MINISTERE DE L'ENSEIGNEMENT SUPERIEURET DE

LA RECHERCHE SCIENTIFIQUE

UNIVERSITE ABDELHAMID IBNBADIS – MOSTAGANEM

Faculté des Sciences Exactes et de l'Informatique

Département de Mathématiques et d'Informatique

Filière Mathématiques

MEMOIRE DE FIN D'ETUDES

Pour l'Obtention du Diplôme de Master en Mathématiques

Option : **Modélisation Contrôle et Optimisation**

THEME :

Sur La Complexité En Programmation Linéaire

Soutenu le 30 /05/2016

Etudiante : LARBI Aicha

LARBAOUI Hadjira

Président : **BELGACEM Rachid** **U. Chlef**

Encadrant : **AMIR Abdessamad** **U. Mostaganem**

Examinateur **BOKHARI Ahmed** **U. Media**

Année Universitaire 2015/20016

Résumé

L'objectif de ce travail est de montrer que la méthode du simplexe n'est pas polynomial. Pour se faire nous étudions le papier classique de klee et Miny. Ils présentent une exemple sur le quel la méthode du simplexe est exponentielle. Nous faisons une comparaison numérique entre le Simplexe et une méthode de point intérieur.

Mots clés : Programme linéaire, Exemple de Klee et Minty, méthode de points intérieurs, Complexité algorithmique.

Dédicaces

Je dédie ce modeste travail :

À ma très chère mère ;

À mon très cher père ;

À tous mes proches de la famille Larbi, et plus particulièrement, mes soeurs et mes frères tout à son nom ;

À toutes mes chères amies et mes collègues de l'Université de Mostaganem ;

À toutes les personnes que j'aime.

Aicha

Dédicaces

Je dédie ce modeste travail :

À ma mère ;

À mon père ;

À mes proches de mes frères et mes sœurs, chacun à son nom ;

À toute la famille ;

À toutes mes amies ;

Hadjira

Remerciements

Tout d'abord, nous remercions le Dieu, notre créateur de nos avoir donné les forces, la volonté et le courage afin d'accomplir ce travail modeste.

Nous adressons le grand remerciement à notre encadreur **Mr. AMIR Abdassamed** qui a proposé le thème de ce mémoire, pour ses conseils et ses dirigés du début à la fin de ce travail.

Nous tenons également à remercier messieurs les membres de jury pour l'honneur qu'ils nos ont fait en acceptant de siéger à notre soutenance, tout particulièrement :

Mr. **BELGACEM Rachid** pour nous avoir fait l'honneur de présider le jury de cette mémoire.Nous souhaitons exprimer notre gratitude à **Mr**. **BOKHARI Ahmed** pour avoir faire de lecteur notre mémoire, aller l'examiner et il peut évaluer cette mémoire. Nous vous remercions pour l'intérêt que vous avez porté à ce travail et pour vos précieux conseils et remarques.

Finalement, nous tenons à exprimer notre profonde gratitude à nos familles qui nous ont toujours soutenues et à tout ce qui participe de réaliser ce mémoire. Ainsi que l'ensemble des enseignants qui ont contribué à notre formation.

Table des matières

Liste des Tableaux

Chapitre I

Introduction En Complexité Algorithmique

Chapitre 3

Expériences Numérique

Liste des Figures

Chapitre 2

Simplexe N'est Pas Un Algorithme Polynomial

Introduction

Après la deuxième guère mondiale, la méthode du simplexe s'est imposée comme la seule méthode efficace pour la résolution des programmes linéaires. Il a été prouvé que cette méthode est vraiment efficiente et a un bon rendement en pratique, mais ce n'est pas le cas en théorie de sorte qu'elle présente une complexité exponentielle. En effet, Klee et Minty ont construit un exemple en 1972 ; qui montre que l'algorithme du simplexe est de complexité exponentielle. Ce mémoire est composé de trois chapitres. Le premier chapitre est consacré à des rappels préliminaires contenant des notions sur la complexité. Le deuxième chapitre est consacré à l'étude de complexité pour l'algorithme du Simplexe. Au dernier chapitre, nous présentons une application qui résoud l'exemple de Klee et Minty par une méthode de point interieur et la méthode du Simplexe, une comparaison numérique et réalisée.

