REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université MOULOUD MAMMERI DE TIZI -OUZOU

Faculté des Sciences
Département de Mathématiques

Mémoire de fin de cycle

En vue de l'obtention du diplôme de Master

Spécialité :

Recherche Opérationnelle

Thème

Résolution d'un problème de programmation linéaire en nombres entiers par la méthode de Branch and Bound

Présenté par : M^{elle} ***SMAH*** **Lisa**
M^{elle} ***IMERZOUKENE*** **Katia**

Devant le jury composé de :

Président :	M^r *CHEBBAH* Mohamed	M.C.B, UMMTO
Encadreur :	M^r *OUANES* Mohand	Professeur, UMMTO
Examinateur :	M^r *GOUBI* Mouloud	M.C.B, UMMTO

Promotion : 2019/2020

Remerciements

Nous tenons tout d'abord à remercier Dieu le tout puissant et miséricordieux, qui nous a donné la force et la patience d'accomplir ce modeste travail.

Nous présentons nos plus vifs remerciements à notre promoteur Mr OUANES Mohand qui nous a encadré et dirigé tout au long de ce travail.

Nos sincères remerciements vont aux membres de jury pour l'intérét qu'ils ont porté à notre tavail en acceptant de l'examiner et de l'enrichir par leurs propositions et remarques.

Nous remercions également Mme FAHEM Karima pour son aide et son soutien.

En outre, nous présentons nos remerciements à tous nos enseignants(es) tout au long de notre cursus universitaire.

Dédicaces

Je tiens très respectueusement à dédier ce modeste travail à :

- *Mes très chers parents qui m'ont donné un honorable modèle de labeur et de persévérance, qu'ils trouvent ici l'expression de toute ma profonde reconnaissance et ma parfaite considération.*
- *Ma très chère tante ALDJIA que dieu lui accord une longue vie.*
- *Ma soeur unique LYNDA, que j'aime trop.*
- *Ma chère nièce Elena, que j'adore énormément.*
- *Ma petite et grande famille.*
- *Je désire le dédier à tous mes amis(es) et camarades sans exception .*

Lisa

Je tiens très respectueusement à dédier ce modeste travail :

- *A mon cher mari, son soutien, son amour, sa générosité, son encouragement n'ont fait qu'éclairer mon cœur et mon esprit.*
- *A mon père, ma mère, mon frère et mon grand-père qui m'ont toujours permis de voir plus claire aux moments les plus sombres.*
- *A la mémoire de ma grand-mère que dieu te garde dans son vaste paradis.*
- *A ma belle-famille qui m'ont toujours encouragé.*
- *A mon petit cousins YANI à qui je souhaite un bon rétablissement .*
- *A mon petit cousin ANYR que dieu le protège.*
- *Je désire le dédier à tous mes amis(es) et camarades sans exception .*

Katia

Table des matières

Table des figures

Liste des abréviations

RO : *Recherche Opérationnelle.*

PL : *Programmation Linéaire.*

SR : *Solutions Réalisables.*

PLNE : *Programmation (programme) Linéaire en Nombres Entiers.*

PLME : *Programmation Linéaire Mixte Entière.*

PLVM : *Programmation (programme) Linéaire à variables mixte .*

PLVB : *Programmation (programme) linéaire à Variables Booléennes.*

ONLE : *Optimisation Linéaire en Nombres Entiers.*

ILP : *Integer Linear Programming.*

B&B : *Branch and Bound.*

Introduction générale

Depuis l'apparition de la révolution technologique, le monde a connu une croissance remarquable dans la taille des organisations. Les petits magasins d'artisanat à l'époque ont évolué pour devenir des grandes sociétés aujourd'hui. Cependant, ce développement a engendré de nouveaux problèmes qui se posent encore à présent dans de nombreuses sociétés. Vu que les problèmes rencontrés doivent être résolus mais aussi rassurer le suivi de l'entreprise d'une meilleure façon, cela a donné un environnement propice à l'apparition de la recherche opérationnelle(RO).

Une des caractéristiques de la recherche opérationnelle, est qu'elle tente souvent de rechercher la meilleure solution (appelée solution optimale) pour le modèle représentant le problème auquel on est confronté. Cette « recherche d'optimalité » est un thème important dans la RO. Ce processus est résumé en un terme dit « optimisation », si cette dernière est linéaire, il s'agit alors de la programmation linéaire(PL). Celle-ci signifie que toutes les fonctions mathématiques de ce modèle doivent être des fonctions linéaires. Le mot programmation ici est synonyme de planification. Ainsi, la programmation linéaire implique la planification d'activités pour obtenir un résultat optimal, c'est-à-dire un résultat qui atteint le mieux le but spécifié parmi toutes les alternatives réalisables.

La programmation linéaire (PL) a été développée en 1947 par G.B.Dantzing et L.Kantorovich, pour répondre à des problèmes complexes que les méthodes classiques n'arrivaient pas à résoudre.

La programmation linéaire est un outil auquel on fait souvent appel dans de nombreux problèmes théoriques ou pratiques, il suppose que la solution doit être représentée en variables réelles. S'il nécessite de trouver des variables discrètes alors on parle de la programmation linéaire en nombres entiers (PLNE).

La PLNE n'est autre que étudié les programmes linéaires dont les variables sont en-

tières, en particulier les variables peuvent être simplement booléennes {0,1}.On dit que les variables sont soumises à des contraintes d'intégrité.

Il apparait que les problèmes linéaires en nombres entiers soient relativement difficiles à résoudre. Après tout, les problèmes de programmation linéaire peuvent être résolus. De manière extrêmement efficace, la seule différence est que les problèmes linéaires en nombres entiers ont beaucoup moins de solutions à considérer.

Dans notre travail, nous nous intéressons à la résolution des problèmes de programmation linéaire en nombres entiers par la méthode : « Séparation et Évaluation » ou « Branch and Bound ».
La méthode de Branch and Bound consiste à énumérer ces solutions d'une manière intelligente, cette technique arrive à éliminer les solutions partielles qui ne mènent pas à la solution que l'on cherche.

Notre travail s'articule autour de 4 chapitres :

Le premier chapitre présente quelques généralités sur la programmation mathématique et programmation linéaire, et quelques méthodes de résolutions de problèmes d'optimisation : méthode graphique, méthode du simplexe et la méthode dual du simplexe, et nous avons illuster un exemple pour chaque méthode.

Dans le deuxième chapitre, on se focalise sur la présentation de la programmation linéaire en nombre entiers avec quelques problèmes pratiques tel que le problème du sac à dos, et le problème de transport. Ensuite, on a introduit la notion de complexité. On a representé aussi deux méthodes de résolution : la méthode graphique et la méthode des toncatures (ou Gomory).

Le troisième chapitre est consacré à la méthode de Branch and Bound. Nous avons présenté le principe de la méthode et ses trois axes principaux : l'évaluation,la séparation et la stratégie de parcours, ainsi son algorithme. En particulier, l'algorithme de « branch and bound » pour le cas binaire {0,1}. A la fin du chapitre, nous avons illuster le problème du voyageur de commerce.

Et enfin dans le quatrième chapitre c'est une simulation numérique des trois exemples représentés aux chapitres 2 et 3 : problème du sac à dos, problème du transport et le problème de voyaguer de commerce, en utilisant le LpSolve IDE.

Chapitre 1

Rappels sur la programmation linéaire

Inroduction

Bien qu'on puisse modéliser des problèmes d'optimisation et utiliser des logiciels de programmation linéaire sans connaitre la théorie qui se cache derrière, des notions sont utiles pour mieux comprendre le sujet. Ce chapitre présente des bases de la programmation linéaire.

La programmation linéaire est un outil très puissant de la recherche opérationnelle. C'est un outil générique qui peut résoudre un grand nombre de problèmes. En effet, une fois un problème est modélisé sous la forme d'équations ou d'inéquations linéaires, des méthodes assurent la résolution du problème de manière exacte.

L'objectif de la programmation linéaire (PL) est de trouver la valeur optimale d'une fonction linéaire sous un système d'équations, d'inégalités de contraintes linéaires. La fonction à optimiser est dite « fonction économique »ou « fonction objectif ». On résout les problèmes de programmation linéaire en utilisant la méthode « graphique »s'il s'agit d'un programme linéaire à deux variables au plus trois variables ou en utilisant des méthodes « numériques ».

1.1 Notions de base

1.1.1 Combinaison linéaire convexe

Soient v_1,v_2,...,v_p des éléments d'un espace vectoriel, on appelle combinaison linéaire convexe de ces vecteurs le vecteur $V = \alpha_1 v_1 + \alpha_2 v_2 + ... + \alpha_p v_p$.

avec $\begin{cases} \alpha_i \geq 0 \\ \sum_{i=0}^{p} \alpha_i = 1 \end{cases}$

1.1.2 Ensemble convexe

Une partie $\boldsymbol{C}$ *de l'espace est convexe, si étant donné deux vecteurs* v_1 *et* v_2 *appartiennent à* $\boldsymbol{C}$*, la combinaison linéaire convexe* $V = \alpha_1 v_1 + \alpha_2 v_2$ *appartient aussi à* $\boldsymbol{C}$*.*

avec $\begin{cases} \alpha_1 \geq 0, \alpha_2 \geq 0 \\ \alpha_1 + \alpha_2 = 1 \end{cases}$

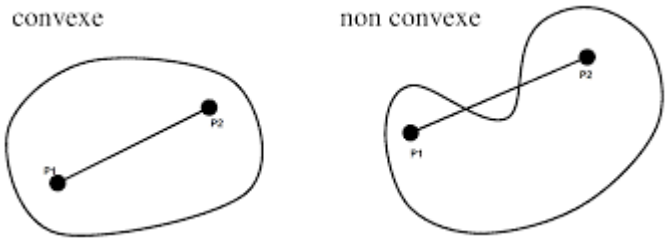


FIGURE 1.1 – Ensemble convexe

1.1.3 Point extrême

$\boldsymbol{V} \in \boldsymbol{C}$ *est dit point extrême si* $\boldsymbol{V}$ *ne peut pas être exprimé comme combinaison linéaire convexe de deux points de* $\boldsymbol{C}$*.*

1.1.4 Polyèdre et polytope

- *Hyperplan est défini par* $\boldsymbol{H}$= $\{x \in \mathbb{R}^N / a^T x = b\}$

- *Demi-espaces fermés :*
  $\boldsymbol{H}_+ = \{x \in \mathbb{R}^N / a^T x \geq b\}$
  $\boldsymbol{H}_- = \{x \in \mathbb{R}^N / a^T x \leq b\}$

Un polyèdre convexe *: est l'intersection d'un nombre fini de demi-espaces fermés de* $\mathbb{R}^N$*.*

Un polytope *: est un polyèdre convexe et borné.*

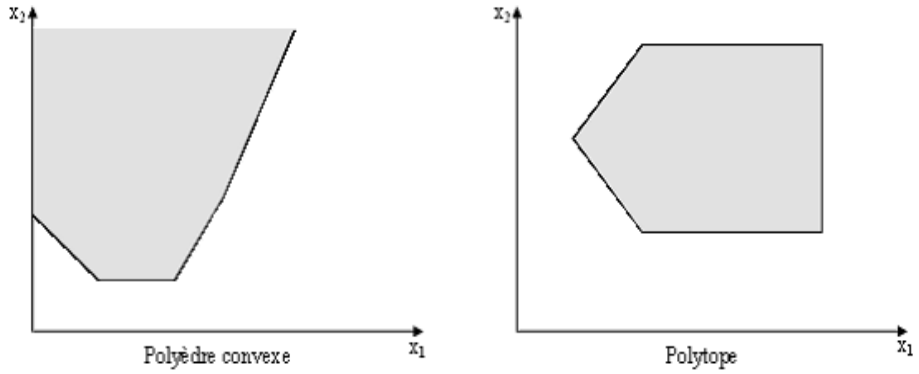


FIGURE 1.2 – Polyèdre et polytope

1.2 Programme linéaire

1.2.1 Forme générale d'un programme linéaire :

[1] Le modèle mathématique d'un PL peut se mettre sous la forme suivante :

$$\begin{cases} Max\ (ou\ \ min)\ \ Z = \sum_{j=1}^{n} c_j x_j \\ \sum_{i=1}^{m} a_{ij} x_j \leq, = ou \geq b_i \\ x_i \geq 0, \quad \forall j = 1, \ldots, n \end{cases}$$

Le programme comporte :

- *La fonction objectif Z à optimiser.*
- ***m** contraintes d'égalités ou d'inégalités (contraintes essentielles).*
- ***n** variables non négatives dites variables de décisions.*
- *Le coefficient de coût de la variable x_j est noté c_j et celui de la variable x_j dans la contrainte i est noté a_{ij}.*
- *La $i^{ème}$ contrainte a un second membre constant b_i.*
- *Les contraintes simples de positivité ne sont pas incluses dans les **m** contraintes.*

1.2.2 Forme standard d'un programme linéaire :

Comment passer de la forme canonique à la forme standard ?

- *On transforme une inéquation de signe ($\leq$) en une équation linéaire en additionnant une variable non négative dite **variable d'écart**.*
- *Pour transformer une inéquation de signe ($\geq$) en une équation linéaire, on doit soustraire une **variable d'écart**.*

Modèle de programmation linéaire dont les m contraintes essentielles sont du type ($\leq$) :

Forme canonique :

$$\begin{cases} Optimiser \quad Z = \sum_{j=1}^{n} c_j x_j \\ \sum_{i=1}^{m} a_{ij} x_j \leq b_i \\ x_i \geq 0, \quad \forall j = 1, \ldots, n \end{cases}$$

*Sa **forme standard** s'exprime :*

$$\begin{cases} Optimiser \quad Z = \sum_{j=1}^{n} c_j x_j \\ \sum_{i=1}^{m} a_{ij} x_j + x_{n+i} = b_i \\ x_i \geq 0, \quad \forall j = 1, \ldots, n+m \end{cases}$$

Comme nous le savons, si le modèle comporte des inéquations du type ($\geq$), il s'agit simplement de soustraire une variable d'écart dans chaque contrainte de ce type pour la transformer en équation. Comme nous le verrons après, on ne pourra toutefois débuter l'algorithme du simplexe avec cette structure (la solution de départ aura au moins une valeur négative pour une des variables). Alors il faudre augmenter le modèle en introduisant une variable artificielle dans chaque contrainte dont on a soustrait une variable d'ecart.

