PUBLICATIONS MATHÉMATIQUES ET INFORMATIQUES DE RENNES

JEAN-FRANÇOIS MAURRAS

Complexité algébrique

Publications de l'Institut de recherche mathématiques de Rennes, 1985, fascicule 4
« Séminaires de mathématiques - science, histoire et société », , p. 96-122

<http://www.numdam.org/item?id=PSMIR_1985___4_96_0>

Jean-François MAURRAS

COMPLEXITE ALGEBRIQUE

Résumé

La notion de nombre dans les données d'un problème combinatoire est reconsidérée ; des nombres "algébriques" sont introduits. Des généralisations polynomiales, d'algorithmes polynomiaux sont alors décrites.

Abstract

The notion of numbre in the data of a combinatorial problem is revisited ; "algebric" numbers are introduced. Polynomial generalizations of polynomial algorithms are then described.

Code AMS : 68 A 20

Key Words

Programming - Algorithms - Computational Complexity.

0 - INTRODUCTION

Nous nous proposons dans cet article d'élargir les types de données acceptables par certains algorithmes polynomiaux résolvant certains problèmes (combinatoires). Ceux qui nous intéresserons ici auront des "instances", c'est-à-dire des couples problème, ensemble de données précisé, finies ; ces instances seront représentées au moyen d'un nombre fini de signes d'un alphabet (fini) donné.

A chaque instance I du problème P, on associe un entier $\mu(I)$ au couple formé par I et l'algorithme A qui résout le problème P on associe l'entier $\nu_A(I)$. On dit que l'algorithme A est <u>polynomial</u> s'il existe un polynome P tel que quel que soit l'instance I du problème P, $\nu_A(I) < P(\mu(I))$.

C'est à dessein que dans la définition précédente nous n'avons pas précisé les choix de μ et ν. Il y a des choix conventionnels : pour μ c'est le nombre de signes nécéssaires à la description de I ; dans nombre de domaines cependant, pratiques ou non, le choix est différent (par exemple dans les problèmes de tri $\mu(I)$ peut être le nombre d'objets à trier, dans les problèmes matriciels $\mu(I)$ peut être le nombre de lignes ou de colonne d'une matrice et dans des problèmes de graphes $\mu(I)$ peut être le nombre de sommet du graphe).

En ce qui concerne ν, on considère généralement A comme une machine de Turing et $\nu_A(I)$ compte généralement le nombre de pas de A dans la résolution du problème P associé à l'instance I. Comme pour μ d'autres choix sont possibles (par exemple le nombre d'opérations algébriques élémentaires, de comparaisons, des deux, etc...).

Dans la suite de cet article, sauf référence explicite, les sens donnés à μ et à ν seront ceux usuels. Essayons de préciser, sur un exemple, dans cette introduction, les problèmes qui peuvent être soulevés par la définition de $\mu(I)$.

Supposons que I soit un nombre entier n, le nombre de signes pour nécessaires pour le décrire varie linéairement avec $\lfloor \text{Max}(\text{Log}(n),0) \rfloor$, où $\lfloor i \rfloor$ représente la partie entière par défaut de i, où l'on admet que $\text{Max}(\text{Log}(0), 0) = 0$, et où la base des logarithmes n'est pas précisée (le changement de base étant linéaire). Dorénavant, pour alléger l'écriture, avec objections précédentes près, on dira que $\mu(n) = \text{Log}(n)$.

Si $n = p!$ le nombre de signes nécessaires pour représenter n est clairement $\text{Log}(p) + 1$ ou encore $\text{Log}(p)$ en ne retenant que le terme d'ordre le plus élevé. On représente avec le même nombre de signes $n = 2^p - 1$, n est un nombre de Mersenne si p est premier. On sait [16] [18] que si l'hypothèse de Riemann généralisée est vraie, il existe un algorithme polynomial permettant de décider si un entier n (quelconque) est premier et $\nu(n,A) < \text{Log}(n)^5$. Si n est un nombre de Mersenne le test de Lucas-Lehmer suivant : n premier $\Leftrightarrow L_{p-2} = 0$, avec $p \neq 2$, $L_o = 4$, $L_{n+1} \equiv L_n^2 - 2 \mod (n)$, permet de décider en $\nu_A(n) = \text{Log}^3(n)$ $(\text{Log}(\text{Log}(n))\text{Log}^2(n)$, en s'y prenant un peu mieux) calculs si n est premier, mais $\mu(n) = \text{Log}(p) = \text{Log}(\text{Log}(n))$, et aucun de ces deux algorithmes n'est polynomial. Ce problème est peut-être de nature fondamentalement non polynomiale.

Terminons cette longue introduction en définissant la notion d'équivalence polynomiale.

Soit $\ell(I)$ et $\ell'(I)$ deux fonctions des instances de P à valeurs dans N ; on dira que ℓ et ℓ' sont <u>polynomialement équivalentes</u> s'il existe deux polynomes P et P' tels que pour toute instance I de P, $\ell'(I) \leqslant P(\ell(I))$ et $\ell(I) \leqslant P'(\ell'(I))$. Notons que c'est parce que $\text{Log}_a(n)$ et $\text{Log}_b(n)$ sont polynomialement équivalents que nous ne précisons pas la base des logarithmes utilisés.

I - DES TYPES DE DONNEES NUMERIQUES

Habituellement on ne considère comme données numériques que des entiers représentés "naturellement". Nous allons admettre d'autres données : des données "algèbriques". Dans une instance I donnée on a un nombre fini de tels nombres, soit $B = \{b_1, b_2, \ldots, b_p\}$ leur ensemble. Un élément b_i de B est décrit au moyen d'un nombre fini de signes (par exemple b_i peut être une racine réelle spécifiée d'un polynome à coefficient entiers donné, le nombre π, le nombre e...).

Dans l'expression de la solution les nombres appartiennent à un ensemble $B^* = \{i \in I , f_i(b_1, b_2, \ldots, b_p)\}$ où pour $i \in I$, f_i se décrit polynomialement en fonction de B. Nous considérerons dans les exemples qui vont suivre des f_i de la forme $\sum_{j=1}^{p} \lambda_j b_j$ avec $\lambda_j \in Z$ et où Log $|\lambda_j|$est polynomial en $\mu(I)$, et plus généralement des f_j fractions rationnelles à coefficients dans Z d'éléments de B.

Nous aurons besoin aussi de comparer les éléments de B^*, et pour que l'algorithme total soit polynomial ces comparaisons devront se faire polynomialement.

Dans les trois sous-sections suivantes, nous allons donner les trois exemples d'ensembles B que nous allons considérer, et les algorithmes permettant de comparer les éléments de B^* (essentiellement dans le cas où $f_i \in B^*$ est de la forme $\sum_{j=1}^{p} \lambda_i b_j$).

I.1 - B est une liste de polynomes à une indéterminée à coefficients entiers totalement ordonnés par le terme de plus bas degré (Type 1).