Chapitre 1

Introduction En Complexité Algorithmique

1.1 Résoudre Efficacement Un Système Linéaire

Avant d'introduire la complexité en programmation linéaire, nous tournons notre attention au coût de calcul nécessaire pour la résolution des systèmes d'équations de la forme

$$Ax = b, \text{ avec } b \in \mathbb{R}^{(n\times 1)} \text{ et } A \in \mathbb{R}^{(n\times n)} \text{ et non singulière.}$$

Nous mesurons le coût en comptant le nombre d'opérations effectué en virgule flottante (flops [1]), +, -, *, et / (addition, soustraction, multiplication et division), nécessaires pour obtenir la solution x. Pour se faire, nous considérons deux méthodes ; La méthode d'échange de Jordan et la méthode de Cramer basée sur le calcul du déterminant à partir de son expression comme somme de monômes (cette méthode était appelée "Règle de Sarrus").

1.1.1 La Méthode D'échange De Jordan

Cette méthode est basée sur la régle d'échange de Jordans réalisée à chaque itération après avoir déterminer le pivot $A(r, s)$ qui devrait être non nul. La nouvelle matrice (ou tableau) B est déterminée par les régles suivantes :

– L'élément situé sur la ligne et la colonne du pivot est

$$B(r, s) = \frac{1}{A\,(r, s)}.$$

– Les élément de la ligne du pivot sont obtenus par

$$B(r,J) = -\frac{A(r,J)}{A(r,s)}, \text{ où } J = \{j = 1,...,n \text{ et } j \neq s\}.$$

– Les élément de la colonne du pivot sont obtenus par

$$B(I,s) = \frac{A(I,s)}{A(r,s)}, \text{ où } I = \{i = 1,...,n \text{ et } i \neq r\}.$$

– Les autres éléments sont obtenus par

$$B(I,J) = A(I,J) - B(I,s) * A(r,J).$$

Les flops nécessaires pour effectuer un échange de Jordan sont présentés dans le tableau suivant :

Elément	flops
Pivot	1
Ligne du pivot	n-1
La colonne du pivot	n-1
Les autres éléments	2(n-1)(n-1)

Il existe donc $2n^2 - 2n + 1$ flops pour chaque échange de Jordan, il est nécessaire d'effectuer n échanges de Jordan afin d'inverser la matrice non singulière A. Le nombre total devient $2n^3 - 2n^2 + n$ flops, pour n assez grand nous pouvons approximer ce chiffre par $2n^3$. Ayant obtenu A^{-1}, nous le multipliant par b afin d'obtenir x. cette multiplication matrice-vecteur nécessite en plus $2n^2$ flops, mais pour n assez grand, ce coût est faible par rapport au coût de calcul de A^{-1}.

1.1.2 Méthode De Cramer Basée Sur Le Calcul Du Déterminant

On va ce limiter à l'évaluation du nombre d'opérations qu'il faut pour calculer le déterminant de A à partir de son expression comme somme de monômes

$$Det(A) = \sum_{\sigma \in S_n} \epsilon(\sigma) \prod_{i=1}^{n} A_{(\sigma(i),i)}.$$

Il y a $n!$ monômes à ajouter, ce qui demande $n! - 1$ additions ; la moitié des monômes demandent un changement de signe, ce qui fait $n!/2$ multiplications par la constante -1 ; enfin, le calcul de chaque monôme demande $n - 1$ multiplications d'inconnues. En tout, cela fait $n! - 1 + n!/2 + (n-1).n! = (n + 1/2).n! - 1$ opérations, ce qui est un nombre énorme, rendant le calcul totalement impraticable, sans rajoutons à ceci les déterminants d'ordre n qu'on doit calculer à fin de trouvez les n composantes x_i.

1.2 Taux de croissance de la complexité

Dans l'étude de la complexité d'un algorithme, nous sommes souvent intéressés uniquement dans le comportement de l'algorithme lorsque les tailles des entrées sont grandes, car se sont ces entrées qui vont déterminer les limites de l'applicabilité de l'algorithme. Le taux de croissance en nombre d'opérations et flagrant entre les deux algorithmes précédent de résolution des systèmes linéaire quand la taille n augmente. Nous nous intéressons, par conséquent, au taux de croissance de la complexité d'un algorithme. Pour faire face à des taux de croissance des fonctions la définition qui suit est utile.