Remarque : *On peut toujours écrire un problème canonique sous forme standard et vice versa.*

Définitions :

- *On dit que $x_j \in \mathbb{R}^n$ est une **solution réalisable** si x_j satisfait aux contraintes essentielles et aux contraintes de positivité..*
- *Région des solutions réalisables **(SR)** : c'est l'ensemble des solutions réalisables.*
- *On dit que x_j est une **solution optimale** si x_j est réalisable et si de plus elle réalise le maximum ou le minimum de Z.*

Remarque : *Les problèmes de minimisation et de maximisation sont équivalents puisque :*
$MinZ = -Max(-Z)$
$MaxZ = -Min(-Z)$

1.3 Méthodes de résolutions :

1.3.1 Méthode graphique :

Elle est aussi appelée résolution géométrique, elle est utilisée généralement pour les PL sous forme canonique à deux variables.

Démarche à suivre pour la résolution graphique :

1. *On reporte sur un graphique chacune des contraintes du modèle et on détermine la région commune à l'ensemble de ces contraintes. Cette région si elle existe constitue la région des solutions réalisables.*
2. *On détermine les coordonnées des points extrêmes de la région des solutions réalisables* ***SR*** *en les localisant directement sur le graphique en résolvant les équations des droites qui se coupent.*
3. *On substitue ensuite les coordonnées de chaque point extrême dans l'expression de la fonction objectif. Le point extrême qui optimise la fonction objectif correspond à la solution optimale.*

Exemple :

Résoudre le programme linéaire suivant par la méthode graphique :

$$\begin{cases} Max & Z = 5x_1 + 3x_2 \\ S.c. & \\ 3x_1 + 5x_2 \leq 15 \\ 5x_1 + 2x_2 \geq 10 \\ x_1 \geq 0, x_2 \geq 0 \end{cases}$$

Sa représentation graphique est :

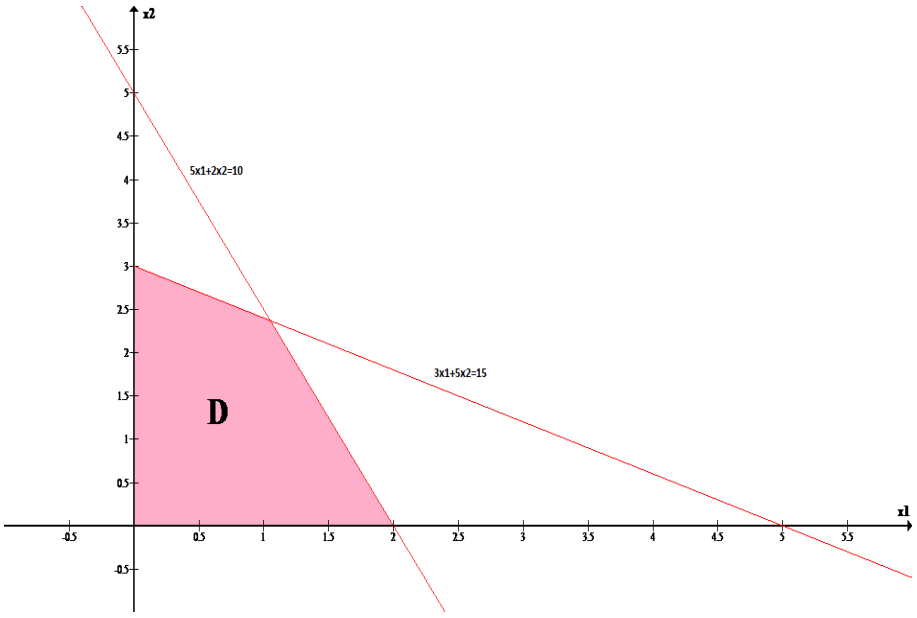


FIGURE 1.3 – Representation graphique des contraintes

Interpretation : ***D*** *représente la région des solutions admissibles .*

Les sommets de la région des solutions admissibles sont :
{(0,0) ; (2,0) ; (0,3) ; (60/57,45/19)}
On calcule la valeur de la fonction objectif pour chacun des sommets :
$Z(0,0) = 0$
$Z(2,0) = 10$
$Z(0,3) = 9$
$Z(60/57, 45/19) \simeq 12.37$
Puisque il s'agit de maximiser la fonction Z alors on prend le point qui donne la valeur maximale de Z qui est le point : (60/57 ,45/19) et la valeur optimale de Z est $Z^* \simeq 12.37$.

1.3.2 Méthode algébrique :

Nous avons présenté divers concepts associés à la programmation linéaire ainsi qu'une première méthode de résolution, soit la méthode graphique pour la résolution des PL à deux variables. Néanmoins, la plupart des situations présentent un nombre important de variables de décisions. Donc ils existent d'autres méthodes de résolution efficaces.

Pour ce faire, on utilise les méthodes suivantes :

Méthode du simplexe :

La méthode du simplexe est une méthode itérative, un procédé qui nous permet de passer d'un point extrême à un autre, de façon que la valeur de la fonctionnelle soit améliorée.

Cet algorithme s'applique lorsque le PL est sous la forme standard.[2]

Définitions :

Solution de base : *considérons un système Ax=b de m équations et n variables. Une solution est dite de base si n-m de ses composantes sont nulles, appelés variables hors bases. Les m autres variables non nulles sont appelés variables de base.*

Solution de base réalisable : *une solution de base est dite réalisable si elle satisfait les contraintes de non négativité.*

Propriétés du simplexe :

- *Chaque solution de base réalisable correspond à un point extrême.*

- *Si la solution optimale existe, il est obligatoirement localisé sur l'un des sommets du polyèdre de solution.*

- *Le nombre de points extrêmes de la région des solutions est fini.*

Principe de la méthode :

1. *Déterminer une première solution de base réalisable, cette solution initiale est le départ du cheminement vers la solution optimale (si elle existe).*
2. *Si la solution obtenue en 1 n'est pas optimale, déterminer une autre solution de base réalisable qui permettrait d'améliorer la fonction objectif (augmentation pour une maximisation, diminution pour une minimisation).*
3. *On répète cette procédure itérative jusqu'à ce que qu'il ne soit plus possible d'améliorer la fonction objectif. La dernière solution de base réalisable obtenue constitue la solution optimale au programme linéaire.*

Organigrame de la méthode du simplexe : Cas d'une maximisation

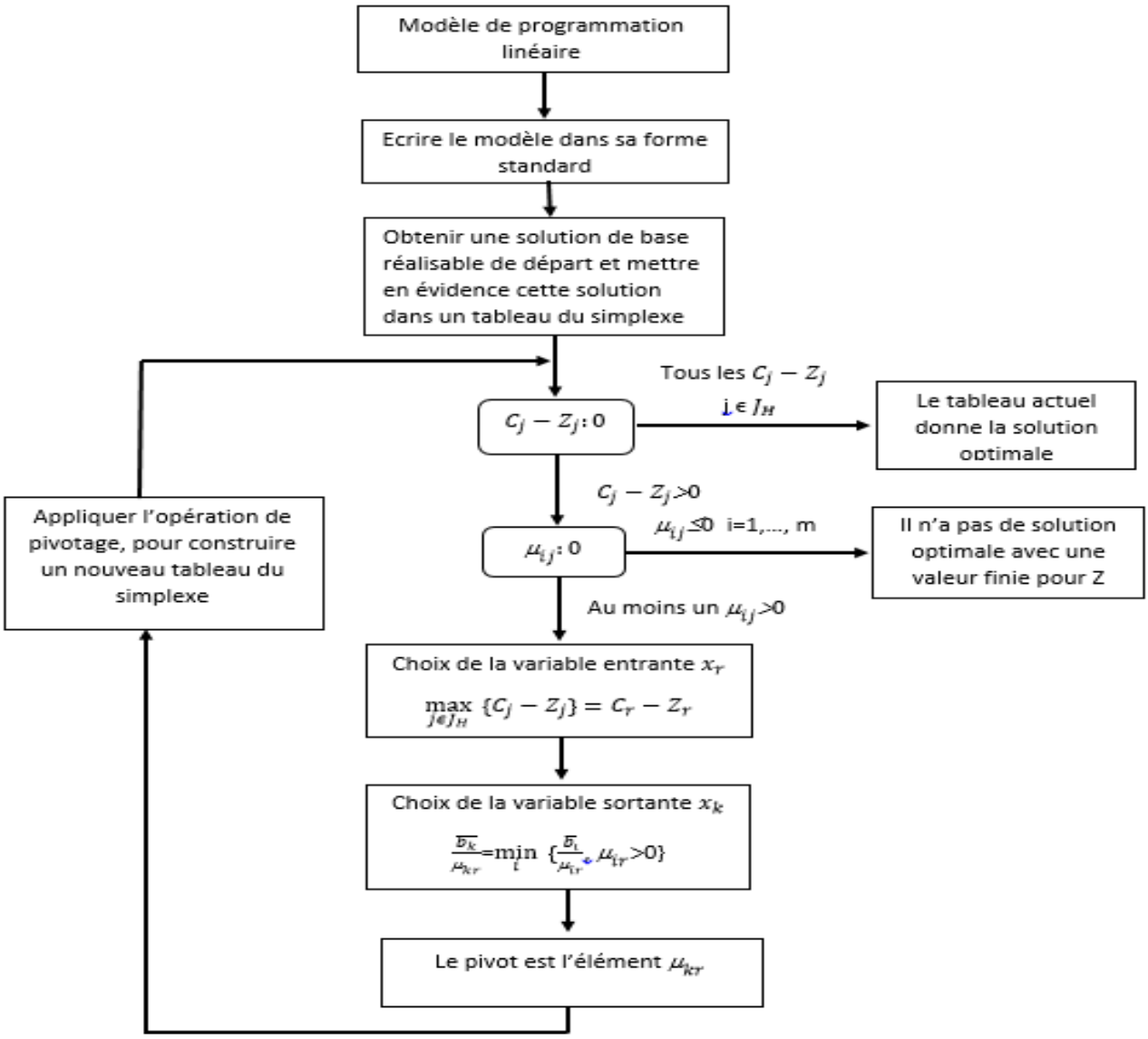


FIGURE 1.4 – Organigramme de la méthode du simplexe

Tableau du simplexe :

C_j		C_1	C_2	...	C_m	C_{m+1}	...	C_j	...	C_n	
C_B	*base*	x_1	x_2	...	x_m	x_{m+1}	...	x_j	...	x_n	b_i
C_1	x_1	*1*	*0*	...	*0*	$a_{1,m+1}$	...	a_j	...	a_n	b_1
C_2	x_2	*0*	*1*	...	*0*	$a_{2,m+1}$					b_2
$\vdots$	$\vdots$			...							$\vdots$
C_m	x_m	*0*	*0*	...	*1*	$a_{m,m+1}$	...	a_{mj}	...	a_{mn}	b_m
Z_j		Z_1	Z_2		Z_m	Z_{m+1}	...	Z_j	...	Z_n	Z
$C_j - Z_j$		$C_1 - Z_1$	$C_2 - Z_2$		$C_m - Z_m$	...	...	$Cj - Z_j$	...	$C_n - Z_n$	

Choix de la variable entrante :

Cas du max : *correspond à la variable de la plus grande valeur de* $C_j - Z_j$ *avec* $C_j - Z_j > 0$

Cas du min : *correspond à la variable de la plus petite valeur de* $C_j - Z_j$ *avec* $C_j - Z_j < 0$

Choix de la variable sortante :

Dans les deux cas (min ou max) elle correspond à la variable de la plus petite valeur de $\frac{\bar{b}_i}{\mu_{ir}}$ *avec* $\frac{\bar{b}_i}{\mu_{ir}} > 0$

Critère d'arrêt :

Nous arrêtons lorsque nous obtenons le critère d'optimalité. L'algorithme du simplexe s'arrête lorsque :

- $C_j - Z_j \leq 0, j \in J_H$ *: pour un problème de maximisation.*
- $C_j - Z_j \geq 0, j \in J_H$ *: pour un problème de minimisation.*

Exemple :

Détermination de la solution optimale à l'aide de l'algorithme du simplexe :

Soit le programme linéaire :

$$\begin{cases} MaxZ = 2x_1 + 2x_2 + 6x_3 \\ S.c. \\ x_1 + x_2 + 3x_3 \leq 48 \\ 4x_1 + x_2 + 6x_3 \leq 60 \\ x_j \geq 0, j = \{1, 2, 3\} \end{cases}$$

En ajoutant les variable des écarts $\{x_4, x_5\}$*, on obtient :*

Forme Standard :

$$\begin{cases} MaxZ = 2x_1 + 2x_2 + 6x_3 \\ S.c. \\ x_1 + x_2 + 3x_3 + x_4 = 48 \\ 4x_1 + x_2 + 6x_3 + x_5 = 60 \\ x_j \geq 0, j = \{1, 2, 3, 4, 5\} \end{cases}$$

Tableau1 :

C_j		*2*	*2*	*6*	*0*	*0*	
C_B	*base*	x_1	x_2	x_3	x_4	x_5	*solution de base*
0	x_4	*1*	*1*	*3*	*1*	*0*	*48*
0	x_5	*4*	*1*	*6*	*0*	*1*	*60*
Z_j		*0*	*0*	*0*	*0*	*0*	*Z=0*
$C_j - Z_j$		*2*	*2*	*6*	*0*	*0*	

$\exists$ *des* $C_j - Z_j \geq 0$, $j \in J_H$, *le critère d'optimalité n'est pas vérifié :*

Choix de la varible entrante :

$$\max_{j \in J_H}\{C_j - Z_j, C_j - Z_j \geq 0\} = C_3 - Z_3 = 6$$

La colonne du pivot est 3, la variable entrante est : x_3

Choix de la varible sortante :

$$\min_i\{\frac{\bar{b}_i}{\mu_{i3}}, \frac{\bar{b}_i}{\mu_{i3}} > 0\} = min\{16, 10\} = 10$$

La ligne du pivot est 2, la variable sortante est : x_5

l'operation du pivotage conduit au tableau 2.

Tableau2 :

C_j		*2*	*2*	*6*	*0*	*0*	
C_B	*base*	x_1	x_2	x_3	x_4	x_5	b_i
0	x_4	*-1*	*1/2*	*0*	*1*	*-1/2*	*18*
6	x_3	*2/3*	*1/6*	*1*	*0*	*1/6*	*10*
Z_j		*4*	*1*	*6*	*0*	*1*	*Z=60*
$C_j - Z_j$		*-2*	*1*	*0*	*0*	*-1*	

$\exists \{C_j - Z_j \geq 0, j \in J_H\} = C_2 - Z_2$ *; le critère d'optimalité n'est pas vérifié :*

Choix de la varible entrante :

$\exists\,! \{C_j - Z_j \geq 0, j \in J_H\} = C_2 - Z_2 = 1$

La colonne du pivot est 2, la variable entrante est : x_2

Choix de la varible sortante :

$$\min_i\{\frac{\bar{b}_i}{\mu_{i2}}, \frac{\bar{b}_i}{\mu_{i2}} > 0\} = min\{36, 60\} = 36$$

La ligne du pivot est 1, la variable sortante est : x_4

l'operation du pivotage conduit au tableau 3.