On a un algorithme immédiat pour comaprer les éléments de B^*, et cet algorithme est polynomial. Il en serait de même avec des fractions rationnelles.

I.2 - B est une liste de nombres de la forme $b_i = \sum_{j=1}^{n} c_j k^{d_j}$, avec $k, d_j \in N$, $c_j \in Z$ (Type 2)

Remarquons que les $f_i \in B$, de la forme $\sum_{i=1}^{p} \lambda_i b_i$, ont la même forme que les éléments de B. Comparer f_i et f_j revient donc à décider si un nombre du type b_i est < 0, $= 0$, ou > 0.

Algorithme A_1.

Données : le nombre $b = \sum_{j=1}^{n} c_j k^{d_j}$, $\mu(b) = \sum_{j=1}^{n} (\text{Log}|c_j| + \text{Log } d_j) + n$

Résultat : $b < 0$, ou $b = 0$, ou $b > 0$

Complexité : $0(n^2 \text{ Max } (\text{Log}(d_j, \text{Log}|c_j|))) < (\mu(b))^3$

Procédure :

0. Ordonner les $c_j k^{d_j}$ par ordre de d_j croissant aller en 1.

1. Si $j \leqslant n$ tel que $c_j = 0$ supprimer le monome correspondant et renuméroter les monomes . Poser $n = n-1$, si $n = 0$ poser $b = 0$. Stop.

2. Si $\forall\, j < n$, $d_{j+1} \geqslant d_j + \lfloor \text{Log}_k|c_j| \rfloor + 1$ aller en 4.

3. Remplacer c_j par $c_j + c_{j+1} k^{(d_{j+1}-d_j)}$, puis c_{j+1} par 0 et aller en 1.

4. Si $c_n < 0$ poser $b < 0$. Stop.

5. Poser $b > 0$. Stop.

Preuve de la correction de cette procédure

1. Le nombre obtenu à la fin de l'étape 3 est clairement b.

2. Si 2 est vérifié on a $\left|\sum_{j=1}^{\ell} c_j k^{d_j}\right| < \left|\sum_{j=1}^{\ell} c_j k^{d_j}\right| < d^{\ell+1}$, et donc b est du signe de c_n.

3. Si la procédure s'arrête sur 1 on a bien entendu b = 0.

Complexité :

1. Etape 0 : n log n Max $(\mathrm{Log}(d_j))$
2. Etape 1 : n (renumérotation) effectués au plus n fois : n^2 .
3. Etape 2 : n addtions et comparaisons à 0 effectuées au plus n fois : n^2 Max $(\mathrm{Log}(d_j)$, $\mathrm{Log}|c_j|)$.
4. Etape 4 : au plus n fois une addition : n Max $(\mathrm{Log}|c_j| + \mathrm{Log}|c_{i+1}|)$.

Remarque : Une implementation attentive de la procédure proposée doit conduire à une complexité linéaire en $\mu(b)$.

I.3 - B est une liste de racines réelles spécifiées de polynomes à coefficients entiers.

Soit $\forall\, i \in I$, $|I| < +\infty$, $P_i = a_o^i + a_1^i x + \ldots + a_{n_i}^i x^{n_i}$ et $k_i \in N$.

$B = \{i \in I\ ,\ b_i\}$, b_i étant la $k^{\text{ième}}$ racine réelle, dans l'ordre de la droite réelle de P_i. Il est clair que tout couple (k_i, P_i) ne définit pas un réel. Sans restreindre la généralité on peut supposer que P_i n'a que des racines simples, sinon on remplace "polynomialement", P_1 par $Q_1 = P_1 / \mathrm{PGCD}(P_i\ P_i')$, où P_i' est le polynome dérivé de P_i.

Soit donc $P = a_o + a_1 x + \ldots + a_p x^p$ un polynome dont toutes les racines sont simples, et un entier k. Nous montrerons que l'on peut séparer les racines réelles de P au moyen d'un algorithme polynomial, et trouver deux rationnels α et β tels que le réel b défini par le couple (k,P) soit la seule racine réelle de P comprise entre α et β, et ce au moyen d'un algorithme polynomial. α et β qui seront obtenus par dichotomie seront de la forme $\frac{r}{2^t}$, $r \in Z$, $t \in N$.

Soit $Q = c_o + c_1 x + \ldots + c_q x^q$, $\ell \in M$ et soit d le réel défini par le couple (ℓ, Q) de plus d est la seule racine réelle de Q entre γ et δ.

Nous allons décrire un algorithme pour décider si $z = \lambda b + \mu d$, avec $\lambda, \mu \in Z$, est < 0 , $= 0$, > 0.

Algorithme A_2

Données : P , α , β ; Q , γ , δ , λ , μ .

Résultat : $z < 0$, ou $z = 0$, ou $z > 0$.

Complexité : 0 (p x q x Log m) calculs des valeurs de P et Q.

Procédure :

0. Calculer par dichotomie des encadrements α', β' de b et γ', δ' de d avec $\beta' - \alpha' \leq 1/2\ M$, $\delta' - \gamma' < 1/2\ M$, avec $M = 8m^{(p+1)(q+1)} x\ 4^{pq}$ et $m = \text{Max}\ (|a_i|, |c_i|, |\lambda|, |\mu|)$.

1. Calculer $z' = \lambda\alpha' + \mu\gamma'$

3. Si $z' < -\frac{4m}{M}$ alors $z < 0$. Stop.

4. Sinon si $-\frac{4m}{M} < z' < \frac{4m}{M}$ alors $z = 0$. Stop.

5. Sinon $z > 0$.

Preuve de la correction de cette procédure.

Remarquons tout d'abord que les racines réelles non nulles du polynome $P = a_o + a_1 x + \ldots + a_p x^p$ ont une valeur absolue supérieure à 1/2 m' avec $m' = \text{Max}\ |a_i|$, on vérifie que $\forall\ x < 1/2\ m'$, $P(x) \neq 0$. De même $\forall\ x > 2\ m'$, $P(x) \neq 0$.

Il nous suffit donc de montrer que les coefficients d'un polynome à coefficients entiers, annullant z, sont, en valeur absolue inférieur à M/8m.

Pour cela rappelons la construction classique d'un tel polynome, il s'obtient comme déterminant de la matrice obtenue en décomposant les p x q produits $b^i d^i(\lambda b + \mu d)$ avec $0 < i < p$, $0 < j < q$ sur le système de générateurs (base en général) formé des pq $b^i d^j$ (et en se servant de P et de Q pour exprimer b^p et d^q). Il est commode pour décrire cette matrice d'ordonner les $b^i d^j$ suivant l'ordre lexicographique ($b^i d^j < b^{i'} d^{j'} \Leftrightarrow jp+i < j'p+i'$), de considérer les matrices B, annulant $z = \lambda b$, et D annulant $z = \mu d$.