Définition 1.2.1 *Soient $f(n)$ et $g(n)$ deux fonctions définies de $\mathbb{N} \to \mathbb{R}^+$.*

(i) *On écrit $f(n) = O(g(n))$, s'ils existent une constante $\gamma > 0$ et un entier n_0 tel que pour tout $n \geq n_0$, $f(n) \leq \gamma(g(n))$.*

(ii) *On écrit $f(n) = \Theta(g(n))$, s'ils existent une constante $\gamma > 0$, une constante $\beta > 0$ et un entier n_0 tel que pour tout $n \geq n_0$, $\beta g(n) \leq f(n) \leq \gamma g(n)$.*

En utilisant cette notation, le taux de croissance de la complexité d'un algorithme peut être majorée par des phrases comme "prend du temps $O(n^3)$".

1.3 La Taille Des Instances

On mesure la complexité d'un algorithme en fonction de la taille des entrées d'un algorithme. Mais, quelle est la taille d'une entrée ? Par exemple, dans le problème de tester si un nombre entier est premier, une instance est simplement un entier. Il y a plusieurs façons de représenter des nombres entiers ; les plus courantes sont des systèmes arithmétiques dans une certaine base fixe, décimal ou binaire. Dans ces systèmes, le nombre de symboles nécessaires pour représenter un nombre entier n est $\lceil \log_B n \rceil$ avec $B > 2$ la base utilisée. Rappelons que $\log_B n = \frac{\log n}{\log B}$, la taille de la représentation de n est $\Theta(\log n)$. Quelle est la taille d'un programme linéaire ? Comme précédemment, nous supposons que les entrées A, b et c sont des nombres entiers. Ainsi, la taille d'un programme linéaire serait le nombre de symboles nécessaires pour écrire A,b, et c. Puisque cela peut être fait en énumérant les éléments des matrices en binaire (ou décimal), en utilisant des délimiteurs appropriées sur les coefficients

des lignes et colonnes du tableau, la taille d'un PL à n variables et m contraintes est $\Theta(mn + \lceil \log P \rceil)$, où P est le produit de tous les coefficients non nuls.

1.3.1 Classe des Algorithmes Polynomiaux

Aujourd'hui, il y a un accord général entre informaticiens et numériciens qu'un algorithme admet une solution utile uniquement, si son taux de croissance est polynomiale. Par exemple les algorithme de complexité $O(n)$ ou $O(n^3)$ sont acceptables en pratique. Bien entendu, les algorithmes pour lesquels la complexité asymptotique n'est pas un polynôme, mais majorée par un polynôme, sont également admissibles. comme exemples $O(n^{2.5})$ ou $O(n \log n)$. Pour comprendre la signification des algorithmes polynomialement bornées en tant que classe, Considérons les algorithmes restants ; ceux dont la complexité violent toutes les bornes polynomiales, On dit habituellement que ces algorithmes admettent une complexité exponentielle, parce que 2^n est le paradigme des taux de croissance non polynomiales . D'autres exemples de taux de croissance exponentielle sont k^n (pour $k > 1$), $n!$, n^n et $n^{\log(n)}$. Il est évident que, lorsque la taille de l'instance croît, tout algorithme polynomial finira par être plus efficace que tout exponentielle (voir le tableau ci-dessous).

La croissance des fonctions polynomiales et exponentielles

Fonction	valeurs approximatives		
n	10	100	1000
$nlog(n)$	33	664	9966
n^3	1000	1000000	10^9
$10^6 n^8$	1014	10^{22}	10^{30}
2^n	1024	1.27×10^{30}	1.05×10^{301}
$n^{\log(n)}$	2099	1.93×10^{13}	7.89×10^{29}
$n!$	3628800	10^{158}	4×10^{2567}

Une autre caractéristique positive des algorithmes polynomiaux et d'avoir l'avantage de mieux tirer profit des progrès technologiques. Par exemple, chaque fois qu'une percée technologique augmente de dix fois la vitesse des ordinateurs, la taille de la plus grande instance résolu par un algorithme polynomial dans une heure, sera multipliée par une constante comprise entre 1et 10. En revanche, un algorithme exponentiel éprouvera seulement une légère augmentation comme le montre le tableau ci-dessous. Enfin, nous pouvons remarquer que les algorithmes polynomiaux ont de belles propriétés de "fermeture" ; un algorithme polynomial peut invoquer un autre algorithme polynômial comme "Sous-programme", et l'algorithme résultant

sera toujours polynômial.