***Tableau3** :*

C_j		*2*	*2*	*6*	*0*	*0*	
C_B	*base*	x_1	x_2	x_3	x_4	x_5	b_i
2	x_2	*-2*	*1*	*0*	*2*	*-1*	*36*
6	x_3	*1*	*0*	*1*	*-1/3*	*1/3*	*4*
Z_j		*2*	*2*	*6*	*2*	*0*	*Z=96*
$C_j - Z_j$		*0*	*0*	*0*	*-2*	*0*	

Du tableau 3,on a :

$C_j - Z_j \leq 0, j \in J_H$, *alors le critère d'optimalité est vérifié.*

***Donc** le tableau 3 est optimal.*

***D'OÙ** la solution optimale du problème est* $x^* = (0, 36, 4)$ *et* $Z^* = 96$.

Remarque :

Si dans le tableau optimal il existe $C_j - Z_j = 0$ *pour une variable hors base alors la solution n'est pas unique.*

La dualité [2] :

Un aspect important de la programmation linéaire est la notion de dualité, à tout programme linéaire appelé par convention PL Primal, on peut associer un autre PL appelé Dual.

Programme primal	***Programme dual***
Maximiser $\sum_{j=1}^{n} c_j x_j$	*Minimiser* $\sum_{i=1}^{m} b_i y_i$
$\sum_{j=1}^{n} a_{ij} x_j \leq b_i$	$\sum_{i=1}^{m} a_{ij} y_i \leq c_j$
$x_i \geq 0, j = 1, \ldots, n$	$y_j \geq 0, i = 1, \ldots, m$

Relation Primal-Dual :

Programme Primal ⟷ **Programme Dual**

(Ou programme dual) (Ou programme primal)

Maximiser Z (ou W)	Minimiser W(ou Z)
Nombre de contraintes → A la contrainte i → Contrainte de type ≤ → Contrainte de type ≥ → Contrainte de type =	Nombre de variables duales → Correspond à la variable y_i → $y_i \geq 0$ → $y_i \leq 0$ → y_i sans restriction de signe
Nombre de variables (de décision) → $x_j \geq 0$ → $x_j \leq 0$ → x_j sans restriction de signe	Nombre de contraintes → Contrainte de type ≥ → Contrainte de type ≤ → Contrainte de type =
Coefficients C_j dans la fonction objectif.	Second membre pour la $j^{ème}$ contrainte.
Second membre b_i de la $i^{ème}$ contrainte.	Coefficients de la variable y_i dans la fonction objectif.
Coefficient, dans la contrainte i, de la variable $x_j(a_{ij})$	Coefficient, dans la contrainte j, de la variable y_i (a_{ij})

Propriétés de la dualité :

- *Le dual du dual est le primal.*
- *La solution du dual est égale aux coûts réduits du primal, et vice versa.*
- *Etant donné un primal et son dual :*
  - *Soit les deux problèmes ont un optimum fini.*
  - *Soit ils sont tous les deux sans solution.*
  - *Soit l'un est sans solution et l'autre sans solution finie.*
- *Pour un primal en maximisation, le coût d'une solution réalisable est inférieur ou égal au coût d'une solution réalisable du dual. Ces coûts sont égaux si et seulement si les deux solutions sont optimales.*

Utilité de la dualité :

La dualité est très utilisée en PL, notamment pour faciliter les calculs. Pour un PL à dix variables et deux contraintes ne peut pas être résolu graphiquement tandis que cela ne pose pas de problème pour son dual à deux variables et dix contraintes.

Algorithme dual du simplexe :[2]

La méthode de dual du simplexe a été développée par C.E.Lemke. Le principe de cette méthode consiste à partir d'une solution non réalisable ($x_b \leq 0$) *obtenir une solution réalisable* ($x_b \geq 0$).

Les étapes à suivre pour appliquer l'algorithme dual du simplexe sont :

1. *Ecrire le problème sous la forme standard.*
2. *Multiplier par* (-1) *les contraintes dont on a soustrait une variable d'écart.*
3. *Obtenir une solution de base de départ*
   - *avec* $C_j - Z_j \leq 0$, *j=1,...,n : dans le cas d'une maximisation.*
   - *avec* $C_j - Z_j \geq 0$, *j=1,...,n : dans le cas d'une minimisation.*
4. ***Critère de sortie d'une variable :*** *la variable* x_r *sort de la base d'après* $x_r = minimum\ \{\ x_{bi},\ x_b < 0\ \}$
5. ***Critère d'entrée d'une variable :*** *la variable* x_k *est introduite dans la base d'après*

   (minimisation)
   $\frac{-(C_r - Z_r)}{\mu_{kr}} = \min_j \{ \frac{-(C_j - Z_j)}{\mu_{kj}}, \mu_{kj} < 0 \}$

(maximisation)
$\frac{C_r - Z_r}{\mu_{kr}} = \min_j \{\frac{C_j - Z_j}{\mu_{kj}}, \mu_{kj} < 0\}$

le pivot est μ_{kr}

6. *Pivoter sur* μ_{kr} *en utilisant les mêmes règles que l'algorithme du simplexe.*
7. ***Critère d'optimalité :*** *on obtient une solution de base réalisable optimale si tous les* $x_{br} \geq 0$ *et*
   - *(minimisation) tous les* $C_j - Z_j \geq 0$.
   - *(maximisation) tous les* $C_j - Z_j \leq 0$.
8. *Si la solution de base n'est pas réalisable, on poursuit l'algorithme (aller en 4).*

Exemple :
Détermination de la solution optimale à l'aide de l'algorithme dual du simplexe .
Soit le programme linéaire :

$$\begin{cases} Minimiser \quad Z = 90x_1 + 120x_2 + 180x_3 \\ S.c. \\ 2x_1 + x_2 + 4x_3 \geq 3 \\ 3x_1 + 2x_2 + x_3 \geq 4 \\ x_1 + 3x_2 - 3x_3 \geq 1 \\ x_j \geq 0, \forall j = \{1, 2, 3\} \end{cases}$$

Soustrayant les variables d'écart et multipliant par (-1) les contraintes, on obtient :

$$\begin{cases} Minimiser \quad Z = 90x_1 + 120x_2 + 180x_3 + 0x_4 + 0x_5 + 0x_6 \\ S.c. \\ -2x_1 - x_2 - 4x_3 + x_4 = -3 \\ -3x_1 - 2x_2 - x_3 + x_5 = -4 \\ -x_1 - 3x_2 + 3x_3 + x_6 = -1 \\ x_j \geq 0, \forall j = \{1, 2, 3, 4, 5, 6\} \end{cases}$$

Tableau 1 :

C_j		*90*	*120*	*180*	*0*	*0*	*0*	
C_B	*base*	x_1	x_2	x_3	x_4	x_5	x_6	*solution de base*
0	x_4	*-2*	*-1*	*-4*	*1*	*0*	*0*	*-3*
0	x_5	*-3*	*-2*	*-1*	*0*	*1*	*0*	*-4*
0	x_6	*-1*	*-3*	*3*	*0*	*0*	*1*	*-1*
Z_j		*0*	*0*	*0*	*0*	*0*	*0*	*Z=0*
$C_j - Z_j$		*90*	*120*	*180*	*0*	*0*	*0*	

Variable sortante :

$x_k = min\{x_{bi}, x_{bi} < 0\} = min\{-3, -4, -1\} = -4 = x_5$

La ligne pivot est k=2 et la variable sortante est x_5.

Variable entrante :

$\frac{-(C_r - Z_r)}{\mu_{2r}} = \min_j\{\frac{-(C_j - Z_j)}{\mu_{2j}}, \mu_{2j} < 0\} = min\{\frac{-90}{-3}, \frac{-120}{-2}, \frac{-180}{-1}\} = min\{30, 60, 180\} = 30.$

La colonne pivot est r=1 et la variable entrante est x_1*.Le pivot est* $\mu_{21} = -3$.

L'operation de pivotage conduit au tableau 2.

Tableau 2 :

C_j		*90*	*120*	*180*	*0*	*0*	*0*	
C_B	*base*	x_1	x_2	x_3	x_4	x_5	x_6	b_i
0	x_4	*0*	*1/3*	*-10/3*	*1*	*-2/3*	*0*	*-1/3*
90	x_1	*1*	*2/3*	*1/3*	*0*	*-1/3*	*0*	*4/3*
0	x_6	*0*	*-7/3*	*10/3*	*0*	*-1/3*	*1*	*1/3*
Z_j		*90*	*60*	*30*	*0*	*-30*	*0*	*Z=120*
$C_j - Z_j$		*0*	*60*	*150*	*0*	*30*	*0*	

Variable sortante : x_4 *(ligne pivot : k=1)*

Variable entrante :

$\frac{-(C_r - Z_r)}{\mu_{1r}} = \min_j\{\frac{-(C_j - Z_j)}{\mu_{1j}}, \mu_{1j} < 0\} = min\{\frac{-150}{-10/3}, \frac{-30}{-2/3}\} = min\{45, 45\} = 45.$

Le minimum n'est pas unique. Choisissons j=3.

La variable entrante est x_3 *et la colonne pivot r=3.le pivot est* $\mu_{13} = -10/3$.

En pivotant sur -10/3, on obtient le tableau 3 :

Tableau 3 :

C_j		*90*	*120*	*180*	*0*	*0*	*0*	
C_B	*base*	x_1	x_2	x_3	x_4	x_5	x_6	b_i
180	x_3	*0*	*-1/10*	*1*	*-3/10*	*1/5*	*0*	*1/10*
90	x_1	*1*	*7/10*	*0*	*1/10*	*-2/5*	*0*	*13/10*
0	x_6	*0*	*-2*	*0*	*0*	*-1*	*1*	*0*
Z_j		*0*	*45*	*180*	*-45*	*0*	*0*	*Z=135*
$C_j - Z_j$		*0*	*75*	*0*	*45*	*0*	*0*	

Puisque $x_{bi} \geq 0$ *pour tout i et que* $c_j - z_j \geq 0$ *(critère d'optimalité pour une minimisation), le tableau 3 est optimal.*

La solution optimale du problème est :
$x_1 = 13/10$, $x_2 = 0$, $x_3 = 1/10$, $x_4 = 0$, $x_5 = 0$, $x_6 = 0$ $Z^* = 135$

Conclusion :

Dans ce chapitre nous avons introduit la programmation linéaire en donnant ses caractéristiques. L'optimisation d'un problème de programmation linéaire nécessite l'utilisation d'une méthode de résolution. Nous avons présenté deux classes de résolution : la résolution graphique et la résolution algébrique. Pour la résolution algébrique on a présenté deux algorithmes importants qui sont : l'algorithme du simplexe, et l'algorithme dual du simplexe.

Chapitre 2

Présentation de la programmation linéaire en nombres entiers

Inroduction

Dans beaucoup de problèmes d'optimisation une solution à valeurs entières est exigée. Par exemple dans le problème de production on cherche le nombre optimal (entier) de pièces à fabrique, dans ce cas on parle d'un problème de programmation linéaire en nombres entiers (PLNE).

La programmation linéaire en nombres entiers est un outil puissant de modélisation. En effet, de nombreux problèmes d'optimisation peuvent être formulés comme PLNE particuliers.

Dans ce chapitre nous introduisons les notions de programmation linéaire en nombres entiers que nous utilisons dans la suite du document. Nous commençons par rappeler les bases de la PLNE. Ensuite, nous abordons quelques méthodes de résolution des problèmes de la PLNE telle que la méthode graphique, et la méthode de Gomory.

Les programmes linéaires obtenus sont souvent difficiles à résoudre. Malheureusement, la majorité des algorithmes connus pour résoudre les problèmes de PLNE sont exponentiels car ils consistent à énumérer un grand nombre de solutions. Maintenant il existe un algorithme qui évite ce grand nombre d'énumérations qui est la méthode de « Branch and Bound »dont on va parler dans le chapitre 3.

2.1 Principe de la programmation linéaire en nombres entiers :

2.1.1 Définition de la PLNE :

L'optimisation linéaire en nombres entiers (OLNE) ou La programmation linéaire en nombres entiers (PLNE) ou integer linear programming (ILP) concerne les programmes d'optimisation sous contraintes pour lesquels les variables doivent prendre des valeurs entières. Elle est un domaine très riche de la programmation mathématique et de l'informatique théorique. Les recherches dans ce domaine sont nombreuses et les commencements étaient avec Gomory en 1955.

2.1.2 Avantages de la PLNE :

[3]

- *On peut modéliser plus de problèmes comme PLNE que comme programme linéaire (PL).*
- *Il y a des méthodes pour résoudre les PLNE, qui sont efficaces en pratique.*
- *Il y a des solveurs qu'on peut utiliser pour appliquer ces méthodes rapidement.*

2.1.3 Formulation d'un problème de PLNE :

[2] La présence de variables entières dans un PL modifie profondément la nature des programmes sous-jacents. Lorsqu'un problème est linéaire avec des variables entières nous parlons de la programmation mixte entiere (PLME). Si toutes les variables sont entières, nous utilisons la terminologie de la programmation (pure) en nombres entiers. Si les valeurs entières sont à valeurs 0 et 1 (binaires) nous parlons de programmation 0-1 (binaire)(PLVB).

La forme générale d'un programme linéaire en nombres entiers se présente comme suit :

Maximiser (ou minimiser) $\quad Z = c_1x_1 + c_2x_2 + \ldots + c_nx_n$

S.c

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n(\leq, =, \geq)b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n(\leq, =, \geq)b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n(\leq, =, \geq)b_m \\ x_j \geq 0, \forall j = 1, \cdots, n \\ x_j entier, j = 1, \cdots, k(k \leq n) \end{cases}$$

2.2 Deux problèmes pratiques sur la PLNE :

2.2.1 Problème du sac à dos :

[4] Le problème du sac à dos fait partie des problèmes d'optimisation les plus étudiés ces longues dernières années en raison de ses nombreuses applications dans le monde réel. Il modélise toute situation analogue au remplissage d'un sac à dos, ne pouvant supporter plus d'un certain poids, avec tout ou partie d'un ensemble donné d'objets ayant chacun un poids et une valeur et tels que les objets mis dans le sac à dos doivent, sans dépasser le poids maximum, maximiser la valeur totale.
C'est un problème linéaire en 0-1 ou encore un problème en variables bivalentes, c'est-à-dire, chacune des variables de décision du problème ne peut prendre comme valeur que 0 ou 1. Le problème du sac à dos intervient souvent comme sous-problème dans plusieurs domaines : le chargement d'avions ou de bateaux par exemple dans le domaine de la logistique, la gestion de portefeuilles dans le domaine de l'économie ou encore la découpe de matériaux dans le domaine de l'industrie. Vu l'intérêt que les problèmes de sac à dos suscitent, il semble intéressant d'en prendre un comme exemple d'application.