On a pour $1 \leqslant i \leqslant p-1$, $B_{ii} = -z$, $B_{ii+1} = \lambda$, $B_{pi} = -\lambda a_i/a_p$ et $B_{pp} = - x - \lambda a_o / a_p$, les autres termes sont nuls.

De même pour $1 \leqslant j \leqslant q-1$, $D_{jj} = -z$, $D_{jj+1} = \mu$, $D_{qj} = -\mu b_i/b_q$, et $D_{qq} = - x - \lambda b_o / b_q$, les autres termes sont nuls.

Soit I la matrice unité de taille p x p, la matrice cherchée s'obtient à partir de D en remplaçant :

pour $1 < j < q-1$ D_{jj} par B , D_{jj+1} par μI , D_{qj} par $\mu b_i/b_q I$, et D_{qq} par $B - \mu b_o / b_q I$, les autres termes sont nuls.

On vérifie que cette matrice a au plus quatre éléments non nuls différents de z par colonne, termes inférieurs ou égaux à m, et que le plus grand monome en a_i b_j μ λ est de degré (après multiplication par les dénominateurs a_p et b_q lorsque c'est nécessaire) (p+1) (q+1) - 1, les différents coefficients de z sont donc bien inférieurs à $m^{(p+1)(q+1)-1} \times 4^{pq}$, ce qui justifie la procédure.

Complexité

On a à effectuer 2 Log_2 M calcul de la valeur des polynomes P et Q pour des nombres rationnels dont le numérateur et dénominateur sont respectivement inférieurs à 2m M et M. On sait par ailleurs que le calcul de la valeur d'un polynome P dont on ne sait pas à priori si certains termes sont nuls est polynomial en $m = \underset{i=0,\ldots,n}{\text{Max}} (a_i)$ et n son degré. De plus on doit

calculer $z' = \lambda\alpha' + \mu\gamma'$ calcul dont la complexité est inférieure à celle du calcul de la valeur du polynome P pour $x = \alpha'$. D'où la complexité annoncée.

Plus généralement si z est combinaison linéaire de racines spécifiées des polynomes $P_1, P_2, \ldots, P_\ell$ de degrés $p_1, p_2, \ldots, p_\ell$, et dont les coefficients sont $\lambda_1, \lambda_2, \ldots, \lambda_\ell$, on aura un algorithme analogue A_3 en posant $m = \underset{\substack{i=1,\ldots,\ell \\ ij=0,\ldots,p_i}}{\text{Max}} (|a_{ij}|, \lambda_i)$, $p = \underset{i=1,\ldots,\ell}{\Pi} (p_i+1)$, et $M = (m \times 2\ell)^p$. Cet algorithme sera polynomial à partir du moment où $p = \prod_{i=1}^{\ell} (p_i+1)$ sera polynomial. Des données ayant cette propriété seront dites du type 3 . Soit $b = (k,P)$ et $B = \{1, b, b^2, \ldots, b^{p-1}\}$ il est clair que l'on a un algorithme polynomial pour trouver le signe de combinaisons linéaires à coefficient entiers de b_i, ces données sont dites du type 4.

Séparation des racines réelles du polynome P.

La construction décrite précédemment permet, bien entendu, de calculer le polynome annulant les différences entre les racines de P, et donc de borner inférieurement la plus petite différence non nulle par 1/2 M avec $M = 8(4\ m)^{(p+1)^2}$ ou $m = \text{Max}(|a_i|)$. On peut dans ce cas abaisser sensiblement la valeur de M en remarquant que $\lambda = \mu = 1$. On a alors $M = 8 \times m^{2p} \times 4^{p^2}$.

Pour séparer les racines de P, dont par hypothèse les racines sont simples, on peut supposer que le polynome dérivé P', a ses racines séparées, et à partir d'un encadrement de ses racines, séparer celle de P. Ceci conduit à un algorithme polynomial que nous ne décrirons pas ici. En revanche nous décrirons un algorithme très simple que l'on déduit du théorème de Sturm [19] :

Soit P *un polynome, soient* $P_1, P_2, \ldots, P_r$ *des polynomes de degrés décroissant obtenus comme il suit :* $P_1 = P'$ *le polynome dérivé,* $P_2, \ldots, P_r$ *sont les restes dans l'algorithme d'Euclide :* $P_{i-1} = Q_i P_i - P_{i+1}$ et $P_{r-1} = Q_r P_r$.

Pour chaque nombre réel a *qui n'est pas une racine de* P(x), *soit* w(a) *le nombre de variation de signe dans la suite suivante (on ne compte pas les zéros comme variations) :*

$$P(a)\ ,\ P_1(a)\ ,\ldots,\ P_r(a)\ .$$

Soient b < c *deux réels qui ne sont pas racines de* P(x), *alors le nombre de racines distinctes de* P(x) *entre* b *et* c *(les racines multiples n'étant comptées qu'une fois), est égal à :* w(b) - w(c).

Il est clair que le calcul de $P_1,P_2,\ldots,P_r$ à partir de P est polynomial. On sait que toutes les racines réelles de P sont comprises entre -2m et 2m avec m = Max ($|a_i|$). En posant b = -2m , c = 2m, par dichotomie on peut séparer la racine la plus petite, puis successivement toutes les autres. On sait de plus que si les calculs sont menées avec une précision égale à 1/2 M avec $M = 8 \times m^{2p} \times 4^{p^2}$, deux racines distinctes sont nécessairement séparées.

Ceci donne un sens au nombre réel défini au début de cette section par le couple (k,P).

Remarquons qu'on peut faire un développement analogue en ce qui concerne les z = b x d avec b défini par (k,P), d par (ℓ,Q).

Remarquons aussi que l'on peut définir notre ensemble de réels B comme l'ensemble des combinaisons linéaires à coefficients dans Z des p_i premières puissances des racines réelles spécifiées des polynomes $P_1,P_2,\ldots,P_k$. Si $p = \prod_{i=1}^{k} (P_i+1)$ est polynomial (en la longueur des données du problème dans lequel ces nombres interviennent) alors l'algorithme A_3 le sera aussi.

Remarquons que si de plus on spécifie une racine pour chaque polynome P_i, donc un couple (k_i,P_i), alors tout produit de ces nombres pourra s'exprimer comme combinaison linéaire de monomes dans lesquels le degré de $(k_i,P_i) = b_i$ est inférieur à p_i = degré (P_i) ; le nombre de ces monomes est inférieur

à $p = \prod_{i=1}^{k} P_i$, et est donc polynomial.

