

# **TP de méthodes de résolution de problèmes NP-complets**

# Table des matières

<b>1</b>	<b>Partie théorique</b>	<b>2</b>
1.1	Sur le problème de la couverture sommet minimale : trois approches différentes :	2
1.2	Sur le problème de la couverture d'ensembles . . . . .	5
1.3	Sur le problème du couplage maximum de poids minimum . . . . .	7
1.4	Sur le problème de la coupe maximum . . . . .	8
1.5	Sur le problème de partition . . . . .	9
1.6	Sur le problème du sac à dos simple . . . . .	10
1.7	Programmation dynamique . . . . .	11
1.8	Sur le produit matriciel . . . . .	11
1.9	Résolution numérique . . . . .	12
1.10	Seuil d'approximation pour le problème Bin Packing . . . . .	12
1.11	Seuil d'approximation pour le problème de la coloration de sommets (resp. d'arêtes) . . . . .	12
1.12	Comparaison de <i>branch and bound</i> et <i>branch and cut</i> . . . . .	12
<b>2</b>	<b>Partie pratique</b>	<b>25</b>

# Chapitre 1

## Partie théorique

### 1.1 Sur le problème de la couverture sommet minimale : trois approches différentes :

#### 1.1.1 Première approche : la programmation linéaire en nombres entiers

##### Justification de l'utilisation de la Programmation Linéaire en Nombres Entiers

On considère le problème de la couverture minimale sous la forme suivante :

$$\begin{cases} \min z = \sum_{j=1}^n x_j \\ x_r + x_s \geq 1, \quad \forall \{v_r, v_s\} \in E \\ x_j \in \{0, 1\} \quad j = 1, \dots, n \end{cases}$$

La fonction objectif représente le nombre de sommets utilisés par la solution du problème. Le fait de minimiser la fonction objectif permet d'assurer la couverture minimale. Chaque clause est relative à une arête du graphe, et impose qu'au moins un des sommets adjacents à cette arête soit dans la couverture.

On a donc bien un problème de Programmation Linéaire en Nombres Entiers permettant de résoudre le problème de la couverture minimale.

##### Justification des clauses

Considérons le graphe donné par la figure 1.1.1.

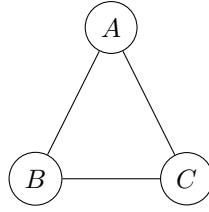


FIGURE 1.1 – Exemple

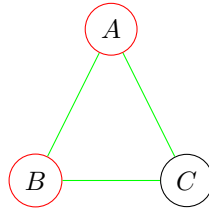


FIGURE 1.2 – Solution

Sur ce graphe, le programme linéaire en nombres entiers est le suivant :

$$\left\{ \begin{array}{l} \min z = x_A + x_B + x_C \\ x_A + x_B \geq 1 \\ x_A + x_C \geq 1 \\ x_B + x_C \geq 1 \\ x_A, x_B, x_C \in \{0, 1\} \end{array} \right.$$

Il est très simple ici de comprendre pourquoi il est impossible de considérer le programme linéaire suivant :

$$\left\{ \begin{array}{l} \min z = x_A + x_B + x_C \\ x_A + x_B = 1 \\ x_A + x_C = 1 \\ x_B + x_C = 1 \\ x_A, x_B, x_C \in \{0, 1\} \end{array} \right.$$

Ce programme ne permet pas de résoudre la couverture minimale sur le graphe donné par la figure 1.1.1. Quelque soit le sommet choisi dans un premier lieu pour appartenir à la couverture minimale, il est impossible d'en choisir un second pour compléter cette dernière. Prenons un exemple, nous forçons le sommet  $A$  à appartenir à la couverture minimale (respectivement  $B$  et  $C$ ). Ce choix force :  $x_B = 0$  et  $x_C = 0$  (respectivement,  $x_A = 0$  et  $x_C = 0$ , et  $x_A = 0$  et  $x_B = 0$ ). Il est donc impossible de respecter la clause  $x_B + x_C = 1$ , le problème (au vu de sa modélisation) n'aurait donc pas de solution, or le graphe de la figure 1.1.1 montre le contraire.

## Une borne inférieure des solutions optimales

On cherche à montrer qu'une solution optimale du programme linéaire en nombres entiers est une borne inférieure de toute solution optimale du programme relaxé. Raisonnons par l'absurde et considérons une solution optimale du programme linéaire, notée  $n^*$  telle qu'il existe  $x^*$  solution optimale du problème relaxé vérifiant  $x^* < n^*$ . Toute solution du programme linéaire est solution du programme relaxé<sup>1</sup>. Ceci implique :  $n^*$  solution du programme relaxé, et donc  $x^* < n^*$  impossible. On a donc :  $n^* \leq x^*$  ce qui est la définition d'une borne inférieure.