Enoncé du problème :

Soit à remplir un sac ne pouvant supporter que 14kg, par un deux ou trois objets parmi les trois objets O_1, O_2, O_3 de poids respectifs 4kg, 6kg, 8kg et de valeurs respectives 6, 8, 7, tout en maximisant la valeur totale des objets placés dans le sac. Soit x_i la variable égale au nombre d'objet O_i placé dans le sac, $1 \leq i \leq 3$.

Modélisation du problème :

La fonction objectif à maximiser est :
$Z = 6x_1 + 8x_2 + 7x_3$
Les variables x_i doivent évidemment être des entiers,$\forall\ i, x_i \in \mathbb{N}$.
Les contraintes du problème peuvent être représentées par le système d'inéquations suivant :

$$\begin{cases} 4x_1 + 6x_2 + 8x_3 \leq 14 \\ \forall i, x_i \in \{0, 1\} \end{cases}$$

Résoudre ce problème revient donc à résoudre le programme linéaire en nombres entiers (P) suivant :

$$\begin{cases} MaxZ = 6x_1 + 8x_2 + 7x_3 \\ 4x_1 + 6x_2 + 8x_3 \leq 14 \\ x_1 \leq 1 \\ x_2 \leq 1 \\ x_3 \leq 1 \\ \forall i, x_i \in \mathbb{N} \end{cases}$$

2.2.2 Problème de transport :

Description :

Dans un problème de transport, nous avons certaines origines, ce qui peut représenter les usines où nous avons produit des articles et fourni une quantité requise de produits à un certain nombre de destinations. Cela doit être fait de manière à maximiser le profit ou à minimiser le coût, Ainsi, nous avons les lieux de production comme origines et les lieux d'approvisionnement en tant que destinations.

Caractéristiques :

- *Un problème de transport est un problème de minimisation du coût de transport.*
- *Le problème de transport a pour but d'acheminer aux moindres coûts des marchandises depuis m origines vers n destinations.*

Enoncé du problème :

[5]

- *Une firme automobile a trois usines à Los Angeles, Detroit et New Orléans, et deux centres de distribution à Denver et Miami.*
- *Les capacités des trois usines sont de 1000, 1500 et 1200 respectivement, et les demandes aux centres de distribution sont de 2300 et 1400 voitures.*
- *<u>Coûts</u> :*

	Denver	*Miami*
Los Angeles	*80*	*215*
Detroit	*100*	*108*
New Orléans	*102*	*68*

Représentation offre et demande en tableau :

	Denver	*Miami*	*offre*
Los Angeles	*80* *1000*	*215* *0*	*1000*
Detroit	*100* *1300*	*108* *200*	*1500*
New Orléans	*102*	*68* *1200*	*1200*
demande	*2300*	*1400*	

Modélisation du problème

Min $Z = 80x_{11} + 215x_{12} + 100x_{21} + 108x_{22} + 102x_{31} + 68x_{32}$

S.c

(Los Angeles) $x_{11} + x_{12} = 1000$

(Detroit) $x_{21} + x_{22} = 1500$

(New Orleans) $x_{31} + x_{32} = 1200$

(Denver) $x_{11} + x_{21} + x_{31} = 2300$

(Miami) $x_{12} + x_{22} + x_{32} = 1400$

Avec $x_{ij} \in \mathbb{N}$, $i=\{1,2,3\}$ *et* $j=\{1,2\}$

2.3 La relaxation linéaire :

La relaxation linéaire(ou continue) R_P *de P est obtenue par relâchement (enlèvement) des contraintes d'intégrité.*

Soit F(P) la région des solutions possibles de P :

On a $F(P) \subseteq F(R_P)$*.[3]*

2.3.1 Propriétés de La relaxation linéaire :

[5]

- *La valeur de la solution optimale du PL relaxé* (R_p) *est une borne supérieure de la valeur de la solution optimale du problème (P).*
- *La valeur d'une solution admissible de P fournit une borne inférieure sur la valeur de la solution optimale de P.*
- *Si la solution de* R_p *est entière (donc admissible pour P), elle est également la solution optimale de P.*

2.4 Complexité :

La théorie de complexité s'attache à classifier les problèmes selon leur difficulté. Un problème est facile s'il existe un algorithme efficace pour le résoudre. Un problème est difficile s'il appartient à la classe des problèmes NP-complets, pour lesquels il est probable de trouver un jour un algorithme efficace.[7]

La programmation linéaire en nombres entiers appartient à la classe des problèmes NP-complets. Aucun algorithme polynomial n'est donc connu pour sa résolution. Les algorithmes exponentiels comme Branch and Bound sont donc susceptibles de rencontrer des difficultés si la taille des problèmes est importante.[6]

2.5 Résolution des programmes linéaires en nombres entiers :

Différentes approches ont été étudiées pour la recherche de solutions optimales entières du PLNE.
Actuellement les plus efficaces sont basées sur des algorithmes de recherche arborescente par séparation et évaluation (Branch-and-Bound), des méthodes de coupes et des relaxations.

2.5.1 Résolution graphique :

Lorsque la contrainte d'intégralité sur les variables x est relâchée, l'ensemble S des solutions réalisables représente un polyèdre. Les contraintes sont représentées graphiquement par des demi-plans et leur intersection est un ensemble convexe. Les solutions, si elles existent, appartiennent à cet ensemble. L'idée est de chercher dans cet ensemble les valeurs des variables de décision qui minimisent ou maximisent la fonction objectif.

Soit le PLNE suivant à deux variables :

$$\begin{cases} MaxZ = 2x_1 + x_2 & \\ x_1 + x_2 \leq 5 & (1) \\ -x_1 + x_2 \leq 0 & (2) \\ 6x_1 + 2x_2 \leq 21 & (3) \\ x_1, x_2 \in \mathbb{N} & \end{cases}$$

Résolu graphiquement sur la figure ci-dessous :

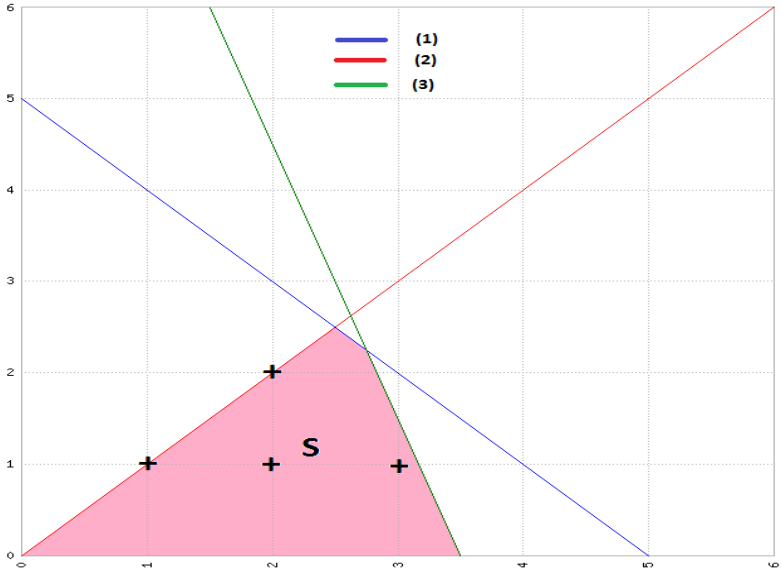


FIGURE 2.1 – Representation graphique des contraintes

Les + indiquent les points à coordonnées entières dans le domaine des solutions réalisables (S).

Si on relâche la contrainte d'intégrité sur les variables, on obtient :
La solution optimale du PL relaxé associé au PNLE est x= (11/4 ,9/4) avec un cout de Z=7.75.
Et pour le PLNE, l'optimum entier est x=(3,1) avec un coût de Z=7.*

Et de cet exemple, on déduit les remarques suivantes qui soulignent la difficulté de la PLNE [1] :

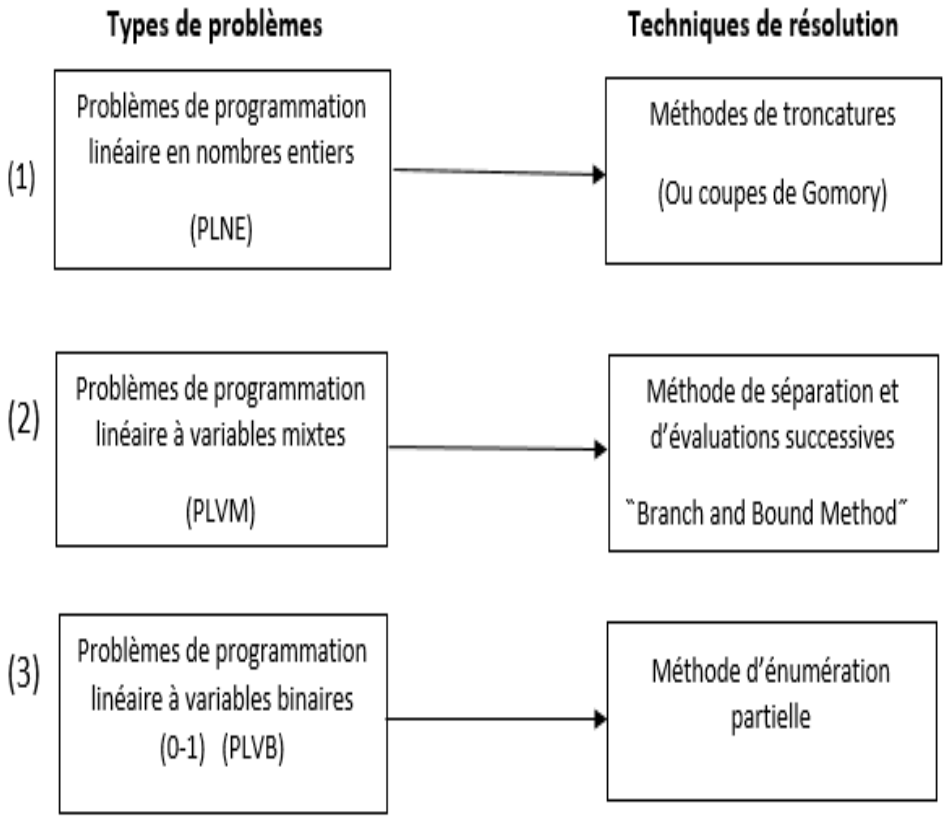
- *L'optimum entier n'est pas forcément un sommet du polyèdre, il peut être à l'intérieur du polyèdre.*
- *Si le PL relaxé avait un optimum entier, ce serait aussi l'optimum du PLNE.*
- *Sinon, le coût optimal du PL relaxé donne une borne supérieure (une borne inférieure en minimisation) du coût optimal du PLNE.*
- *L'arrondi de la solution du PL relaxé (ici le point (2,2) réalisable) n'est pas nécessairement optimal pour le PLNE, il pourrait même ne pas être réalisable.*
- *Le polyèdre peut être non vide, mais n'admettre aucune solution entière.*
- *On peut construire des cas où les optima du PLNE et du PL relaxé sont aussi éloignés que l'on veut.*

Les méthodes pour la PL ordinaire ne marchent donc pas pour la PLNE puisqu'elles cherchent un sommet optimal du polyèdre qui, en général, n'a pas de coordonnées entières.

Puisque nos méthodes de résolution des programmes linéaires ne garantissent que la

non-négativité des solutions, alors que nous voulons également des solutions avec valeurs entières, il faut par conséquent faire appel à d'autres techniques de résolution.

Les principaux algorithmes de résolution se divisent en trois catégories [2] :



2.5.2 Méthode de Gomory :

La méthode de troncatures (on dit également algorithme du « plan sécant »ou « méthode de Gomory ») développé par Ralph E. Gomory (1958) fournit une méthode de résolution des problèmes de programmation linéaire en nombres entiers (PLNE).

Le principe de cette méthode est d'introduire de nouvelles contraintes linéaires au problème pour réduire le domaine de réalisabilité du problème relaxé.

La procédure consiste à résoudre une suite de problèmes relaxés jusqu'à ce qu'une solution optimale en nombres entiers soit obtenue.

Un problème de la suite est obtenu du précèdent en lui ajoutant une contrainte linéaire (coupe supplémentaire).

Description de la méthode [4] :

Après avoir résolu le programme linéaire relaxé à l'aide de l'algorithme du simplexe, une solution optimale est obtenue. Si elle est entière, c'est la solution optimale du programme linéaire en nombres entiers (P). Sinon, la résolution de (P) n'est pas encore achevée. Soit à noter x la solution optimale non entière, x* = (x_1^*,x_2^*,...,x_m^*,...,x_n^*).*

Sans perdre la généralité, supposons que les variables de base, à l'issue de la méthode du simplexe, soient les m premières variables c'est-à-dire x_1,x_2,...,x_m et que dans le tableau final, on ait, en faisant varier k de 1 à m, les équations de la forme $x_k+\sum_{p=m+1}^{m+n} \bar{a}_{kp}x_p = \bar{b}_k$. La solution optimale x est donc telle que $x_i^*=\bar{b}_i$ pour i variant de 1 à m et pour $x_l^*=0$ pour $l \geq m+1$. Pour tout réel x, [x] désignera le plus grand entier inférieur ou égal à x et $\{x\}$ désignera la partie décimale de x alors $x = \{x\} + [x]$. Les coupes de Gomory se déduisent des équations du tableau final comme suit : Pour k, tel que, $1 \leq k \leq m$,*

$$x_k + \sum_{p=m+1}^{m+n} \bar{a}_{kp}x_p = \bar{b}_k$$

Équivalent à :

$$x_k + \sum_{p=m+1}^{m+n} [\bar{a}_{kp}]x_p + \sum_{p=m+1}^{m+n} \{\bar{a}_{kp}\}x_p = [\bar{b}_k] + \{\bar{b}_k\}$$

Comme

$$\sum_{p=m+1}^{m+n} [\bar{a}_{kp}]x_p \leq \sum_{p=m+1}^{m+n} \bar{a}_{kp}x_p$$

Alors

$$x_k + \sum_{p=m+1}^{m+n} [\bar{a}_{kp}]x_p \leq x_p + \sum_{p=m+1}^{m+n} \bar{a}_{kp}x_p$$

Ou encore

$$x_k + \sum_{p=m+1}^{m+n} [\bar{a}_{kp}]x_p \leq \bar{b}_k$$

En tenant compte des contraintes d'intégrité et puisque $[\bar{b}_k] \leq \bar{b}_k$
On a :

$$x_p + \sum_{p=m+1}^{m+n} [\bar{a}_{kp}]x_p \leq [\bar{b}_k]$$

Ainsi

$x_p + \sum_{p=m+1}^{m+n} [\bar{a}_{kp}] x_p \geq \{\bar{b}_k\}$

Cette dernière inégalité obtenue pour un certain k est appelée coupe de Gomory. En faisant varier k de 1 à m, il y a donc au plus m coupes de Gomory.