Remarquons enfin que l'on pourrait définir d'autres nombres réels, au moyen d'un polynome P, par exemple la partie réelle (ou imaginaire) d'une recine complexe de P, on peut encadrer un tel réel, il suffit pour cela de calculer un polynome l'annulant : par exemple en éliminant y entre :

$$P(x+iy) = 0 \qquad \text{et} \qquad P(x-iy) = 0.$$

Ce qui peut se faire polynomialement en calculant un résultant [20] qui est ici le déterminant d'une matrice (3p+2) x (3p+2) dont les termes sont des polynomes de degré au plus 2p. Ce déterminant peut se calculer polynomialement [7]. Le polynome obtenu est de degré au plus 2p x (3p+2) et donc polynomial en les données du polynome P.

Notons ici les problèmes non résolus de cette première partie :

1. Existe-t-il un algorithme polynomial pour décider du signe (-,0,+) de l'expression :

$$x^d + y^d - z^d \quad , \quad x,y,z,d \in N \ ?$$

2. Existe-t-il un algorithme polynomial pour décider du signe de $\sum_{i \in I} \lambda_i b_i$ avec

$$b_i = (k_i, P_i) \quad , \quad (k_i^{\text{ième}} \text{ racine réelle de } P_i(x) = 0) \ ?$$

3. Le problème de 1 est-il N P , complet ?

4. Le problème de 2 est-il N P , complet ?

II - LES PROBLEMES, LES ALGORITHMES ET LES DONNEES QU'ILS ACCEPTENT

Dans cette section nous ferons référence à la terminologie de la théorie des graphes que nous ne rappellerons pas, le lecteur pourra la trouver dans [2] .

II.1 - Base de poids maximum dans un matroïde.

Soit M = (E,F) un matroïde fini [2] , soit ∀ e C E, c_e appartenant à un ensemble totalement ordonné, l'algorithme B_1 suivant permet de résoudre le problème de la partie libre ($F \in F$) de M, de poids maximum ($\sum_{e \in F} c_e$)

B_1

1. Classer les c_e dans l'ordre décroissant
2. $F_o = \emptyset$, si $c_1 < 0$ stop, sinon, poser i = 1 .
3. Si $\{i\} \notin E$ stop, sinon poser $F'_i = F_{i-1} \cup \{i\}$.
4. Si $F'_i \notin F$ et si $c_{i+1} > 0$ poser $i \leftarrow i+1$ et aller en 3 .
5. Si $F'_i \notin F$ et si $c_{i+1} \leqslant 0$, Stop.
6. Si $F'_i \in F$ poser $F_i = F'_i$, $i \leftarrow i+1$ et aller en 3.

On trouvera une preuve de cet algorithme dans [2] .

Cet algorithme est polynomial, à partir du moment où les étapes 1 4,6 le sont. Dans 1 on a à comparer un nombre polynomial de fois ($n^2/2$ ou mieux n log n) les éléments de c_e, qui peuvent donc indifféremment être des éléments des types 1 et 2, et aussi des racines spécifiées de polynomes à coefficients entiers.

II.2 - Le plus court chemin dans les graphes sans circuits absorbants.

Soit G = (X,U) un graphe orienté [2] et $d \in R^U$. De plus ∀ γ circuit de G [2], $\sum_{u \in \gamma} d_u \geqslant 0$.

L'algorithme suivant B_2 [4] [5], résout le problème du plus court chemin entre a et b $\in$ X.

B_2

0. $S_o = \{a\}$, $D_o^a = 0$, $\forall\, i \neq a$, $D_o^i = \infty$; poser i = 0 et aller en 1.

1. Soit u=(x,y) avec $x \in S_t$, $y \notin S_t$ et $\Delta y = \underset{\substack{x \in S_t \\ y \notin S_t}}{\text{Min}} (D_t^x + d_u)$

2. $S_{t+1} = S_t \cup \{y\}$, $\forall\, x \in S_t$, $D_{t+1}^x = D_t^x$, $D_{t+1}^y = \Delta y$, si $X \setminus S_{t+1} = \emptyset$ Stop.

3. Sinon poser t $\leftarrow$ t+1, aller en 1.

Cet algorithme est polynomial à partir du moment où l'on peut effectuer l'étape 1 polynomialement. C'est le cas pour des d des types 1, 2, 3 et 4.

II.3 - Flot maximum dans un (multi) graphe fini.

Soit G = (X,U) un multigraphe orienté, S(X,U) sa matrice d'incidence sommets arcs [2], $\forall\, u \in U$, $\Gamma_u = [a,b]$, a,b $\in$ R , soit g $\in$ U.

Le problème du flot maximum est :

$$\begin{cases} \text{Max } \phi g \\ S\varphi = 0 \\ \forall\, u \in U \ , \ \phi_u \in \Gamma_u \end{cases}$$

On sait qu'un flot peut toujours être représenté comme combinaison linéaire de cycles élémentaires (à un cycle élémentaire γ on associe un flot ϕ_γ en orientant γ comme l'un de ses arcs, puis en posant $\phi_{\gamma e} = 1$ si $e \in \gamma$, e orienté dans le même sens que γ , $\phi_{\gamma e} = 1$ si $e \in \gamma$, orienté en sens contraire de γ, $\phi_{\gamma e} = 0$ si $e \notin \gamma$).

Edmonds et Karp [8] proposent une modification B_3 de l'algorithme de Ford et Fulkenson [9] qui permet de résoudre polynomialement ce problème lorsque a et b $\in$ Z.

On suppose que l'on connait ϕ_o tel que $\forall\, u \in U$, $\phi_{ou} \in \Gamma_u$, sinon $|U|$ applications de l'algorithme B_3 permettent de trouver ϕ_o ou de prouver qu'un tel ϕ_o n'existe pas.

B_3

0. ϕ_o

1. Chercher le cycle γ contenant g telque $\forall\, u \in U$, $\phi_{tu} + \phi_{\gamma u} \in \Gamma_u$, ayant le plus petit nombre d'arcs.

2. Si un tel cycle n'existe pas Stop.

3. Sinon soit $\lambda = \operatorname{Min} \left(\operatorname{Min}_{\phi_{\gamma u}=1} (b_u - \phi_{tu}) , \operatorname{Min}_{\phi_{\gamma u}=-1} (\phi_{tu} - a_u) \right)$

   poser $\phi_{t+1} = \phi_t + \lambda\, \phi_\gamma$, $t \leftarrow t+1$, aller en 1.

E et K démontrent que le nombre de retour en 1 d'un tel algorithme est borné par $1/2\ (|X| \times |U|)$, de plus l'étape 1 s'effectue polynomialement au moyen de B_2 et de comparaisons entre ϕ_{tu} et a_u ou b_u. Par construction de φ_t on remarque que le flot optimal est combinaison linéaire de a_u et b_u avec des coefficients qui ne peuvent pas dépasser $2^{(|X| \times |U|)/2}$ et donc les données des types 1, 2, 3, 4 peuvent être utilisées dans la définition des a_u et b_u.