Mieux tirer Parti de la thecnologie des Algorithmes Plynômiaux

Fonction	Taille des données résolu en une journée	Taille des données résolu en une journée Sur une machine dix fois plus rapide
n	10^{12}	10^{13}
$nlogn$	0.948×10^{11}	0.87×10^{12}
n^2	10^6	3.16×10^6
n^3	10^4	2.15×10^4
$10^8 n^4$	10	18
2^n	40	43
10^n	12	13
$n^{\log n}$	79	95
$n!$	14	15

Chapitre 2

Le Simplexe n'est Pas Un Algorithme Polynomial

2.1 La méthode du Simplexe

Nous considérons le problème de programmation linéaire suivant :

$$\begin{cases} min\ c^T x \\ Ax \leq b \\ x \geq 0 \end{cases} .$$

Avec $A \in \mathbb{R}^{m*n}$, $b \in \mathbb{R}^{m*1}$ et $c \in \mathbb{R}^{n*1}$ avec rg(A)=m. Une des méthodes la plus utilisées pour résoudre ce problème est la méthode du simplexe. Cette méthode a été développée à la fin des années 40 par G. Dantzig. Elle évolue sur la frontière du domaine réalisable de sommet en sommet adjacent, en réduisant la valeur de l'objectif jusqu'à l'optimum. Un critère simple permet de reconnaître le sommet optimal. Le nombre de sommet étant fini, l'algorithme ainsi défini converge en un nombre fini d'itération n'excédant pas le nombre C_n^m, sous l'hypothèse que tous les sommets visités sont non dégénérés[1]. Dans le cas dégénéré l'algorithme risque de cycler, cependant il existe des techniques convenables pour éviter ce phénomène. En général, la méthode du simplexe possède un comportement numérique très satisfaisant confirmé par ses applications multiples dans la résolution d'une large classe de problèmes pratiques. Les compréhensions de la méthode du simplexe est nécessaire pour la suite, nous renvoyons le lecteur aux référence [4], [5] pour un bon exposé de cette méthode. En

[1] Un sommet correspond à une solution de base $x = (x_1, x_2, ..., x_m, 0, ..., 0) \in \mathbb{R}^n$. Une solution de base est dite non-dégénéré si les $x_i > 0$ pout $i = 1, ..., m$.

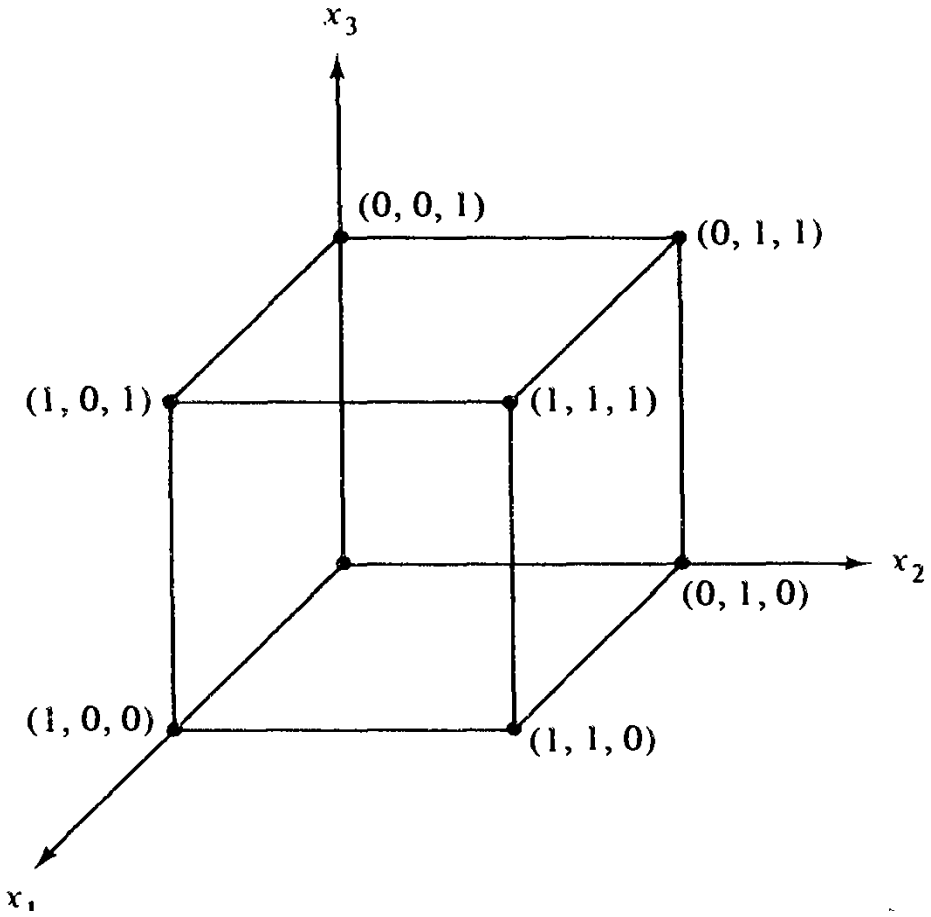
théorie, la méthode n'a pas autant de succès, elle est plutôt jugée inefficace par sa complexité arithmétique exponentielle de l'ordre $\bigcirc(2^n)$ opérations. Le but de ce chapitre est de montrer ceci.