Il faut par la suite résoudre à l'aide de la méthode du simplexe le nouveau programme linéaire obtenu en ajoutant au programme linéaire précédent ces coupes de Gomory. Ce mécanisme de rajout des coupes de Gomory ne s'arrête que lorsque la solution optimale obtenue est entière.

Exemple d'application :

Résolvant le problème du sac à dos déjà donné.

On a le problème est modélisé comme suit :

Soit d'abord à résoudre, par la méthode du simplexe, le programme linéaire relaxé (P) :

$$\begin{cases} MaxZ = 6x_1 + 8x_2 + 7x_3 \\ 4x_1 + 6x_2 + 8x_3 \leq 14 \\ x_1 \leq 1 \\ x_2 \leq 1 \\ x_3 \leq 1 \\ \forall i, x_i \geq 0 \end{cases}$$

Sa forme standard :

$$\begin{cases} MaxZ = 6x_1 + 8x_2 + 7x_3 \\ 4x_1 + 6x_2 + 8x_3 + x_4 = 14 \\ x_1 + x_5 = 1 \\ x_2 + x_6 = 1 \\ x_3 + x_7 = 1 \\ \forall i, x_i \geq 0 \end{cases}$$

Dans notre cas, la première phase c'est-à-dire la recherche de base réalisable n'est pas nécessaire puisque les variables d'écart x_4, x_5, x_6, x_7 constituent déjà une base réalisable. La résolution se fera sous forme de tableaux et les pivots seront encadrés.

Tableau 1 :

C_j		6	8	7	0	0	0	0	
C_B	base	x_1	x_2	x_3	x_4	x_5	x_6	x_7	solution de base
0	x_4	4	6	8	1	0	0	0	14
0	x_5	1	0	0	0	1	0	0	1
0	x_6	0	1	0	0	0	1	0	1
0	x_7	0	0	1	0	0	0	1	1
Z_j		0	0	0	0	0	0	0	Z=0
$C_j - Z_j$		6	8	7	0	0	0	0	

Tableau 2 :

C_j		6	8	7	0	0	0	0	
C_B	base	x_1	x_2	x_3	x_4	x_5	x_6	x_7	b_i
0	x_4	4	0	8	1	0	-6	0	8
0	x_5	1	0	0	0	1	0	0	1
8	x_2	0	1	0	0	0	1	0	1
0	x_7	0	0	1	0	0	0	1	1
Z_j		0	8	0	0	0	8	0	Z=8
$C_j - Z_j$		6	0	7	0	0	-8	0	

Tableau 3 :

C_j		6	8	7	0	0	0	0	
C_B	base	x_1	x_2	x_3	x_4	x_5	x_6	x_7	b_i
0	x_4	4	0	0	1	0	-6	-8	0
0	x_5	1	0	0	0	1	0	0	1
8	x_2	0	1	0	0	0	1	0	1
7	x_3	0	0	1	0	0	0	1	1
Z_j		0	8	7	0	0	8	7	Z=15
$C_j - Z_j$		6	0	0	0	0	-8	-7	

Tableau 4 :

C_j		6	8	7	0	0	0	0	
C_B	base	x_1	x_2	x_3	x_4	x_5	x_6	x_7	b_i
6	x_1	1	0	0	1/4	0	-3/2	-2	0
0	x_5	0	0	0	-1/4	1	3/2	2	1
8	x_2	0	1	0	0	0	1	0	1
7	x_3	0	0	1	0	0	0	1	1
Z_j		6	8	7	3/2	0	-1	-5	Z=15
$C_j - Z_j$		0	0	0	-3/2	0	1	-5	

Tableau 5 :

C_j		6	8	7	0	0	0	0	
C_B	base	x_1	x_2	x_3	x_4	x_5	x_6	x_7	b_i
6	x_1	1	0	0	0	1	-1/2	0	1
0	x_7	0	0	0	-1/8	1/2	3/4	1	1/2
8	x_2	0	1	0	0	0	1	0	1
7	x_3	0	0	1	1/8	-1/2	-3/2	0	1/2
Z_j		6	8	7	7/8	5/2	11/2	0	Z=35/2
$C_j - Z_j$		0	0	0	-7/8	-5/2	-11/2	0	

D'après ce tableau final qui sera noté (Tableau 5), $x_1 = 1$, $x_2 = 1$, $x_3 = 1/2$, $x_4 = x_5 = x_6 = 0$, $x_7 = 1/2$ *La solution optimale non entière du problème est donc* $x^* = (1, 1, 1/2)$ *et la valeur maximale que peut avoir la fonction objectif Z correspondant à cette solution optimale est donc z=35/2= 17,5.*

Recherche de la solution optimale de (P) par la méthode des coupes de Gomory : *Du tableau 5, on a le problème (T) suivant :*

$$\begin{cases} x_1 + x_5 = 1 \\ x_7 - 1/8x_4 + 1/2x_5 + 3/4x_6 = 1/2 \\ x_2 + x_6 = 1 \\ x_3 + 1/8x_4 - 1/2x_5 - 3/4x_6 = 1/2 \end{cases}$$

Les coupes de Gomory sont :

$$\begin{cases} \{-1/8\}x_4 + \{1/2\}x_5 + \{3/4\}x_6 \geq \{1/2\} \\ \{1/8\}x_4 + \{-1/2\}x_5 + \{-3/4\}x_6 \geq \{1/2\} \end{cases}$$

Ce qui implique :

$$\begin{cases} 7/8x_4 + 1/2x_5 + 3/4x_6 \geq 1 \\ 1/8x_4 + 1/2x_5 + 1/4x_6 \geq 1/2 \end{cases} \iff \begin{cases} 7x_4 + 4x_5 + 6x_6 \geq 4 \\ 4x_5 + 2x_6 \geq 4 \end{cases}$$

Par l'introduction de variables d'écart, ces coupes sont transformées en égalité :

$$\begin{cases} 7x_4 + 4x_5 + 6x_6 - x_8 = 4 \\ x_4 + 4x_5 + 2x_6 - x_9 = 4 \end{cases}$$

Il faut ensuite ajouter ces égalités à (T). Les variables d'écart introduites dans ces égalités sont précédées du signe moins, le tableau nouvellement obtenu ne sera donc pas directement réalisable, il faut alors passer par la phase de recherche d'une base réalisable. Pour cela, il faut introduire des variables artificielles au niveau de ces égalités. Soient y_0 *et* y_1 *ces variables artificielles, elles sont positives ou nulles.*

$$\begin{cases} 7x_4 + 4x_5 + 6x_6 - x_8 + y_0 = 4 \\ x_4 + 4x_5 + 2x_6 - x_9 + y_1 = 4 \end{cases} \iff \begin{cases} y_0 = -7x_4 - 4x_5 - 6x_6 + x_8 + 4 \\ y_1 = -x_4 - 4x_5 - 2x_6 + x_9 + 4 \end{cases}$$

Puis, résoudre le programme linéaire suivant :

$$\begin{cases} MaxW = -y_0 - y_1 = 8x_4 + 8x_5 + 8x_6 - x_8 - x_9 - 8 \\ x_1 + x_5 = 1 \\ -x_4 + 4x_5 + 6x_6 + 8x_7 = 4 \\ x_2 + x_6 = 1 \\ 8x_3 + x_4 - 4x_5 - 6x_6 = 4 \\ 7x_4 + 4x_5 + 6x_6 - x_8 + y_0 = 4 \\ x_4 + 4x_5 + 2x_6 - x_9 + y_1 = 4 \end{cases}$$

Équivalent à :

$$\begin{cases} MaxW = -y_0 - y_1 = 8x_4 + 8x_5 + 8x_6 - x_8 - x_9 - 8 \\ x_1 + x_5 = 1 \\ -1/8x4 + 1/2x5 + 3/4x6 + x_7 = 1/2 \\ x_2 + x_6 = 1 \\ x3_+1/8x_4 - 1/2x_5 - 3/4x_6 = 1/2 \\ 7x_4 + 4x_5 + 6x_6 - x_8 + y_0 = 4 \\ x_4 + 4x_5 + 2x_6 - x_9 + y_1 = 4 \end{cases}$$

Tableau 1 :

C_j		*0*	*0*	*0*	*8*	*8*	*8*	*0*	*-1*	*-1*	*0*	*0*	
C_B	*base*	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	y_0	y_1	b_i
0	x_1	*1*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*1*
0	x_7	*0*	*0*	*0*	*-1/8*	*1/2*	*3/4*	*1*	*0*	*0*	*0*	*0*	*1/2*
0	x_2	*0*	*1*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*1*
0	x_3	*0*	*0*	*1*	*1/8*	*-1/2*	*-3/4*	*0*	*0*	*0*	*0*	*0*	*1/2*
0	y_0	*0*	*0*	*0*	*7*	$\boxed{4}$	*6*	*0*	*-1*	*0*	*1*	*0*	*4*
0	y_1	*0*	*0*	*0*	*1*	*4*	*2*	*0*	*0*	*-1*	*0*	*1*	*4*
W_j		*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*W=0*
$C_j - W_j$		*0*	*0*	*0*	*8*	*8*	*8*	*0*	*-1*	*-1*	*0*	*0*	

Tableau 2 :

C_j		*0*	*0*	*0*	*8*	*8*	*8*	*0*	*-1*	*-1*	*0*	*0*	
C_B	*base*	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	y_0	y_1	b_i
0	x_1	*1*	*0*	*0*	*-7/4*	*0*	*-3/2*	*0*	*1/4*	*0*	*-1/4*	*0*	*0*
0	x_7	*0*	*0*	*0*	*-3/4*	*0*	*0*	*1*	*1/8*	*0*	*-1/8*	*0*	*0*
0	x_2	*0*	*1*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*1*
0	x_3	*0*	*0*	*1*	*1*	*0*	*0*	*0*	*-1/8*	*0*	*1/8*	*0*	*1*
0	x_5	*0*	*0*	*0*	*7/4*	*1*	*3/2*	*0*	*-1/4*	*0*	*1/4*	*0*	*1*
0	y_1	*0*	*0*	*0*	*-6*	*0*	*-4*	*0*	$\boxed{1}$	*-1*	*-1*	*1*	*0*
W_j		*0*	*0*	*0*	*0*	*14*	*8*	*12*	*0*	*-2*	*2*	*0*	*W=8*
$C_j - W_j$		*0*	*0*	*0*	*-6*	*0*	*-4*	*0*	*1*	*-1*	*-2*	*0*	

Tableau 3 :

C_j		0	0	0	8	8	8	0	-1	-1	0	0	
C_B	base	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	y_0	y_1	b_i
0	x_1	1	0	0	-1/4	0	-1/2	0	0	1/4	0	-1/4	0
0	x_7	0	0	0	0	0	1/2	1	0	1/8	0	-1/8	0
0	x_2	0	1	0	0	0	1	0	0	0	0	0	1
0	x_3	0	0	1	1/4	0	-1/2	0	0	-1/8	0	1/8	1
0	x_5	0	0	0	1/4	1	1/2	0	0	-1/4	0	1/4	1
0	x_8	0	0	0	-6	0	-4	0	1	-1	-1	1	0

Les variables artificielles sortent de base et sont donc de valeur nulle. Le tableau, une fois les colonnes des variables artificielles enlevées, est réalisable et l'algorithme du simplexe pourra se poursuivre. Il ne faut pas oublier de reprendre l'expression contenant Z dans (P), et de remplacer les variables de base par leurs nouvelles expressions données par le dernier tableau ci-dessus.

Tableau final :

C_j		6	8	7	0	0	0	0	0	0	
C_B	base	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	b_i
6	x_1	1	0	0	-1/4	0	-1/2	0	0	1/4	0
0	x_7	0	0	0	0	0	1/2	1	0	1/8	0
8	x_2	0	1	0	0	0	1	0	0	0	1
7	x_3	0	0	1	1/4	0	-1/2	0	0	-1/8	1
0	x_5	0	0	0	1/4	1	1/2	0	0	-1/4	1
0	x_8	0	0	0	-6	0	-4	0	1	-1	0
Z_j		6	8	7	1/4	0	3/2	0	0	5/8	Z=15
$C_j - Z_j$		0	0	0	-1/4	0	-3/2	0	0	-5/8	

$C_j - Z_j \leq 0$, l'algorithme du simplexe s'arrête. Et puisque la solution ici est entière on n'a pas besoin de poursuivre le mécanisme des coupes de Gomory.
Comme $x_1 = 0, x_2 = 1, x_3 = 1, x_5 = 1, x_4 = x_6 = x_7 = x_8 = x_9 = 0$. Et Z=15 c'est la valeur totale maximale possible des objets à placer dans le sac. L'optimum discret de notre problème est (0, 1, 1), il faut donc choisir de placer dans le sac les objets O_2 et O_3.

Conclusion :

Dans ce chapitre, nous avons présenté les concepts de base de la PLNE tel que la formulation, la relaxation linéaire et sa difficulté de résolution. Or résoudre un problème de PLNE est un problème Difficile.

Ensuite, nous avons vu comment résoudre un PLNE graphiquement et l'une des principales méthodes de résolution : coupes de Gomory. Cette dernière permet de couper le domaine de réalisabilité de sorte de s'approcher de la solution optimale entière.

L'inconvénient de la méthode de troncatures est qu'elle fournit souvent une solution réalisable uniquement à la fin. Contrairement à la méthode arborescente de « Branch and Bound », qui livre en cours de la recherche une suite de bonnes solutions. Ce qu'on verra dans le chapitre suivant.

Chapitre 3

Méthode de Branch and Bound

Inroduction

On a déjà vu une méthode de résolution des problèmes de PLNE en utilisant « les coupes de Gomory ». Les coupes attaquant le problème de l'extérieur en gardant le domaine réalisable convexe, ce qui peut être inefficace dans certaines configurations.

On peut résoudre aussi les problèmes de PLNE/PLVM en fractionnant le domaine en régions faisable bornées par des frontières alignées sur des entiers : on sépare ainsi les domaines, et on évalue quelle région explorer en premier : on appelle cette méthode, la méthode de « séparation-évaluation », ou « Branch-and-Bound »en anglais.

L'algorithme de Branch-and-Bound (B&B) (Land et Doig 1960), repose sur une méthode arborescente de recherche d'une solution optimale par séparation et évaluation, en représentant les états solutions par un arbre d'états, avec des nœuds, et des feuilles.

3.1 Principe de la méthode de Branch-and-Bound :

Pour plusieurs problèmes, en particuliers les problèmes d'optimisation, l'ensemble de leurs solutions est fini (en tout cas dénombrable). Il est donc possible en principe d'énumérer toutes ses solutions, et ensuite de prendre la meilleure. L'inconvénient majeur de cette approche est le nombre prohibitif du nombre de solutions : il n'est guère évident d'effectuer cette énumération.