III - LE FLOT A COUT MAXIMUM

Soit $G = (X,U)$, $\forall\, u \in U$, $\Gamma_u = [a_u, b_u]$, $a_u, b_u \in Z$, $c_u \in Z$

Le Problème du flot à coût maximum est :

$$\begin{cases} \max \sum_{u \in U} c_u\, \phi_u \\ S\phi = 0 \\ \forall\, u \in U \ , \ \phi_u \in \Gamma_u \end{cases}$$

Minty [17] et Fulkerson [10] proposent de trouver un couple $(\hat{\phi},\hat{\theta})$ où est une tension, (i.e. $\forall\ \phi$, $S = 0$, $<\theta,\phi> = 0$), tel que $\forall\ u \in U$, $(\phi_u,\theta_u) \in C_u$ une courbe définie dans le plan au moyen de ses segments $[(a_u,+\infty),(a_u,c_u)]$, $[(a_u,c_u),(b_u,c_u)]$, $[(b_u,c_u),(b_u,-\infty)]$. Le fait pour $(\hat{\phi}_u,\hat{\theta}_u)$ d'appartenir à l'un des trois segments précédents partitionne l'ensemble des arcs en 3, U_1, U_2, U_3 et :

$$\forall\ \phi\ ,\quad c\phi - c\hat{\phi} = c(\phi-\hat{\phi}) = (c-\theta)\ (\phi-\hat{\phi})$$

$$= \underset{u \in U_1}{\sum} \underset{\leq 0}{(c_u-\theta_u)} \underset{\geq 0}{(\phi_u-\hat{\phi}_u)} + \underset{u \in U_2}{\sum} \underset{=0}{(c_u-\theta_u)(\phi_u-\hat{\phi}_u)} + \underset{u \in U_3}{\sum} \underset{\geq 0}{(c_u-\theta_u)} \underset{\leq 0}{(\phi_u-\hat{\phi}_u)}$$

et finalement $c\phi \leq c\hat{\phi}$.

L'algorithme proposé par Minty et Fulkerson consiste pour un arc u tel que $(\phi_u,\theta_u) \notin C_u$ et tel qu'il existe $\lambda > 0$ tel que $(\varphi_n+\lambda_1\ ,\theta_u) \in C_u$, à à trouver ou bien un cycle dans lequel le flot puisse augmenter, ou bien un cocycle dans lequel la tension puisse augmenter. L'existence d'un cocycle dans lequel la tension puisse augmenter indéfiniment correspond au cas d'arrêt de l'algorithme B_3 et prouve que le flot ϕ_u obtenu dans l'arc u est maximum.

Edmonds et Karp [8] proposent une modification de cet algorithme qui permet de résoudre ce problème polynomialement.

Remarquons que si seul c a ses éléments dans un des quatre types de nombres de la section 1, l'algorithme proposé par Edmonds et Karp reste polynomial. En effet dans cet algorithme le nombre d'augmentations de θ est borné par $|X| \times |U| \times \text{Log}_2 \underset{u \in U}{\text{Max}} (|a_u|,|b_u|)$ et on a vu que l'on sait comparer polynomialement les combinaisons linéaires, à coefficients ayant un nombre de digits polynomiaux, pour les quatre types de données.

Revenons à l'algorithme d'Edmonds et Karp.

Ils proposent d'arrondir suivant les puissances d'un entier k (2 dans leur papier) les valeurs de a_u et b_u, a_u par défaut, b_u par excès. Ils

résolvent alors une suite de $\sum_{u \in U} \text{Log}_2 \text{ Max } (|a_u|,|b_u|)$ problèmes de flot à coût maximum.

Soit a_u^ℓ la partie entière par défaut de $a_u/2^\ell$, b_u^ℓ la partie entière par excès de $b_u/2^\ell$. Soit P^ℓ le problème de flot à coût minimum correspondant.

Remarque :

Si le problème $P = P^\circ$ a une solution, alors $\forall\ \ell$, P^ℓ a une solution, de plus on déduit de la solution optimale de P^ℓ une solution optimale de P'^ℓ, problème que l'on construit à partir de P^ℓ en remplaçant a_u^ℓ par $a'^\ell_u = 2a_u^\ell$, $b'^\ell_u = 2b_u^\ell$, et ce en remplaçant ϕ_u^ℓ par $\phi'^\ell_u = 2\phi_u^\ell$. Clairement
$(\phi_u^\ell,\theta_u^\ell) \in C_u^\ell \Rightarrow (2\phi_u^\ell,\theta_u^\ell) \in C'^\ell_u$.

Pour résoudre $P^{\ell-1}$ on va résoudre, au moyen de l'algorithme F_1, à partir de P'^ℓ, une suite d'au plus $|U|$ problèmes P'^ℓ_I avec $I \subset U$.

F_1

0. $I = \{ j \in U , a_j^{\ell-1} \leq \phi'^\ell_j \leq b_j^{\ell-1}\}$ si $I = U$ stop. Sinon $\forall\ j \in I$ remplacer $2a_j^\ell$ par $a_j^{\ell-1}$, $2b_j^\ell$ par $b_j^{\ell-1}$, soit $j \in U \setminus I$ aller en 1.

1. Si $j \notin [(2a_j^\ell,c_j),(2a_j^\ell,+\infty)]$ aller en 5 .

2. Chercher un cycle contenant j dans lequel le flot puisse augmenter. Si un tel cycle existe augmenter le flot d'une unité et retourner en 0.

3. Sinon on a un cocycle dans lequel la tension peut augmenter. Si la tension ne peut pas augmenter indéfiniment, l'augmenter de la valeur maximum et aller en 2.

4. Stop le flot ne peut pas satisfaire $\forall\ u \in U$, $\phi_u \in \Gamma_u$.

5. $j = (x,y)$. Poser $j' = (y,x)$, $\phi'^\ell_{j'} = -\phi'^\ell_j$, $a_{j'}^{\ell-1} = -b_j^{\ell-1}$, $b_{j'}^{\ell-1} = -a_j^{\ell-1}$
Supprimer j, noter que l'arc j a été remplacé par j' aller en 2.

Comme on ne peut trouver plus de $|X|$ cocycles consécutifs et que chaque fois que l'on trouve un cycle $|I|$ augmente au moins de 1 le nombre de recherches de cycles est borné par $|X| \times |U|$.

Remarque

A l'étape initiale, on part du flot nul et de la tension nulle, et on peut avoir, au point 2 de F_1 à diminuer le flot dans l'arc j. L'algorithme se déroule de façon similaire.

III 1

Soient à présent, $\forall\, u \in U$, c_u, a_u, b_u des polynomes à une indéterminée sur Z.