2.2 Le d-cube

Définition 2.2.1 *Le d-cube est définie comme suite :*

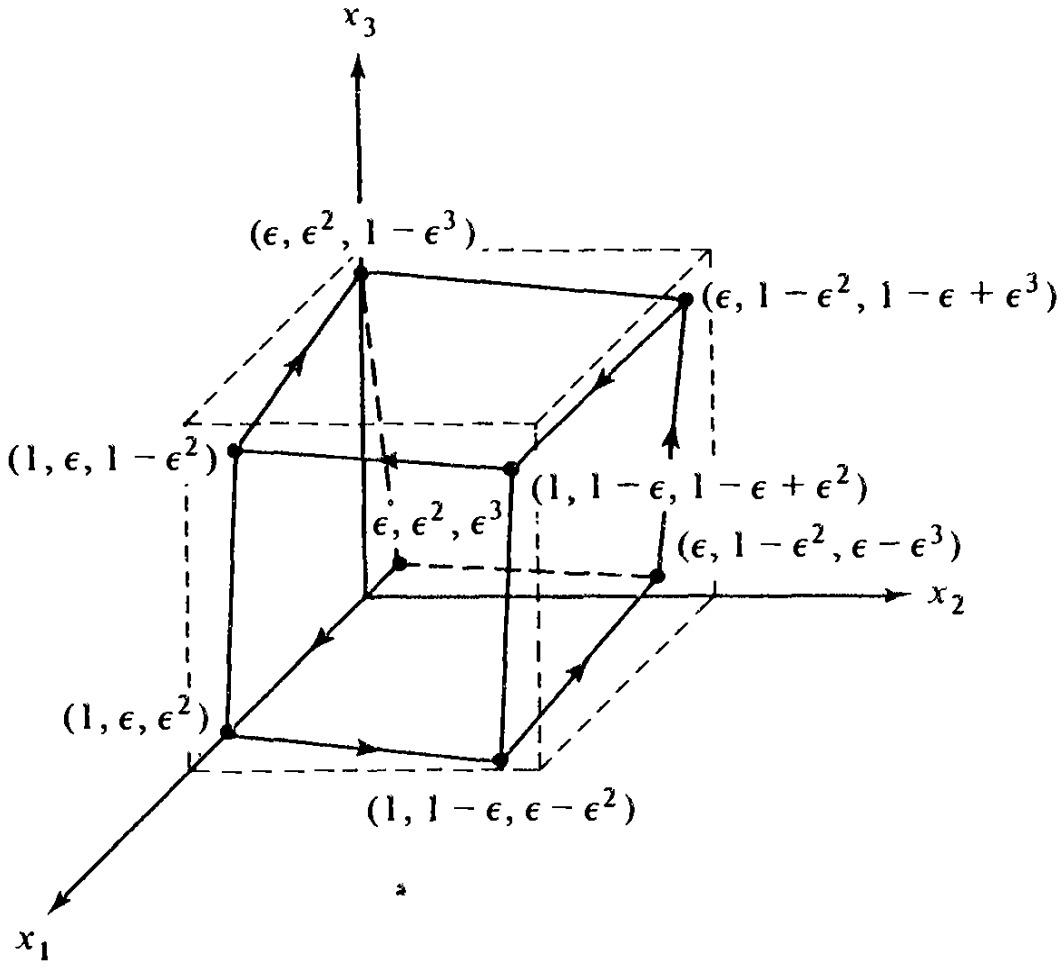
$$\{x \in \mathbb{R}^d \ / \ \ 0 \leq x_j \leq 1 \ , \ \ j = 1, 2, ..., d\}.$$

Exemple 2.2.1 *La* **Figure 1**, *représente un* 3*-cube.*



(**Figure 1**)

3*-cube contient* 6 *faces et* 8 *sommets.*



(**Figure 2**)