La méthode (B&B) consiste à énumérer ces solutions d'une manière intelligente en ce sens que, en utilisant certaines propriétés du problème en question. Cette technique arrive à éliminer des solutions partielles qui ne mènent pas à la solution que l'on cherche. De ce fait, on arrive souvent à obtenir la solution recherchée en un temps raisonnable. Bien entendu, dans le pire cas, on retombe toujours sur l'élimination explicite de toutes les solutions du problème.

[8] La méthode B&B est basée sur trois axes principaux :

- *L'évaluation.*
- *La séparation.*
- *La stratégie de parcours.*

3.1.1 L'évaluation :

L'évaluation permet de réduire l'espace de recherche en éliminant quelques sous-ensembles qui ne contiennent pas la solution optimale. L'objectif est d'essayer d'évaluer l'intérêt de l'exploration d'un sous-ensemble de l'arborescence. Le Branch-and-Bound utilise une élimination de branches dans l'arborescence de recherche de la manière suivante : la recherche d'une solution de coût minimal, consiste à mémoriser la solution de plus bas coût rencontré pendant l'exploration, et à comparer le coût de chaque nœud parcouru à celui de la meilleure solution. Si le coût du nœud considéré est supérieur au meilleur coût, on arrête l'exploration de la branche et toutes les solutions de cette branche seront nécessairement de coût plus élevé que la meilleure solution déjà trouvée.

3.1.2 La séparation :

La séparation consiste à diviser le problème en sous-problèmes. Ainsi, en résolvant tous les sous-problèmes et en gardant la meilleure solution trouvée, on est assuré d'avoir résolu le problème initial. Cela revient à construire un arbre permettant d'énumérer toutes les solutions. L'ensemble des nœuds de l'arbre qu'il reste encore à parcourir comme étant susceptibles de contenir une solution optimale, c'est-à-dire encore à diviser, est appelé ensemble des nœuds actifs.

3.1.3 La stratégie de parcours :

- ***La largeur d'abord :*** *Cette stratégie favorise les sommets les plus proches de la racine en faisant moins de séparations du problème initial. Elle est moins efficace que les deux autres stratégies présentées.*

- ***La profondeur d'abord :*** *Cette stratégie avantage les sommets les plus éloignés de la racine (de profondeur la plus élevée) en appliquant plus de séparations au problème initial. Cette voie mène rapidement à une solution optimale en économisant la mémoire.*

- ***Le meilleur d'abord :*** *Cette stratégie consiste à explorer des sous-problèmes possédant la meilleure borne. Elle permet aussi d'éviter l'exploration de tous les sous-problèmes qui possèdent une mauvaise évaluation par rapport à la valeur optimale.*

[9] L'algorithme de B&B construit et parcourt l'arbre de recherche. Sa racine est le domaine réalisable du problème initial, du modèle PLNE, tandis que les nœuds représentent les domaines réalisables des sous-problèmes. À chaque itération, l'algorithme de B&B résout la relaxation du sous-problème courant. Cette dernière fournit une borne inférieure sur la valeur optimale du sous-problème courant. Différents cas peuvent être rencontrés :

- *La borne inférieure du nœud est supérieure à la meilleure solution réalisable obtenue par l'algorithme de B&B : on peut alors affirmer que la solution optimale globale ne peut pas être contenue dans le sous ensemble de solutions représenté par ce nœud. On l'élague.*
- *La solution de la relaxation continue du nœud devient la meilleure solution réalisable du problème initial si elle est entière, et que sa valeur optimale est inférieure à celle obtenue par l'algorithme de B&B jusqu'à présent.*
- *Finalement, si la borne inférieure obtenue est meilleure que la dernière valeur obtenue par l'algorithme de B&B, mais sa solution n'est pas entière, le problème sera divisé en sous-problèmes.*

La méthode sera appliquée récursivement au reste des nœuds, et elle s'arrête lorsqu'il n'y a plus de nœud à évaluer.

3.2 Algorithme de Branch-and-Bound (version de base) :

[7]

1. *Initialiser avec le calcul d'une solution admissible de valeur Z ou poser $Z = +\infty$.*
2. *Résoudre la relaxation continue (R_P) et mettre à jour éventuellement Z.*
3. *Appliquer un test d'élagage.*
4. ***Si** il reste des nœuds non élagués **alors***
5. *choisir un nœud non élagué et brancher sur une des variables.*
6. *Pour chacun des 2 nouveaux nœuds :*
7. *Résoudre la relaxation continue (R_P) et mettre à jour Z.*
8. *Revenir à l'étape 3.*

9. ***finSi***
10. *La solution courante Z est optimale.*

3.3 Branch-and-Bound : cas des variables binaire 0-1 :

[11] La méthode est particulièrement bien adaptée à la résolution de problèmes linéaires en variables booléennes : les inconnues ne peuvent prendre que les valeurs 0-1.

Un algorithme simple pour énumérer toutes les solutions d'un modèle 0-1 consiste à :

- *Choisir un sommet dans l'arbre des solutions ;*
- *Choisir une variable x non encore fixée relativement à ce sommet ;*
- *Générer les deux variables $x = 0$ et $x = 1$ (la variable x est dite fixée) : chaque alternative correspond à un sommet de l'arbre des solutions ;*
- *Recommencer à partir d'un sommet pour lequel certaines variables ne sont pas encore fixées.*

A la racine de l'arbre, aucune variable n'est encore fixée, tandis qu'aux feuilles de l'arbre, toutes les variables ont été fixées. Le nombre de feuilles est 2^n (pour n variables 0-1). Le calcul de borne (ou évaluation) consiste à résoudre la relaxation linéaire en chaque sommet. L'élagage (ou élimination) consiste à utiliser l'information tirée de la résolution de la relaxation linéaire pour éliminer toutes les solutions émanant du sommet courant.

Un sous-problème est elagué si une des trois conditions suivantes est satisfaite :

- *Test 1 : sa borne supérieure (valeur optimale de la relaxation PL) est inférieure ou égale à Z (valeur de la meilleure solution courante) ;*
- *Test 2 : sa relaxation PL n'a pas de solution réalisable ;*
- *Test 3 : la solution optimale de sa relaxation PL est entière.*

Lorsque le Test 3 est verifié, nous testons si la valeur optimale de la relaxation PL du sous-problème, Z_i, est supérieure à Z^*. Si $Z_i > Z^*$, alors $Z^* := Z_i$, et nous conservons la solution, qui devient la meilleure solution courante. En résumé, nous obtenons l'algorithme ci-dessous.

3.3.1 Algorithme de Branch-and-Bound : cas binaire :

[10]

1. *Initialisation :*

(a) *poser* $Z^* = \infty$ *;*

(b) *appliquer le calcul de borne et les critères d'élagage à la racine (aucune variable fixée).*

2. *Critère d'arrêt : s'il n'y a plus de sous-problèmes non élagués, arrêter.*
3. *Branchement :*
   (a) *parmi les sous-problèmes non encore élagués, choisir celui qui a été créé le plus récemment (s'il y a égalité, choisir celui de plus grande borne supérieure).*

   (b) *appliquer le Test 1 : si le sous-problème est élagué, retourner en 2.*

   (c) *brancher sur la prochaine variable non fixée.*
4. *Calcul de borne :*
   (a) *résoudre la relaxation PL de chaque sous-problème ;*

   (b) *arrondir la valeur optimale si tous les paramètres de l'objectif sont entiers.*
5. *Elagage : élaguer un sous-problème si*
   (a) *la borne supérieure est inférieure ou égale à* Z^* *;*

   (b) *la relaxation PL n'a pas de solution réalisable ;*

   (c) *la solution optimale de la relaxation PL est entière : si la borne supérieure est strictement supérieure à* Z^**,* Z^* *est mise à jour et la solution de la relaxation PL devient la meilleure solution courante*
6. *Retourner en 2.*

3.4 Problème du voyageur de commerce :

[11] Le problème du « voyageur de commerce », ou TSP (Traveling Salesman Problem), est le suivant : un représentant de commerce ayant n villes à visiter souhaite établir une tournée qui lui permette de passer exactement une fois par chaque ville et de revenir à son point de départ pour un moindre coût, c'est-à-dire en parcourant la plus petite distance possible. C'est un des problèmes les plus anciennement et largement étudiés en optimisation combinatoire.

Considérons le problème du voyageur de commerce afin d'introduire la méthode de Branch&Bound.

Considérons n + 1 villes $V_1, V_2, \ldots, V_n$.

Et les distances inter-villes d_{ij}, $i\,;\, j \in \{0, \ldots, n\}$.

Le problème du voyageur de commerce consiste à donner une tournée allant et revenant à la ville V_0 *en passant une fois par chacune des villes.*

Une solution de ce problème s'écrit donc sous la forme d'une liste de villes données dans l'ordre de la tournée, que l'on peut noter $\sigma(1), \ldots, \sigma(n)$ *ou est une permutation des n villes.*

On a donc clairement n ! solutions possibles. Nous avons vu que l'explosion combinatoire d'une énumération la rend difficilement envisageable dans tous les cas. Néanmoins, dans cette section, nous nous intéressons aux méthodes énumératives, c'est-à-dire des méthodes qui explorent l'espace des solutions pour trouver une solution de meilleur coût.

Les méthodes de Branch&Bound essayent d'éviter d'explorer entièrement l'espace des solutions en utilisant les caractéristiques des solutions optimales et des estimations des coûts des solutions. Malheureusement, ces améliorations ne changent pas la complexité de ces méthodes qui restent de complexité exponentielle. C'est pourquoi il est souvent nécessaire de chercher d'autres techniques de résolution. Ce que nous allons voir :

Prenons dans cet exemple 5 villes $V_0, V_1, V_2, V_3 et V_4$ *et les distances inter-villes données par le tableau suivant :*

	V_0	V_1	V_2	V_3	V_4
V_0	–	8	4	2	3
V_1	9	–	7	1	6
V_2	3	7	–	6	6
V_3	2	1	6	–	4
V_4	7	6	6	2	–

Pour construire une méthode énumérative, il faut choisir une façon d'énumérer les solutions de façon à n'en oublier aucune et en évitant de traiter plusieurs fois la même solution. Dans notre exemple, on peut considérer l'arborescence d'énumération suivante :

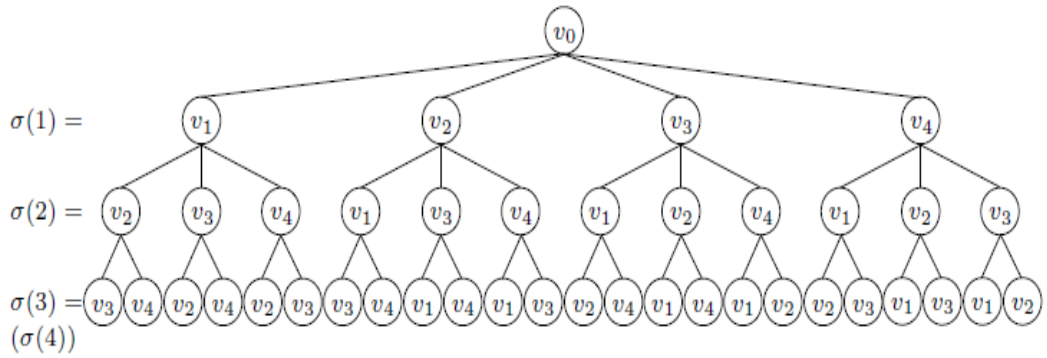


FIGURE 3.1 – Arborescence d'énumération

A chaque niveau de l'arborescence en partant de la racine, on fixe quelle ville sera visitée à la $i^{ème}$ place de la tournée ($i = 0$, puis $i = 1$, puis $i = 2$,...). Ainsi, chaque nœud correspond à une tournée partielle (et valide) des villes. Par exemple, le $4^{ème}$ nœud en partant de la gauche du niveau où l'on affecte la $2^{ème}$ place, correspond à la tournée partielle $\{V_0, V_2, *, *\}$. Au rang le plus bas, chaque feuille correspond à une tournée complète, c'est-à-dire à une solution du problème.

Notre but est donc de décider si l'exploration d'une branche est utile avant d'enénumérer toutes les solutions. Pour cela, on va donner une évaluation des solutions que peut porter une branche, c'est-à-dire donner une borne inférieure des coûts des solutions de la branche. Si cette évaluation est plus grande qu'une solution que l'on connait déjà, on peut élaguer cette branche, c'est-à-dire éviter de l'explorer.

Dans notre exemple du voyageur de commerce, avant de commencer l'énumération, tentons de trouver une bonne solution. Par exemple, on peut prendre une heuristique gloutonne qui consiste à prendre en premier la ville la plus près de V_0, puis en deuxième une ville (parmi les villes restantes) qui soit la plus près de la première, et ainsi de suite. On obtient ainsi une solution réalisable $S_1 = \{V_0, V_3, V_1, V_4, V_2, V_0\}$ de coût $c1 = 18$. Une solution réalisable nous donne une bonne supérieure pour notre problème, et nous savons que la solution optimale du problème est au plus de coût 18.

Il nous serait à présent utile de trouver une évaluation du coût de la solution par une borne inferieure.

En effet, si l'on pouvait par exemple déterminer que la solution optimale est au moins de coût 18, nous aurions déjà résolu notre problème. Une idée possible ici est de dire que l'on passe par toutes les villes et que l'on doit donc sortir de chacune des villes pour aller vers une autre ville. En d'autre terme, le trajet pour aller d'une ville V_i à une autre ville

sera au moins égale à la plus petite distance issue de V_i.

Ainsi, en sommant les coûts minimaux de chaque ligne du tableau, on obtient une valeur qui sera toujours inferieure au coût d'une solution du problème. Par exemple, ici on obtient une évaluation de 2+1+3+2+2=9. Une solution de ce problème sera donc au moins de coût 9. Malheureusement, la meilleure solution que nous connaissons est de coût 18, nous devons donc commencer l'énumération.

La première branche à gauche issue de la racine de l'arborescence contient toutes les solutions où V_i est affectée à la place 1.

Si l'on applique notre idée d'évaluation a la sous-matrice obtenue en ôtant la ligne correspondant à V_0 (on connait déjà la ville qui suit V_0) et la colonne correspondant à V_1 (la ville allant en V_1 est déjà fixée, c'est V_0), on obtient alors 8 pour la somme des coûts minimaux des lignes. En ajoutant le coût $C_{V_0V_1}$= 8, on sait alors que toute solution de cette branche a au moins un coût de 16.