Remarque

Soient $\forall\, u \in U$, $(\phi_u,\theta_u) \in C_u$, on a $a_u \leqslant \phi_u \leqslant b_u$ où a_u, b_u, ϕ_u sont des polynomes à une indéterminée de degré donné n. Soit $p \leqslant n$, et soit a_u^p, b_u^p, ϕ_n^p, les polynomes déduits de a_u, b_u, ϕ_u en supprimant les termes de degré $> p$. Alors, par définition de la relation d'ordre entre les polynomes, on a $a_u^p \leqslant \phi_u^p \leqslant b_u^p$. Soit C_u^p la courbe déduite de C, en remplaçant a_u par a_u^p b_u par b_u^p, alors $(\phi_u,\theta_u) \in C_u \Rightarrow (\phi_u^p,\theta_u^p) \in C_u^p$.

En effet $\phi_u = a_u \Rightarrow \phi_u^p = a_u^p$ et $a_u < \phi_u \Rightarrow a_u^p \leqslant \phi_u^p$.

En conséquence : Si pour p=0,1,...,n la solution des différents problèmes est unique, on pourra successivement résoudre les problèmes avec $p = 0$, $p = 1,\ldots,$ $p = n$, sans remettre en cause à l'étape p la valeur de p trouvée à l'étape p-1.

En conséquence : On peut remplacer le problème P par le problème P^q où a_u, ϕ_u, b_u, c_u sont des polynomes du sous anneau $Z[x^q]$ où x est remplacé par x^q avec $q = |U|$, et $\forall\, j \in U$, c_j remplacé par $c_j + x^{nq+j}$. Après cette

transformation la solution des différents problèmes P^{qp} est unique (dans P^{qp} on supprime pour a,b, et ϕ les termes de degré > qp). De plus à la solution optimale de P^q correspond une solution optimale de P, et celle des différents P^{qp}.

En effet soit $\forall\ j \in U, (\hat{\phi}_j, \hat{\theta}_j) \in C_j^q$ alors la relation d'ordre entre les polynomes $(\hat{\phi}_{jp}, \hat{\theta}_j) \in C_j^{qp}$ où $\hat{\phi}_{jp}$ et C_j^{qp} s'obtiennent à partir de $\hat{\phi}_j$, a_j, b_j en supprimant les termes de degré > qp. On a aussi pour les mêmes raison (sur θ) $(\hat{\phi}_j, \bar{\theta}_j) \in C_j$ où $\hat{\phi}_j$ s'obtient à partir de $\hat{\phi}_{jp}$ en remplaçant x^q par x et $\bar{\theta}_j$ à partir de $\hat{\theta}_j$ en supprimant tout d'abord les termes de degré > nq puis en remplaçant x^q par x.

L'unicité s'obtient en considérant deux solutions optimales $\hat{\phi}$ et $\bar{\phi}$ et en considérant le terme de plus petit degré r par lequel elles diffèrent et l'arc de plus petit indice j ayant cette propriété. Une comparaison des coûts montre qu'elles ne peuvent pas être optimales simultanément $(x^{nq+j}\ \hat{\phi}_j\ x^r = x^{nq+j}\ \bar{\phi}_j\ x^r)$.

<u>En conséquence</u> : On résoudra P en résolvant tout d'abord P^{q_o} puis on redéfini les intervalles de définition de ϕ_1 comme suit :

$$\phi_{u_o} = a_{u_o} = b_{u_o} \Rightarrow \phi_{u_1} \in [a_{u_1}, b_{u_1}]$$

$$\phi_{u_o} = a_{u_o} \neq b_{u_o} \Rightarrow \phi_{u_1} \in [a_{u_1}, +\infty]$$

$$\phi_{u_o} = b_{u_o} \neq a_{u_o} \Rightarrow \phi_{u_1} \in [-\infty, b_{u_1}]$$

$$a_{u_o} < \phi_{u_o} < \phi_{u_1} \Rightarrow \phi_{u_1} \in [-\infty, +\infty]$$

et ainsi de suite, les différents problèmes étant séparés, ceci constitue l'algorithme F_2.

<u>Remarque</u> : l'algorithme déduit reste polynomial.

III 2

Supposons à présent que c_u, a_u et b_u sont des expressions de la forme suivante :

$$c_u = \sum_{i=1}^{n_u} c_{u_i} 2^{\gamma_{u_i}} \quad , \quad a_u = \sum_{i=1}^{n'_u} a_{u_i} 2^{\alpha_{u_i}} \quad , \quad b_u = \sum_{i=1}^{n''_u} b_{u_i} 2^{\beta_{u_i}}$$

Remarque

Sans restreindre la généralité, au moyen d'une transformation polynomiale, on peut supposer que $\forall\, u \in U$, $n_u = n'_u = n''_u = n$; $\forall\, i=1,\ldots,n$, $\gamma_{u_i} = \alpha_{u_i} = \beta_{u_i} = \gamma_i$ et finalement $\gamma_{i+1} > \gamma_i$.

Remarque

Soit ϕ un flot compatible obtenu au moyen de l'algorithme B_3 et soit pour $\forall\, i \in \{1,\ldots,n\}$, $\phi_{u_i} = f_{u_i} 2^{\gamma_i}$, alors $|f_{u_i}| \leq \underset{u \in U}{\text{Max}} (|a_{u_i}|, |b_{u_i}|)\, 2^{\frac{|U| \times |X|}{2}}$

En conséquence, si l'on choisit de concatener(polynomialement) les coefficients a_{u_i} et a_{u_i+2} et b_{u_i} et b_{u_i+1} si

$\gamma_i \leq 2 \text{Log}_2 \underset{u \in U}{\text{Max}} (|a_{u_i}|, |b_{u_i}|) + 2\,|U| \times |X| + 2 + \gamma_{i+1}$, l'hypothèse qu'il y ait un flot compatible, implique qu'il y en ait un compatible pour tout les intervalles $[a'^{j}_{u}, b'^{j}_{u}]$ avec $a'^{j}_{u} = \frac{1}{2^{\gamma_j}} \sum_{i=j}^{n} a_{u_i} 2^{\gamma_i}$, $b'^{j}_{u} = \frac{1}{2^{\gamma_j}} \sum_{i=j}^{n} b_{u_i} 2^{\gamma_i}$.

En effet $\phi_{u_i} < \underset{u \in U}{\text{Max}} (|a_{u_i}|, |b_{u_i}|) \times 2^{\frac{|U| \times |X|}{2}} 2^{\gamma_i}$ et

$$\phi_{u_{i+1}} > - \text{Max} (|a_{u_i}|, |b_{u_i}|) \times 2^{\frac{|U| \times |X|}{2}} 2^{\gamma_{i+1}}$$

et si ϕ'_{u_i+1} n'est pas dans ses bornes (d'une unité $\times\, 2^{\gamma_{i+1}}$) , $\phi_{u_i} < 2^{\gamma_{i+1}}$.

On effectue donc cette concaténation ; et soient les γ_i suffisamment distants les uns des autres. On a alors l'algorithme F_3 :

0. $i = n$, $\phi_o = 0$, aller en 1.

1. Appliquer $\text{Log}_2 \max_{u \in U} (|a_{u_i}|, |b_{u_i}|)$ fois l'algorithme F_1 à une petite variante près. Lors de la première application de F_2 pour chaque nouvel i on peut avoir à chercher un cycle dans lequel le flot diminue.