Le polytope qu'on va considérer est similaire au d-cube. Il est définie par :

$$\begin{cases} x_1 \geq \varepsilon \\ x_1 \leq 1 \\ x_j \geq \varepsilon x_{j-1} \\ x_j \leq 1 - \varepsilon x_{j-1} \quad j = 2, 3, ...d. \end{cases} . \tag{a1}$$

Avec $\varepsilon \in \left]0, \frac{1}{2}\right[$. La **Figure 2**, représente ce cube pour $d = 3$. Si $\varepsilon \longmapsto 0$ on a le d-cube. Pour ecrire (a1) sous forme standard on ajoute des variables d' écart, alors $m = 2d$, $n = 3d$

et nous essayons de maximiser x_d ($\max x_d = -\min -x_d$).

$$\begin{cases} \min -x_d & & \\ x_1 - r_1 = \varepsilon & & (1) \\ x_1 + s_1 = 1 & & (2) \\ x_j - \varepsilon x_{j-1} - r_j = 0 & & (3) \\ x_j + \varepsilon x_{j-1} + s_j = 1 & j = 2, 3, ..., d & (4) \\ x_j, r_j, s_j \geq 0 & j = 1, ..., d & \end{cases} . \tag{a2}$$

Lemme 2.2.1 *[3]L'ensemble des solutions de base admissibles de (a2) sont de la forme {x_1, ..., x_d, r_1, ..., r_d, s_1,.....s_d} qui contiennent tous les x_j pour $j = 1, ...d$ et exactement un des s_j ou r_j pour chaque $j = 1, ..., d$. En plus, toutes ces bases sont non dégénérées.*

Preuve. On a $x_1 \geq \varepsilon$ et $x_{j+1} \geq \varepsilon x_j$ pour $j = 1, ..., d-1$, nous avons donc dans toute solution réalisable, $x_j \geq \varepsilon^j > 0$. Donc les bases réalisables de (a2) doivent contenir toutes les colonnes d correspondant à x. Supposons que $r_j = s_j = 0$ pour certains j. Si $j = 1$, alors $\varepsilon = x_1 = 1$, ce qui est absurde. Si $j > 1$, alors la troisième contrainte de (a2) donne $x_j = \varepsilon x_{j-1}$ et la quatrième donne $x_j + \varepsilon x_{j-1} = 1$, ou $2\varepsilon x_{j-1} = 1 \Rightarrow x_{j-1} = \frac{1}{2\varepsilon} > 1$, ce qui contredit le fait que $0 \leq x_j \leq 1$ pour toute j. Comme $m = 2d$ est une bases admisible contient l'une des colonnes correspondantes à s_j et r_j pour chaque j, nous déduisons que les bases sont non dégénéres. □

Nous écrirons une solution de base réalisable de (a2) est x^S, où S est le sous-ensemble de $\{1, 2, ..., d\}$ qui correspond au r non nul dans x^S. La valeur de x_j dans x^S sera notée par x_j^S. Nous aurons besoin des lemmes suivants.

Lemme 2.2.2 *[3]Supposons que $d \in S$ mais $d \notin S'$; alors $x_d^S > x_d^{S'}$. En plus, si $S' = S - \{d\}$, alors $x_d^{S'} = 1 - x_d^S$.*

Preuve. Puisque $d \in S$, $s_d = 0$ et la quatrième contrainte de (a2) donne $x_d^S = 1 - \varepsilon x_{d-1}^S$. On a $x_{d-1}^S \leq 1$ et $\varepsilon < \frac{1}{2}$; donc $x_d^S > \frac{1}{2}$. D'autre part, $r_d = 0$ car $d \notin S'$, la troisième contrainte donne $x_d^{S'} = \varepsilon x_{d-1}^{S'} < \frac{1}{2}$. Par coséquent $x_d^{S'} < x_d^S$. Pour la seconde partie, on remarque que si $S = S' \cup \{d\}$, alors $x_{d-1}^{S'} = x_{d-1}^S$ (car si $d-1 \in S \Longrightarrow d-1 \in \acute{S}$). Donc

$$\begin{array}{lll} x_d^{S'} & = \varepsilon x_{d-1}^{S'} & \text{de la troisième contrainte} \\ & = 1 - (1 - \varepsilon x_{d-1}^s) & \text{de la quatrième contrainte} \\ & = 1 - x_d^S. & \end{array}$$

□

Lemme 2.2.3 *[3]Supposons que les sous-ensembles de* $\{1, 2, ..., d\}$ *sont énumérés de manière que* $x_d^{S_1} \leq x_d^{S_2} \leq ... \leq x_d^{S_{2^d}}$. *Alors les inégalités sont strict et les solutions de base réalisable* $x_d^{S_j}$ *et* $x_d^{S_{j+1}}$ *sont adjacentes pour* $j = 1, 2, ..., 2^d - 1$.