On explore alors le nœud de l'arborescence commençant par $\{V_0, V_1, V_2\}$. Cela correspond à une matrice où l'on a ôté les lignes correspondant à V_0 et V_1 et les colonnes correspondant à V_1 et V_2. La somme des coûts minimaux des lignes restantes est alors de 7. En ajoutant alors le coût $C_{V_0V_1}$ + $C_{V_1V_2}$= 15, on sait que toute solution de cette branche est au moins de 22. Comme nous connaissons déjà une solution de coût 18, on peut donc éviter d'explorer (élaguer) cette branche.

La figure suivante résume l'exploration complète de notre arborescence par un parcours en profondeur. Les valeurs qui accompagnent les nœuds sont les évaluations des branches et celles qui accompagnent les feuilles sont les coûts des solutions associées.

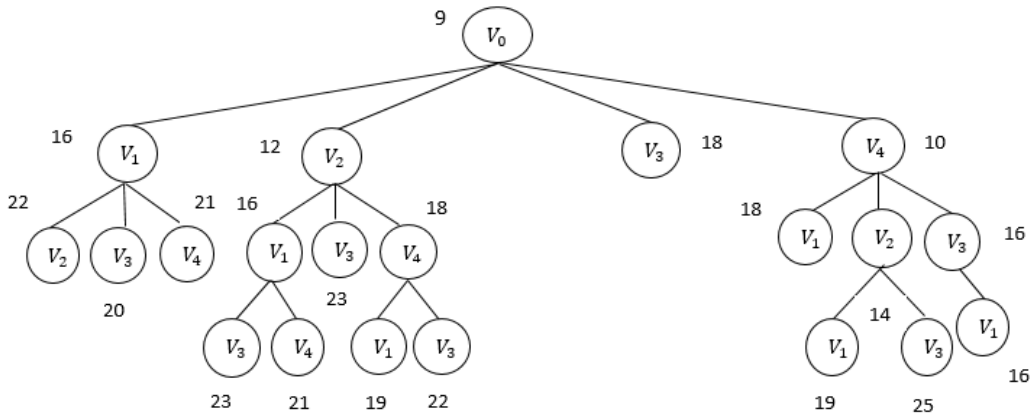


FIGURE 3.2 – Arborescence par parcours de profondeur

Lors de cette exploration, on a pu élaguer totalement la $3^{ème}$ *branche. C'est la dernière branche qui nous a conduit à une feuille correspondant à une solution* $S_2 = \{V_0, V_4, V_3, V_1, V_2\}$ *de coût 16. Or, la dernière branche ne peut pas mener à une solution de meilleur coût que 16, on a pas besoin d'explorer cette dernière branche. En conclusion, la meilleure solution pour le problème est de coût 16 et nous connaissons* S_2 *une des solutions optimales possibles.*

Conclusion

Dans ce chapitre nous nous sommes intéressés à la résolution d'un problème de PLNE par la méthode de Branch-and-Bound basée sur l'énumération intelligente des solutions.

En particulier, nous avons illustré un problème de PLVB, un des problèmes les plus connu qui est le problème du voyageur de commerce.

Chapitre 4

Simulation numérique

Inroduction

Lp solve est un logiciel libre de la Free Software Foundation (licence GNU lesser general public license). Lp solve est un logiciel de résolution de programmes linéaires et programmes linéaires en nombres entiers. Ce solveur est basé sur la méthode révisée du simplexe et une méthode de type Branch-and-Bound pour les PLNE.

LPSolve a au plus 4 arguments qui sont dans l'ordre :

- ***Obj** : expression symbolique représentant la fonction objectif,*
- ***constr** (optionnel) : liste des contraintes linéaires qui sont soit des équalités, soit des inéequalités, soit des expressions bornées définies par expr=a..b,*
- ***bd** (optionnel) : séquence d'expressions du type var=a..b specifiant que la variable var est comprise entre a et b,*
- ***opts** (optionnel) : séquence de paramètres option=value pour le solver, où option est choisi parmi assume, lp_maximize, lp_variables, lp_integervariables, lp_binaryvariables or lp_depthlimit.*

Résolvant les problèmes déjà données aux chapitres 2 et 3, en utilisant le solveur LP Solve IDE :

4.1 Problème du sac à dos :

Le problème est comme suit :

```
/* objective function */
max : 6x1+8x2+7x3 ;
```

/ Variable bounds */*

4x1 +6x2 + 8x3 <= 14 ;

x1 <=1 ;

x2 <=1 ;

x3 <=1 ;

*/*integer definitions */*

int x1, x2 , x3 ;

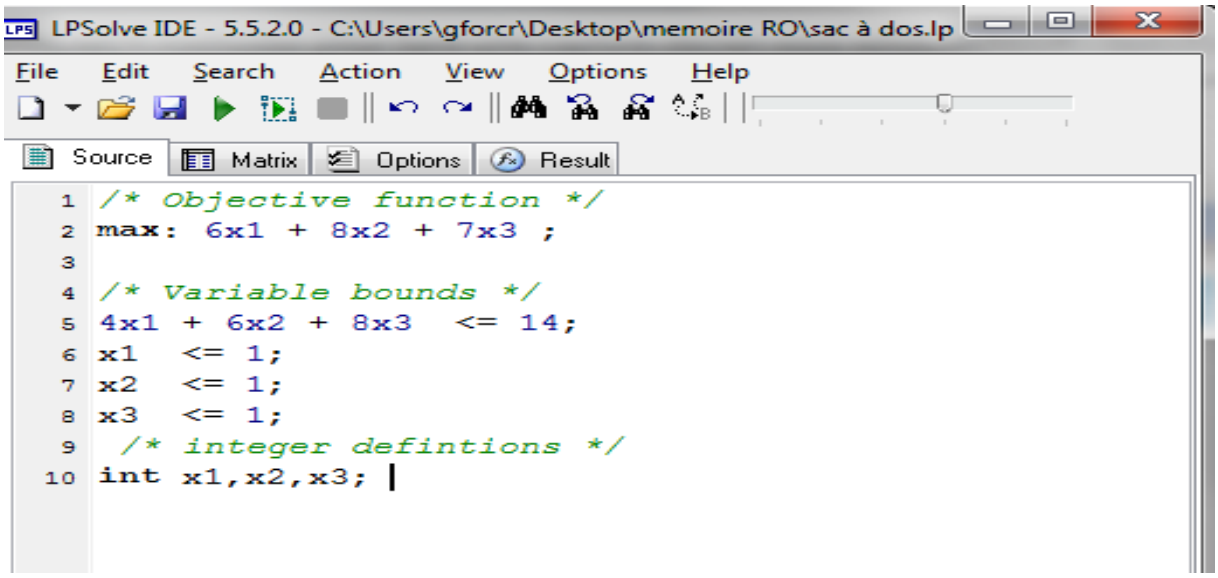


FIGURE 4.1 – Programme du problème du sac à dos (LPSolve IDE)

Le résultat obtenu est :

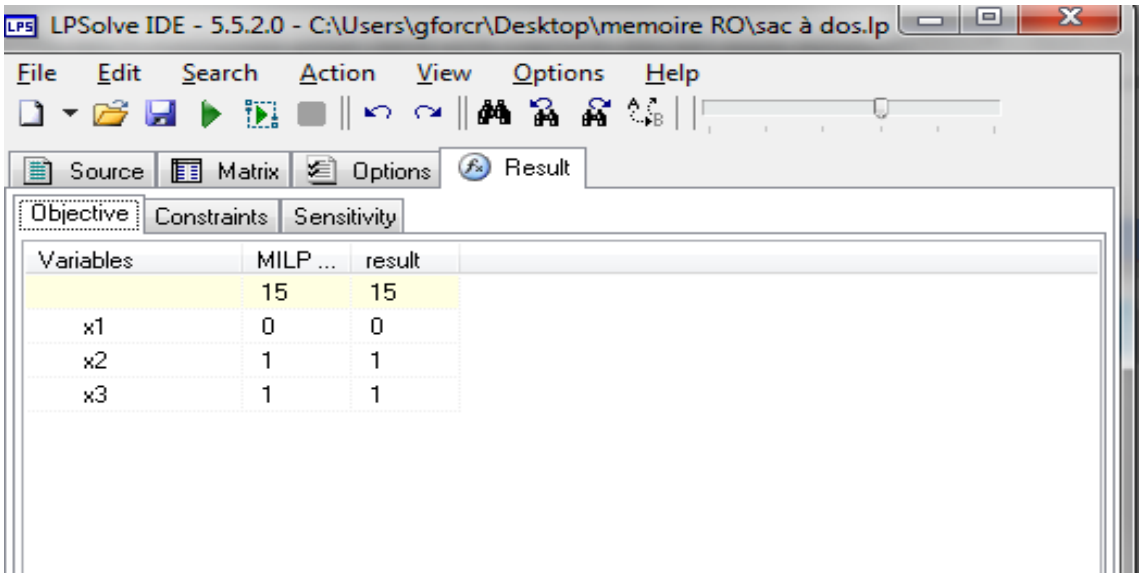


FIGURE 4.2 – Résultat du problème du sac à dos (LPSolve IDE)

La solution est (0,1,1)
Ce qui confirme le résultat déjà trouvé, en utilisant la méthode de Gomory.

4.2 Problème du transport :

Le problème est comme suit :

$$/* \text{ objective function } */$$
$$max : 80x1+215x2+100x3+108x4+102x5+68x6 ;$$
$$/* \text{ Variable bounds } */$$
$$x1 +x2 = 1000 ;$$
$$x3+x4 =1500 ;$$
$$x5+x6=1200 ;$$
$$x1+x3+x5=2300 ;$$
$$x2+x4+x6=1400 ;$$

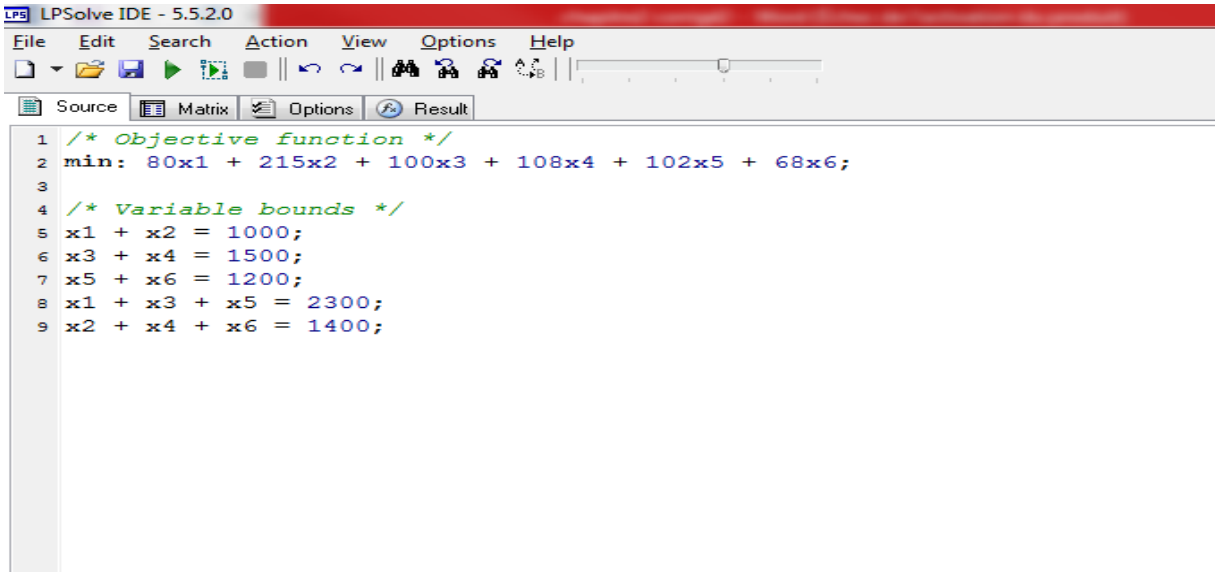


FIGURE 4.3 – Programme du problème de transport (LPSolve IDE)

Le résultat obtenu est :

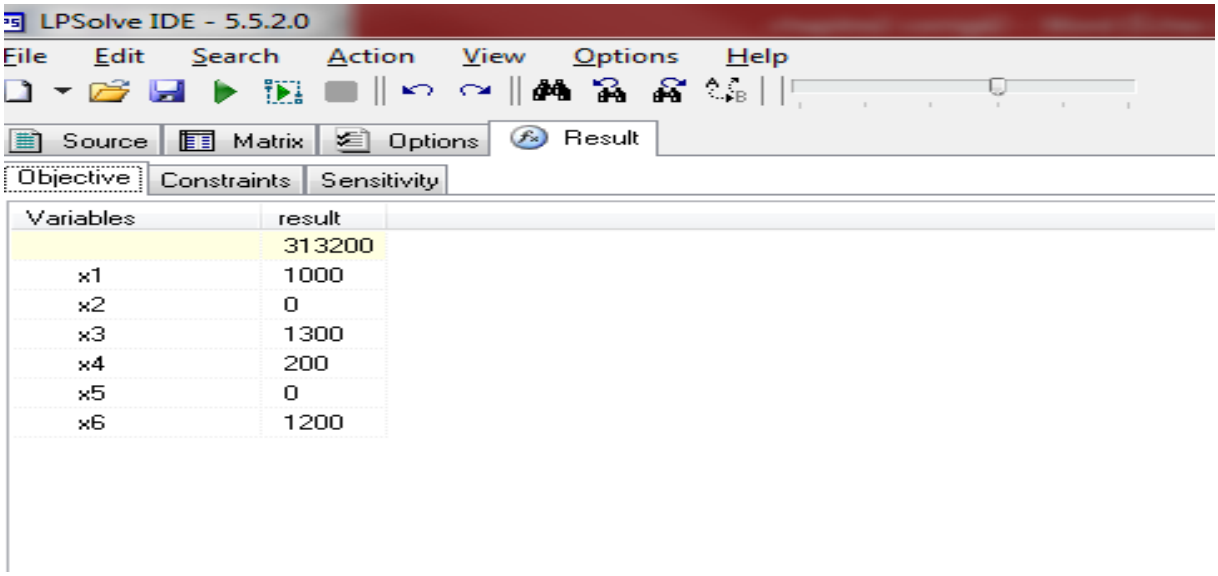


FIGURE 4.4 – Programme du problème de transport (LPSolve IDE)