2. si $i = 1$ stop.
   Sinon multiplier la solution obtenue par
   $2^{\gamma_i - \gamma_{i-1} - \text{Log}_2 \left(\max_{u \in U} (|a_{u_{i-1}}|, |b_{u_{i-1}}|) \right) - 1}$. Poser $i \leftarrow i-1$, aller en 1.

La modification des γ_i implique que si le problème général a une solution, les différents problèmes résolus à la fin de chacune des étapes en auront une.

Remarque

Dans la cas où a_u et b_u auraient des expressions de la forme : $a_u = \sum_{i=1}^{n} a_{u_i} k^{\alpha_i}$, $b_u = \sum_{i=1}^{n} b_{u_i} k^{\beta_i}$, où k peut ne pas être polynomial en la longueur des données du problème, il suffira, une fois les mêmes modifications ayant été faites, d'exprimer a_{u_i} et b_{u_i} dans la base 2.

Avant de décrire un moyen de traiter les cas où les données sont des types 3 ou 4, nous allons faire l'étude du problème général de la programmation linéaire.

IV - PROBLEME DE LA PROGRAMMATION LINEAIRE.

Soit A une matrice indicée en lignes par L et en colonnes par C avec $|L| = m$; $|C| = n$. Ce problème est polynomialement équivalent à (1).

Trouver $x \in R^C$ tel que :

$$(1) \qquad \begin{cases} A\,x \leqslant b \quad , \quad A_{ij} \in Z \quad , \quad b \in R^L \\ x \geqslant 0 \end{cases}$$

Soit un deuxième problème :

$$(2) \qquad \begin{cases} A\,x \leqslant b + \varepsilon \\ \qquad\qquad \text{avec } \varepsilon' = \dfrac{1}{2mnM\alpha^2} \ , \ \varepsilon = \varepsilon' \eta \ , \ M = \underset{\substack{i \in L \\ i \in C}}{\text{Max}} |A_{ij}| , \alpha - M^m \times m^{\frac{m}{2}} \\ x \geqslant -\varepsilon \end{cases}$$

On a le lemme [14] [15]

(1) non vide $\Leftrightarrow$ (2) non vide

Dem : (2) $\Leftrightarrow$ (3) $\begin{cases} Ax + Uy = b+\varepsilon \\ x > -\varepsilon \ , \ y > 0 \end{cases}$ avec U matrice unité de R^L.

On sait que si (3) a une solution, (3) a une solution extreme :

Soit donc une telle solution : $I \subset (C \cup L)$, $I = (I_1, I_2)$ avec $I_1 \subset C$, $I_2 \subset L$, soit $B = (A^{I_1} , U^{I_2})^{-1}$, la solution de base $s = (x_{I_1}, x_{I_2})$ attachée à I s'écrit :

$$x_{C \setminus I_1} = -\varepsilon \quad , \quad x_{U \setminus I_2} = 0$$

$x_I = B\,b + B\,\varepsilon + B\,A^{\overline{I}_1}\,\varepsilon$ $(\overline{I}_1 = C \setminus I_1)$, de plus on sait que $x_{I_1} > -\varepsilon$, $x_{I_2} > 0$.

Le théorème d'Hadamard implique $\frac{1}{\alpha} \leqslant B_{ij} \leqslant \alpha$, et en rappelant $\hat{x}_I = B\, b$,

$$\forall\, i \in I \quad , \quad \hat{x}_i = \sum_{j \in L} b_{ij}\, b_i \quad \text{ou en posant} \quad B_{ij} = B'_{ij} \times \mathrm{Det}\,(B)$$

$$\hat{x}_i = \mathrm{Det}\,(B) \sum_{j \in L} B'_{ij}\, b'_i \quad , \quad \text{avec } \mathrm{Det}\,(B) \geqslant \frac{1}{\alpha} \, , \; B'_{ij} \leqslant \alpha, \; B'_{ij} \in Z.$$

Finissons la démonstration dans le cas où $b \in Z$, en se réservant le droit de la reprendre ici dans les autres cas.

Supposons que $\forall\, i \in I$, $x_i > -\varepsilon$ et que $i \in I$ tel que $\hat{x}_i < 0$, alors comme $B'_{ij} \in Z$, $\hat{x}_i \leqslant -\frac{1}{\alpha}$ d'où :

$$x_i \leqslant -\frac{1}{\alpha} + \varepsilon \, |\mathrm{Det}(B)| \, \left| \left| \sum_{j \in L} B'_{ij} \left(1 + \sum_{p \in I_1} A_{jk} \right) \right| \right| \leqslant -\frac{1}{\alpha} + \varepsilon\, \alpha\, mnM.$$

et en prenant $\eta = 1$ et en remplaçant ε par sa valeur :

$$x_i \leqslant -\frac{1}{\alpha} + \frac{1}{2\alpha} \leqslant -\varepsilon$$

ce qui est contraire à l'hypothèse $x_i \geqslant -\varepsilon$ et entraine le résultat.

Ce lemme entraine que si (1) est non vide (2) contient une sphère de rayon ε, ce qui permet d'utiliser l'algorithme des ellipsoïdes de Kachian-Shor [12],[19] au problème (2). A partir de la solution obtenue, en n étapes de la méthode Simplex [3] dans laquelle les calculs matriciels sont effectués au moyen d'algorithmes polynomiaux, on trouve une base réalisable I d'où l'on déduit une solution de (1).

IV 1 - Les différents cas où b appartient aux divers ensembles de données.

On peut reprendre la fin de cette démonstration avec des données du type 1 (polynomes) ou du type 2, entiers représentés de façon particulière,

et ce de façon identique au cas entier, cependant l'algorithme de Kachian-Shor ne donnera pas toujours un résultat polynomial.

Interessons nous essentiellement au cas où b est du type 3 ou 4.

Dans ces deux cas, soit $\Lambda = \underset{i,j}{\text{Max}} (\text{Log}_2|B_{ij}|)$, on sait qu'il existe un polynome P en Λ (on pourrait bien entendu ne considérer qu'un polynome en les données du problème puisque Λ est polynomial en la donnée du problème), tel que si les b_i sont calculés avec $P(\Lambda)$ chiffres significatifs (en base 2), et soit b'_i, ces approximations des b_i, et si

$$\sum_{j \in L} B'_{ij}\, b'_i < -\frac{1}{2^{P(\Lambda)}} \times 2 \times m, \text{ alors } \sum_{j \in L} B'_{ij}\, b_i < -\frac{1}{2^{P(\Lambda)}}, \text{ et si}$$

$$-\frac{1}{2^{P(\Lambda)}} \times 2^{\Lambda} \times m < \sum_{j \in L} B'_{ij}\, b'_i < \frac{1}{2^{P(\Lambda)}} \times 2^{\Lambda} \times m,$$

$$\text{alors } \sum_{j \in L} B'_{ij}\, b_i = 0, \text{ et si}$$

$$\frac{1}{2^{P(\Lambda)}} \times 2^{\Lambda} \times m < \sum_{j \in L} B'_{ij}\, b'_i, \text{ alors } \sum_{j \in L} B'_{ij}\, b_i > \frac{1}{2^{P(\Lambda)}}.$$

Soit $\eta = \frac{1}{2^{P(\Lambda)}}$, posons $x'_I = B\,b' + B\,\varepsilon + B\,A^{I_1}\varepsilon$, $\hat{x}_I = B\,b$.