Preuve. Par récurrence sur d, pour $d = 1$: nous avons deux solutions de base réalisable à savoir de $(x_1,\ r_1,\ s_1) = (\varepsilon,\ 0,\ 1-\varepsilon)$ et $(1,\ 1-\varepsilon,\ 0)$. Les deux solutions de base réalisables admet des x_1 défférents, ils sont certainement adjacents. On suppose que le lemme est vrai pour d-cube, et soient S_1, $...S_{2^d}$ énumération. Approprée les lemmes S_1, $...S_{2^d}$ sont également sous-ensembles de $\{1, 2, ..., d+1\}$, et en fait $x_{d+1}^{S_j} = \varepsilon x_d^{S_j}$. Par l'hypothèse de récurrence, $x_{d+1}^{S_1} < x_{d+1}^{S_2} < ... < x_{d+1}^{S_{2^d}}$. On considère sous-ensemble de $\{1, 2, ..., d+1\}$, de même $S'_j = S_j \cup \{d+1\}$, $j = 1, ..., 2^d$. Nous avons d'après le lemme (2.2.2) $x_{d+1}^{S'_J} > x_{d+1}^{S_{2^d}}$ et $x_{d+1}^{S'_j} = 1 - x_{d-1}^{S_j}$. Par l'hypothèses de récurrence x^{S_j} et $x^{S_{j+1}}$ sont adjacents, et de même pour $x^{S'_j}$ et $x^{S'_{j+1}}$. en plus, $x^{S_{2^d}}$ et $x^{S'_{2^d}}$ sont adjacents aussi, puisque cette dernière base résulte en ajoutant r_{d+1} et en omettant s_{d+1}. Dou le résultat. □

Théorème 2.2.1 *[3]Pour chaque* $d > 1$ *il ya un programme linéaire avec* $2d$ *équations,* $3d$ *variables et coefficients entiers avec valeur absolue délimités par* 4, *telle que le simplexe peut prendre* $2^d - 1$ *itérations pour trouver l'optimum.*

Preuve. On prend $\varepsilon = \frac{1}{4}$ et multiplie toutes les équations de (a2) par 4, donc tous les coefficients sont des nombres entiers. Puisque la fonction objectif de (a2) est de maximiser x_d, l'exponentielle longue chaîne de solution de base realisable d'adjacentes dont l'existence est établie par le lemme 3 à la baisse des coûts.

$$\begin{cases} \min -x_d \\ x_1 - r_1 = \frac{1}{4} \\ x_1 + s_1 = 1 \\ x_j - \frac{1}{4}x_{j-1} - r_j = 0 \\ x_j + \frac{1}{4}x_{j-1} + s_j = 1 \quad j = 2, 3, ..., d \\ x_j, r_j, s_j \geq 0 \qquad\quad j = 1, ..., d \end{cases} \overset{(*\,4)}{\Rightarrow}$$

$$\begin{cases} \min -x_d \\ 4x_1 - 4r_1 = 1 \\ 4x_1 + 4s_1 = 4 \\ 4x_j - x_{j-1} - 4r_j = 0 \\ 4x_j + x_{j-1} + 4s_j = 4 \quad j = 2, 3, ..., d \\ x_j, r_j, s_j \geq 0 \qquad\quad j = 1, ..., d \end{cases}$$

□

Chapitre 3

Expériences Numérique

Ci-dessous un programme sous Matlab qui permet de résoudre le problème de Klee et Minty avec la méthode du simplexe. Nous avons écrit un code qui génere l'exemple de Kle et Minty, pour n'importe quel valeur de d et ε. Pour l'algorithme du simplexe nous avons utilisé la fonction Matlab Linprog avec l'option ('LargeScale','off'). Afin de faire une étude comparative avec d'autre approches de résolution des problème de programation linéaire, nous avons opté pour une méthode de point interieur, vue son caractére polynômiale. Le code utilisé ici pour une méthode de point intérieur est toujours la fonction Linprog avec l'option ('LargeScale','on').