4.3 Problème du voyageur de commerce :

Avant d'ecrire le programme de ce problème posons les variables suivantes :

x1 le chemin de la ville V_0 *à la ville* V_1
x2 le chemin de la ville V_0 *à la ville* V_2
x3 le chemin de la ville V_0 *à la ville* V_3
x4 le chemin de la ville V_0 *à la ville* V_4
x5 le chemin de la ville V_1 *à la ville* V_0
x6 le chemin de la ville V_1 *à la ville* V_2
x7 le chemin de la ville V_1 *à la ville* V_3
x8 le chemin de la ville V_1 *à la ville* V_4
x9 le chemin de la ville V_2 *à la ville* V_0
x10 le chemin de la ville V_2 *à la ville* V_1
x11 le chemin de la ville V_2 *à la ville* V_3
x12 le chemin de la ville V_2 *à la ville* V_4
x13 le chemin de la ville V_3 *à la ville* V_0
x14 le chemin de la ville V_3 *à la ville* V_1
x15 le chemin de la ville V_3 *à la ville* V_2
x16 le chemin de la ville V_3 *à la ville* V_4
x17 le chemin de la ville V_4 *à la ville* V_0
x18 le chemin de la ville V_4 *à la ville* V_1
x19 le chemin de la ville V_4 *à la ville* V_2
x20 le chemin de la ville V_4 *à la ville* V_3

Le problème est comme suit :

/ Objective function */*
min :8x1+4x2+2x3+3x4+9x5+7x6+1x7+6x8+3x9+7x10+6x11+6x12+2x13+1x14
+6x15+4x16+7x17+6x18+6x19+2x20 ;
/ Variable bounds */*
x1 + x5 <=1 ;
x2 + x9 <=1 ;
x6 + x10 <=1 ;
x3 + x13 <=1 ;
x7 + x14 <=1 ;
x11 + x15 <=1 ;
x4 + x17 <=1 ;
x8 + x18 <=1 ;

$$x12 + x19 <= 1;$$
$$x16 + x20 <= 1;$$
$$x1 + x2 + x3 + x4 = 1;$$
$$x17 + x18 + x19 + x20 = 1;$$
$$x5 + x6 + x7 + x8 = 1;$$
$$x9 + x10 + x11 + x12 = 1;$$
$$x13 + x14 + x15 + x16 = 1;$$
$$x5 + x9 + x13 + x17 = 1;$$
$$x1 + x10 + x14 + x18 = 1;$$
$$x2 + x6 + x15 + x19 = 1;$$
$$x3 + x7 + x11 + x20 = 1;$$
$$x4 + x8 + x12 + x16 = 1;$$
$$/* \text{ integer defintions } */$$
$$int\ x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20;$$

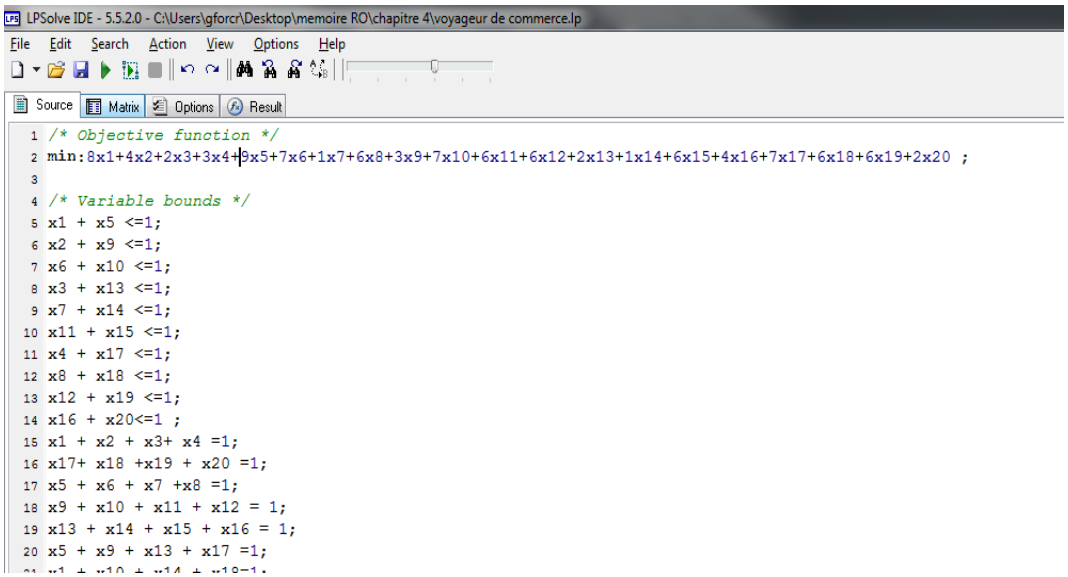


FIGURE 4.5 – Programme du problème du voyageur de commerce (LPSolve IDE)

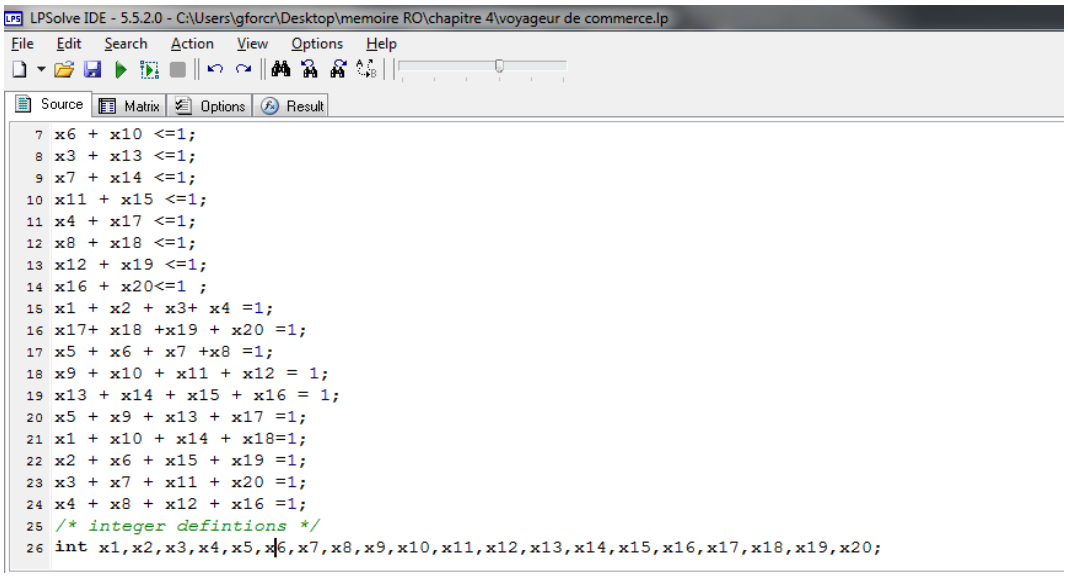


FIGURE 4.6 – Suite du programme du problème du voyageur de commerce (LPSolve IDE)

Le résultat obtenu est :

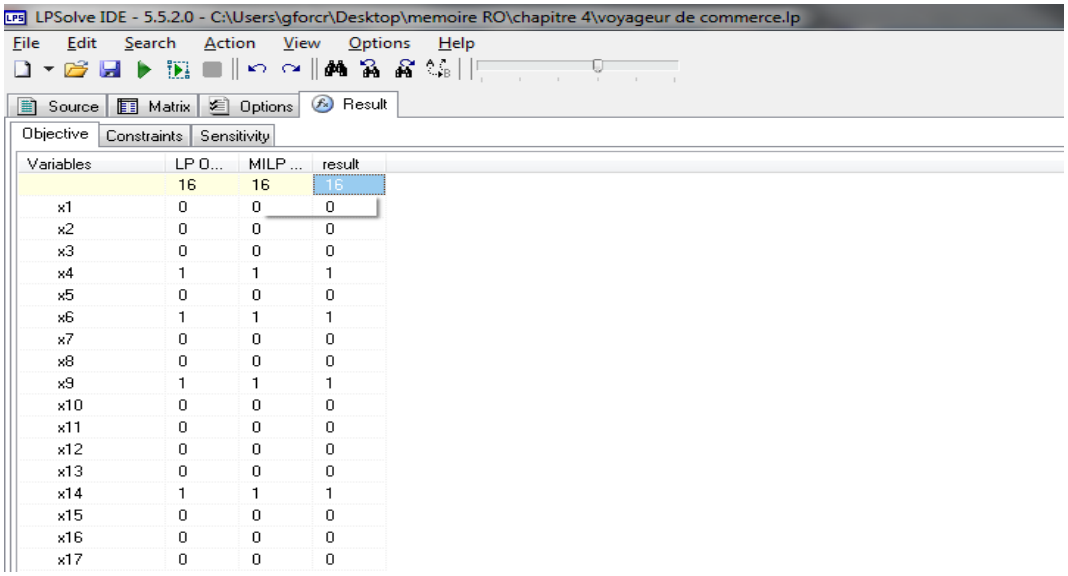


FIGURE 4.7 – Résultat du problème du voyageur de commerce (LPSolve IDE)

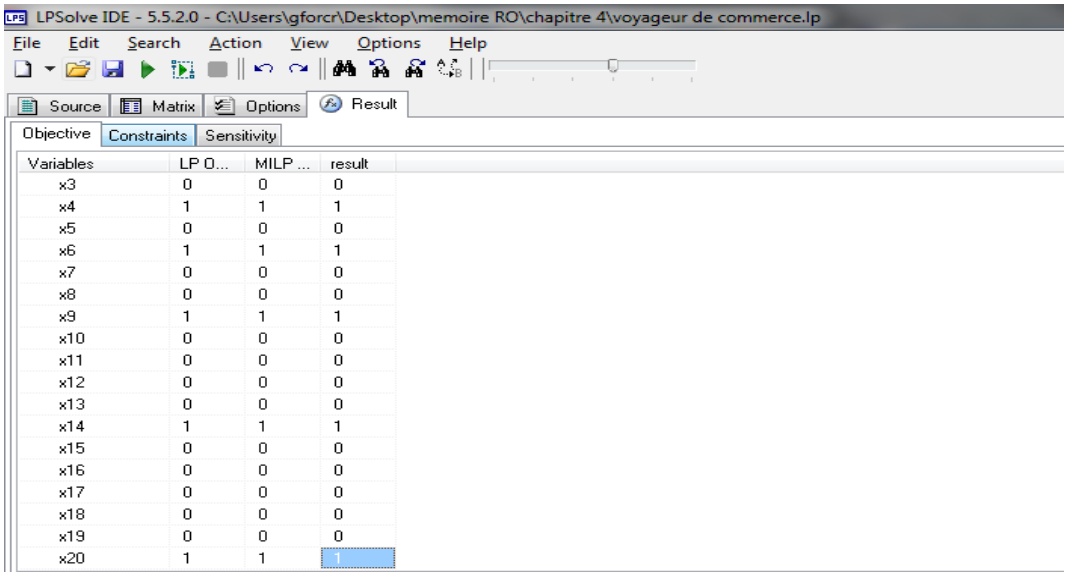


FIGURE 4.8 – Suite du résultat du problème du voyageur de commerce (LPSolve IDE)

La solution est X^*=(0,0,0,1,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,1)
Le chemin est dans l'ordre suivant : $\{V_0, V_4, V_3, V_1, V_2\}$

Conclusion générale et Perspectives

Dans ce mémoire, nous avons traité les programmes linéaires en nombres entiers (PLNE) sont plus complexes à résoudre qu'un programme linéaire tout court (PL). Pourtant, la plupart des problèmes dans les situations réelles correspondent plutôt à des PLNE qu'à de simples programmes linéaires, d'où la nécessité de savoir les résoudre.

La résolution des PLNE passe habituellement par la résolution du programme linéaire relaxé c'est-à-dire du programme linéaire sans les contraintes d'intégrité. Ce programme linéaire relaxé se résout le plus souvent par la méthode du simplexe. La solution finale obtenue est optimale mais n'est pas nécessairement entière, dans ce cas la résolution du PLNE se poursuit, et cela, toujours à l'aide de la méthode du simplexe mais combinée soit à des mécanismes de coupe, les coupes de Gomory, soit à la méthode de séparation et évaluation (Branch and Bound).

La méthode des coupes de Gomory coupe le polyèdre et se rapproche de la solution optimale d'itération en itération, alors que la méthode de B&B divise, à chaque itération le polyèdre en parties, et élimine celle qui ne contient pas de solutions entières.

L'inconvénient de la méthode des troncatures est qu'elle augmente de la complexité du problème, et sa solution reste irréalisable jusqu'à l'arrêt de l'algorithme.

Pour la méthode de Branch and Bound nous avons illustré le problème du voyageur de commerce. Ce problème est toujours d'actualité dans la recherche en informatique, étant donné le nombre important de problèmes réels auxquels il correspond. Les problèmes dérivés et les extensions en sont très nombreux.

Parmi les perspectives de ce mémoire nous identifions deux axes possibles :

Le premier axe est la programmation linéaire mixte en nombres entiers.

Une deuxième piste intéressante est la possibilité d'avoir un programme non linéaire en nombres entiers.

Bibliographie

[1] *Christelle Guéret - Christian Prins - Marc Sevaux, programmation linéaire, éditions EYROLLES ISBN 2-212-09202-4.*

[2] *Gérald Baillargeon, outil de la recherche opérationnelle programmation linéaire appliquée, les éditions SMG ISBN : 2-89094-054-3.*

[3] *Leo Liberti, Ruslan Sadykov ,Introduction à la Programmation Linéaire en Nombres Entiers, LIX, Ecole Polytechnique, 2006.*

[4] *Ramonjison Domoina Mahefasoa, mémoire de master : Une autre méthode de résolution des programmes linéaires en nombres entiers, Université D'Antananarivo, 2017.*

[5] *Yves De Smet, Bernard Fortz, INFO-F-310 - Algorithmique 3 et Recherche Opérationnelle 2013-2014.*

[6] *Bernard Fortz, Recherche opérationnelle et applications 2012-2013.*

[7] *Maria Alejandra Ayala P,thèse de doctorat : Programmation linéaire en nombres entiers pour l'ordonnancement cyclique sous contraintes de ressources, à l'université de Toulouse, le 15 Juin 2011.*

[8] *Sidi Mohamed Douiri, Souad Elbernoussi, Halima Lakhbab, Cours des Méthodes de Résolution Exactes Heuristiques et Métaheuristiques, master codes, cryptographie et sécurité de l'information, Faculté des Sciences de Rabat Laboratoire de Recherche Mathématiques, Informatique et Applications, Université Mohammed V.*

[9] *Sameh Grainia, mémoire en vue de l'obtention du grade de Maître ès sciences (M. SC.) en informatique : L'algorithme de Branch and Price and Cut pour le problème de conception de réseaux avec coûts fixes et sans capacité, à l'université de Montréal, Avril 2015.*

[10] *Fabian Bastin, Modèles de Recherche Opérationnelle, Département d'Informatique et de Recherche Opérationnelle Université de Montréal, IFT-1575, Hiver 2010*

[11] Pierre Fouilhoux, Recherche Opérationnelle et Optimisation Combinatoire, (Rappels) Branchement et Evaluation (Branch-and-Bound), Master d'Informatique - Spécialité Androide Module MAOA, Sorbonne Université 2019-2020)