrité de x permettra au mieux de mettre le couple $(A^I\,B^I)$ sous la forme :

$$\begin{matrix} (0 & \text{id}) \\ (S & H) \end{matrix}$$

où S est en FNH et où H est une matrice quelconque fractionnaire, ce qui ne sera pas suffisant pour déduire une notion de coupe.

On peut envisager de se raccrocher au schéma de décomposition dit de Benders [29]. Ce schéma consiste à travailler tour à tour sur x et sur y, en utilisant x comme variable principale. Cette variable est donc au départ non

contrainte et initialisée par approximation entière de la solution fractionnaire de (PMIX). A chaque étape au cours du processus, on résout d'abord (PMIX) en y, x étant fixé, puis en cas d'absence de solution, on extrait par dualité un vecteur $t \geq 0$, indexé sur les lignes de B, tel que $t \,.\, B = 0$ et que $t \,.\, (b - A\,x) < 0$, on injecte la contrainte $t \,.\, (b - A\,x) \geq 0$ dans le système courant de contraintes portant sur x, et on ajoute x en tenant compte de cette nouvelle contrainte. Le procédé se résume donc à la résolution en alternance d'une succession de programmes linéaires fractionnaires en y et d'une succession de programmes linéaires entiers en x. Ces derniers évoluent de façon incrémentale. Il serait donc alors intéressant de préciser la façon dont les algorithmes HERMI2 et SEP-HERMI s'adaptent à ce traitement de systèmes évoluant de façon incrémentale et de tester numériquement leur comportement dans ce contexte.

3) *Problème* : *La recherche de changements de variables susceptibles d'induire une régularisation globale du polyèdre P^* associé à un problème de type (P).*

Nous avons, au cours de ce travail, concentré notre attention autour de la possibilité d'utiliser avantageusement des changements de variables associés à la mise en Forme Normale de Hermite d'une matrice carrée, dans le cadre de la résolution d'un problème (P) de la forme :

{Trouver x dans Z tel que $A \,.\, x \leq b$, où A est une matrice entière à m lignes et n colonnes et b un vecteur colonne de taille $m \geq n$}.

Ces changements de variables sont du type $x = U \,.\, X$, ou U est une matrice carrée entière inversible, et leur application, à partir d'une sous-matrice carrée de A définie par une famille I de lignes de A de cardinalité n, permet de rendre triangulaire la sous-matrice A^I et de modifier la représentation du polyèdre P^* au voisinage du sommet associé à I.

Cette modification de la structure de P^* n'est cependant avantageuse que localement et les changements de variables en question doivent donc être réitérés tout au long des processus de résolution que nous avons pu décrire ici. On peut se poser la question de savoir s'il est possible de dégager des critères globaux de régularité du polyèdre P^* qui reflètent la complexité pratique du problème (P) ainsi que des changements de variables qui permettent alors de modifier la valeur de ces critères.

Plus précisément, on peut par exemple définir ce qu'on nommera l'épaisseur du polyèdre P^*, par la formule :

$$\mathrm{Th}\,(P^*) = \inf_{c/\|c\|=1\,x,} \quad \sup_{x' \text{ dans } P^*} \quad \mathrm{c} \,.\, (\mathrm{x} - \mathrm{x}')$$

Intuitivement, pour V donné, il sera d'autant plus plausible qu'un polyèdre P^* de volume V puisse se « faufiler » entre les vecteurs entiers du treillis Z, que son épaisseur $\mathrm{Th}(P^*)$ sera petite. On pourra donc se poser les deux questions suivantes :

– Le fait d'appliquer au problème (P) un changement de variable $x = UX$ qui préserve les contraintes d'intégrité et augmente l'épaisseur de P^* (le volume de P^* restera forcément égal à V) serait-il susceptible d'améliorer de façon significative les performances des heuristiques travaillant sur (P) ?

– Comment faire pour déterminer pratiquement de tels changements de variables ?