3.1 Le programme

tic

clear all

clc

syms x

d=input('donner d :')

epsilon=input('donner epsilon :') %doit être entre 0 et $\frac{1}{2}$

c=[zeros$(d-1,1);-1]$;

A=zeros$(2*d,d)$;

b=zeros$(2*d,1)$;

for $i = 2 : d$

```
        A(1,1) = -1;
        A(2,1) = 1;
        A(2*i,i) = 1;
        A(2*i-1,i) = -1;
        A(2*i-1,i-1) =epsilon;
        A(2*i,i-1) =epsilon;
end
b(1) = -epsilon;
b(2) = 1;
for j = 3 : 2*d
    if mod(j,2) == 0
            b(j) = 1;
    else b(j) = 0;
    end
end
c
b
A
x0= [];     % on ne fournit pas de solution réalisable au départ
Aeq= []; beq = [];      % pas de contraintes d'égalités
lb= zeros(d,1);   % bornes inférieures = 0
ub= [];     % pas de borne supérieure
options = optimset('LargeScale','off')
[X,fval]=linprog(c,A,b,Aeq,beq,lb,ub,x0,options)

toc
```

3.2 Une Comparaison Entre La Méthode Du Point Interieur Et La Méthode Du Simplexe

d	Temps d'éxucution par la méthode du point interieur(seconde)	Temps d'éxucution par la méthode du Simplexe(seconde)
2	1.406712	2.719137
6	1.494153	2.978788
10	1.662239	3.124518
50	2.181572	3.371740
100	3.067146	3.882018
500	3.093063	6.545448
1000	8.844201	13.697504
3000	51.426874	Out of memory.
4000	88.264232	Out of memory.
5000	131.855052	Out of memory.
6000	136.734053	Out of memory.
7000	261.254648	Out of memory.
8000	Out of memory.	Out of memory.

Remarque 3.2.1 *La méthode du point interieur est rapide que Simplexe pour résoudre le probléme de Klee et Minty.*

3.3 Un Autre Exemple

Bien que l'exemple pathologique de Klee et Minty à montrer le caractère exponentiel de la méthode du simplexe, cette dernière est toujours utiliser dans les logiciels commerciaux. L'exemple suivant montre ce paradoxe.

$$\begin{cases} \max & 1700x_1 & + & 3200x_2 & & \\ & & & 3x_2 & \leq & 39 \\ & 1.5x_1 & + & 4x_2 & \leq & 60 \\ & 2x_1 & + & 3x_2 & \leq & 57 \\ & 3x_1 & & & \leq & 57 \\ & & x_1, & x_2 & \geq & 0 \end{cases}$$

Le programme :

tic

A = [0, 3; 1.5, 4; 2, 3; 3, 0] ;

b= [39; 60; 57; 57] ;

c= −[1700; 3200] ; % Attention, linprog cherche un min !

options = optimset('LargeScale','off') ;

Aeq= [] ; beq= [] ; % pas de contraintes d'égalités

lb=zeros(2, 1) ; % bornes inférieures = 0

ub= [] ; % pas de borne supérieure

x0= [] ; % on ne fournit pas de solution réalisable au départ

[X,fval] =linprog(c,A,b,Aeq,beq,lb,ub,x0,options)

toc

L'éxucution

Par La Méthode Du Simplexe

X =

13.7143

9.8571

fval =

-5.4857e+004

Elapsed time is 0.043480 seconds.

Par La Méthode Du Point Interieur

X =

13.7143

9.8571

fval =

-5.4857e+004

Elapsed time is 3.317691 seconds.

Conclusion

Les deux exemples présentés dans ce travail, montre à la fois le caractère exponentielle de l'algorithme du Simplexe, via l'exemple pathologique de Klee et Minty. Par contre le second exemple montrer aussi l'efficacité de cet algorithme. Malgré l'exemple de Klee et Minty, le Simplexe et toujours utilisé comme solveur dans la majorité des logiciels numérique. À titre d'exemple la dernière version de la fonction "linprog" de Matlab-R2016 contient toujours cet algorithme.

Bibliographie

[1] Charles F. Van Loan et Gene H. Golub, **Matrix Computations,** Edition United States of America (1996).

[2] Christos H. Papadimitrou et Kenneth Steiglitz, **Combinatorial Optimization : Algorithms and Complexity,** Edition Dover (1998).

[3] Klee V et Minty G.J. : **How good is the simplex algorithm** in Shisha Os (Ed.) "Inéqualities III, Academic Press (1972).

[4] J .C. Culioli, **Introduction à l'optimisation**, Edition Ellipses (1994).

[5] J. Teghem, **Programmation linéaire**, Edition Statistique et Mathématiques Appliquées (2003).