Supposons que $\forall\, i \in I$, $x'_i > -\varepsilon$ et $i \in I$ tel que $\hat{x}_i < 0$, alors $\hat{x}_i < -\frac{1}{\alpha} \times \frac{1}{2^{P(\Lambda)}}$

et donc $x_i \leq -\frac{1}{\alpha} \times \frac{1}{2^{P(\Lambda)}} + \varepsilon|\text{Det } B| \left| \sum_{i \in L} B'_{ij} \left(1 + \sum_{k \in I_1} A_j\, k\right) \right|$

et en remplaçant ε par sa valeur en fonction de η

$$x_i < -\frac{1}{\alpha}\eta + \frac{1}{2\alpha}\eta < -\varepsilon$$

d'où la contradiction.

On obtient la même conclusion sur la polynomialité de l'algorithme de Kachian-Shor, et sur l'application de la méthode Simplex pour obtenir la solution de (1).

Remarque

Dans les développements traditionnels sir l'algorithme de Kachian, la méthode des fractions continues est utilisée généralement pour obtenir une solution de (1) à partir de celle de (2), il est clair que cette méthode est ici inappliquable, ces développements en fractions continues étant, a priori, infinis. La méthode développée ici, qui est reprise de [14],[15] est bien préférable.

Remarque

On peut très aisément au moyen de données du type 2 fabriquer des P.L. tels que la méthode Simplex sans cycle ait un nombre de calculs, une complexité,(a priori exponentiel : nombre de changements de bases bornées par $\binom{n}{m}$) plus faible que celle de l'algorithme de Kachian !...

Remarque

Si le programme linéaire est un problème de flot à coût maximum il faudra tenir compte des particularités (Det(B) = 1, etc...) pour établir une meilleure borne, puis on pourra obtenir une solution du problème approché au moyen de l'algorithme F_1.

V - CONCLUSION

Nous aurions pu dans cet article multiplier les exemples d'algorithmes pour lesquels on peut élargir les ensembles de données. Il est clair que je ne désire pas que tous les algorithmes d'optimisation (ou presque) s'appellent désormais "algorithme de Maurras", les modifications faites ici sont "naturelles". En revanche nous avons prouvé contrairement à certaines affirmations hâtives [1] p. 31, que l'on pouvait considérer comme données d'un programme linéaire des nombres algèbriques, nous n'avons pas fait le développement du cas où ces réels seraient des éléments de la matrice A (cas des programmes linéaires), ceci pour ne pas trop compliquer l'exposé. Dans ce cas il faut que dans l'expression formelle du déterminant d'une matrice, un petit (polymonial) nombre de monomes comportent des réels, sauf dans le cas de données du type 4 où l'on peut regrouper les termes de même degré*. Nous pouvons affirmer pour conclure que des problèmes de complexité persistent pour la programmation linéaire.

Je tiens à remercier J.P. Lafon, pour l'aide formatrice qu'il m'a prodiguée dans mes premiers contacts avec les nombres algèbriques.

---

* Dans le cas où les coûts et les seconds membres sont du type 3 ou 4, seule une ligne de la matrice transformée les contients (par dualité) et dans tous les traitements matriciels on peut la prendre en dernier. En conséquence on a une seule combinaison linéaire de ces nombres à considérer.

VI - BIBLIOGRAPHIE

1 A. BACHEM and M. GROSTCHEL , "New Aspects of Polyhedral Theory" Rep. n° 79149-OR, Institut Für Okonometrie und Operation Research. Bonn Universität.

2 C. BERGE "Graphes et Hypergraphes" , Dunod, Paris 1970.

3 G.B. DANTZIG "Linear Programming and Extensions" , Princeton University Press, Princeton, New Jersey, 1963.

4 G.B. DANTZIG "All shortest routes in a graph" , Proc. I.C.C. Conference on Theory of Graph, Rome Gordon and Breach, N.Y. p. 91-92.

5 E.W. DJIKSTRA "A note on linear problems in connection with Graphs", Numerische Math. 1. pp. 169-271.

6 J. EDMONDS "Paths, Trees and Flowers", Can. J. Math. 17 (1965) 449-467.

7 J. EDMONDS "Systems of Distincts Representatives and Linear Algebra", J. Res. Nat. Bur. Stds. B 71 B (1967) 241-245.

8 J. EDMONDS and R.M. KARP , "Theoritical Improvements in Algorithmic Efficiency for Network Flow Problems", J. ACM 19 (1972) 248-264.

9 L.R. FORD and D.R. FULKERSON , "Flows in Networks" , Princeton University Press, Princeton, New Jersey.

10 D.R. FULKERSON "An out of Kilter Method for Minimal Cost Flow Problem", Siam J. Appl. Math., 9 (1961) 18-27.

11 M.R. GAREY, D.S. JOHNSON , "Computers and Intractability..." W.H. Freeman and Co. San Francisco (1979).

12 L.G. KACHIAN "A Polynomial Algorithm in Linear Programming", Soviet Math. Dokl., 20 (1979) 191-194.

13 R.M. KARP "Reducibility Among Combinatorial Problems", in Complexity of Computer Computations, R.E. Miller, et. al (éds), Plenum Press, New-York, 1972.

14 J.F. MAURRAS "Bons Algorithmes, Vieilles Idées", Note E d F, HR 320320, 1978.

15 J.F. MAURRAS and K. TRUEMPER and M. AKGUL , "Polynomial Algorithms For a class of Linear Programs, Math. Programming, 21 (1981) 121-136.

16 G.L. MILLER "Riemann's Hypothesis and Test for Primality", Proc. of the Seventh Annual ACM Symp. on Theory of Computing (1971) pp. 234-239.

17 G.J. MINTY "Monotone Networks", Proc. Roy. Soc. London, SER. A. 257 (1960) 194-212.

18 M.O. RABIN "Probabilistic Algorithms" in Algorithms and Complexity, J.F. Traub ed., Academic Press., New-York, 1976.

19 N.Z. SHOR "Convergence Rate of the Gradient Descent with Dilatation of the space", Cybernetics 6 (1970) 102-108.

20 B.L. Van der WANDEN, "Modern Algebra", Frederick Ungar Publishing Co. New-York, 1950.