REFERENCES

1. E. B. Balas, Intersection n-cuts–a new type of cutting planes for integer programming, *Operat. Research,* 1971, *19*, p. 19-39.
2. F. L. Bauer, Algorithms 153 Gomory; *Comunications of the ACM,* 1963, *6*, p. 68.
3. R. Bixby et W. Cunningham, Converting linear programs to network problems; *Maths of Operat. Research*, 1980, *5*, p. 321–357.
4. V. J. Bowman et G. L. Nemhauser, A finiteness proof for modified Dantzig cuts in integer programming: *Naval Research Log Quarter*, 1970, *17*, p. 309-313.
5. V. J. Bowman et G. L. Nemhauser, Deep cuts in integer programming; *Operat Research,* 1971, *8*, p. 89-111.
6. M. Carter, A survey on practical applications of examination timetabling algorithms; *Operat Research,* 1986, *34*, 2, p. 193-302.
7. A. Charnes et W. Cooper, *Management models and industrial applications of linear programming*, J. Wiley and sons, 1961.
8. V. Chvatal, Cutting planes and combinatorics; *European Journal of combinatorics,* 1985, *6*, p. 217-226.
9. R. J. Dakin, A tree search algorithm for mixed integer programming problems, *The Computer Journal*, 1965, *8*, p. 250-255.
10. P. D. Damich, R. Kannan et L. Trotter, Hermite normal form computation using modulo determinant arithmetic: *Math. Operat. Research*, 1987, *12*, 1, p. 50-59.
11. J. Edmonds, F. Giles, Total dual integrality of linear inequality systems in *Progress in Combinatorial Optimization*, Acad. Press. Toronto, 1984, p. 117-129.
12. D. Fayard et G. Plateau, An efficient algorithm for the 0-1 knapsack problem, R. M. Nauss, *Management Sciences*, 1977, *24*, p. 918-919.
13. M. Garey et D. Johnşson, *Computer and intractability*; W. Freeman and Co. N.Y., 1979.
14. R. Garfinkel et G. L. Nemhauser, Integer programming; J. Wiley and sons, N.Y., 1972.
15. A. M. Geoffrion, Lagrangean relaxation for integer programming; Math Programming Study 2, 1974, p. 82-114.
16. F. Glover, Generalized cuts in Diophantine programming, Management Sciences 13, 1966-1967, p. 254-268.

17. S. GODANO, Méthodes géométriques pour la programmation linéaire, Thèse Université Blaise Pascal, Clermont-Ferrand, 1994.
18. R. E. GOMORY, Outlines of an algorithm for integer solutions to linear programs, Bull American Math. Soc., 1958, *64*, p. 275-278.
19. R. E. GOMORY, An algorithm for integer solutions to linear programs, Recent Advances in Math. programming. (R. L. GRAVES and P. WOLFE Eds.), Mac Graw Hill, N.Y., 1963, p. 269-302.
20. M. GONDRAN, Un outil pour la programmation en nombres entiers, la méthode des congruences décroissantes : RAIRO 3, 1973, p. 35-54.
21. M. GROTSCHEL, L. LOVACZ et A. SCHRIJVER, The ellipsoid method and combinatorial optimization, Springer-Verlag, Heidelberg, 1986.
22. M. HELD et R. KARP, The travelling salesman problem and minimum spanning trees, Operat. Research 18, 1970, p. 1138-1162.
23. A. HOFFMAN et J. KRUSKAL, Integral boundary points of integer polyedra in Linear Inequalities and Related Systems, H. KUHN and A. TUCKER Eds., Princeton Univ. Press, 1986, p. 223-246.
24. R. KANNAN et A. BACHEM, Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix; SIAM Journ. Comput 8, 1979, *4*, p. 499-507.
25. H. LANGMAACK, Algorithm 263 Gomory 1 [H]; Communications of the ACM, 1965, *8*, p. 601-602.
26. H. LENSTRA, Integer Programming with a fixed number of variables, Maths of Operat. Research, 1983, *8*, p. 538-548.
27. L. G. PROLL, Certification of algorithm 263 A [H] Gomory 1, Comm. ACM 13, 1970, p. 326-327.
28. I. ROSEMBERG, On Chvatal's cutting planes in integer programming, Mathematische Operation Forschung und Statistik, 1975, *6*, p. 511-522.
29. A. SCHRIJVER, Theory of linear and integer programming, Wiley, Chichester, 1986.
30. A. V. SRINIVASAN, An investigation of some computational aspects of integer programming, JACM 12, 1965, p. 525-535.