## A propos de la relaxation de contrainte

Pour démontrer que la relaxation des contraintes d'intégrité implique  $x_r \geq \frac{1}{2}$  ou  $x_s \geq \frac{1}{2}$ , le raisonnement par l'absurde sera utilisé. Soient  $x_s$  et  $x_r$  les variables relatives aux sommets  $r$  et  $s$  adjacents à l'arête  $(rc)$  et telles que, après relaxation des contraintes, on a :  $x_r < \frac{1}{2}$  et  $x_s < \frac{1}{2}$ . On en déduit donc que  $x_r + x_s < 1$  et donc la contrainte liée à l'arête  $(rs)$  est violée, l'hypothèse de départ est donc fausse. On a donc,  $\forall (rs) \in V : x_r \geq \frac{1}{2}$  et  $x_s \geq 12$ .

## Une 2-approximation

Mettons en évidence le pire des cas pouvant se présenter : pour une arête  $(rs) \in V$ , un seul sommet est nécessaire pour la couverture de cette dernière dans le cas de la couverture minimale, mais l'algorithme approché retourne :  $x_r = x_s = \frac{1}{2}$ . Après la phase d'arrondis, on a  $x_r = x_s = 1$  est donc les deux sommets appartiennent à la solution approchée, cette phase multiplie donc au pire le nombre de sommets (pour chaque clause par 2), ce qui implique que le cardinal de la solution approximée est au plus 2 fois la solution optimale.

Cet algorithme est donc une 2-approximation.

## Dans le cas d'un graphe valué

### 1.1.2 Seconde approche : la recherche d'un couplage maximal

## Une 2-approximation

Commençons par prouver que l'algorithme retourne une couverture des arêtes par les sommets. Considérons donc une arête  $(rs)$  non couverte par l'ensemble de sommets retourné par l'algorithme, par définition du couplage, il serait donc possible d'ajouter  $(rs)$  au couplage. Or le couplage calculé par l'algorithme est maximal, on en déduit que l'arête  $(rs)$  telle qu'elle est définie ne peut exister et donc que l'ensemble de sommets obtenu couvre l'ensemble des arêtes du graphe.

---

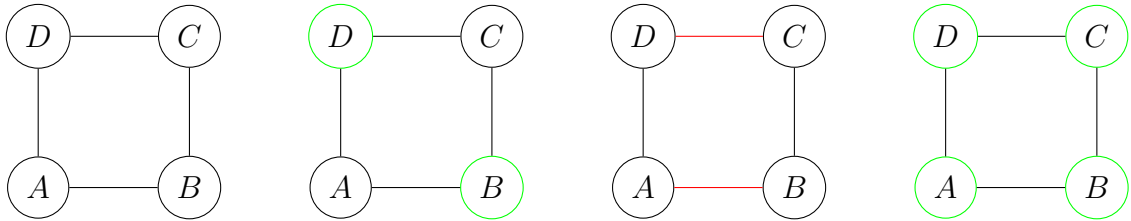
1. Une solution appartenant à  $\mathbb{N}$  appartient aussi à  $\mathbb{R}$

Appelons  $c$  le couplage calculé par l'algorithme et  $x^*$  la solution optimale du problème de la couverture par les sommets, on sait que  $\text{Card}c \leq \text{Card}x^*$ , or pour construire la solution approchée, on ajoute à  $C$  les deux extrémités des arêtes utilisées pour le couplage. On a donc :

$$\begin{aligned} \text{Card}C &= 2 \times \text{Card}c \leq 2 \times \text{Card}x^* \\ \Rightarrow \frac{\text{Card}C}{\text{Card}x^*} &= 2 \end{aligned}$$

### Exemple de graphe foireux

Le graphe suivant met en évidence la borne 2 de l'algorithme.



Le graphe      Une couverture minimale      Un couplage maximale      La solution renvoyée

### Application de l'algorithme

## 1.2 Sur le problème de la couverture d'ensembles

### 1.2.1 Modélisation du problème à l'aide de la PLNE

On considère le problème de la couverture d'ensemble, défini par : Soit  $E = \{e_1, \dots, e_n\}$ , soient  $S_1, \dots, S_m$  des sous-ensembles non vides de  $E$  tels que  $\forall i \in \{1, \dots, m\}$ , on a :  $S_i \subset E$ . On associe à chaque ensemble  $S_j$  un poids  $w_j \geq 0$ . Le problème consiste à trouver une collection de sous-ensemble de poids minimum et telle que  $\bigcup_{i=1}^m S_i = E$ .

Ce problème peut s'exprimer à l'aide de la Programmation Linéaire en Nombres entiers de la manière suivante :

$$\begin{cases} \min z = \sum_{i=1}^m w_i x_i \\ \sum_{x_j/e_i \in x_j} x_j \geq 1 \quad \forall e_i \in E \\ x_i \in \{0, 1\} \end{cases}$$

### 1.2.2 Procédure d'arrondis

Soit  $f = \max_{i=1,\dots,n} f_i$ , avec  $f_i$  le cardinal de l'ensemble des sous-ensembles de  $E$  contenant  $e_i$  :  $f_i = |\{j : e \in S_j\}|$ .

Définissons la procédure d'arrondis suivante :

$$x_i = \begin{cases} 1 & \text{si } x_i \geq \frac{1}{f} \\ 0 & \text{sinon} \end{cases}$$

Cherchons à démontrer que cette procédure d'arrondis garantit une solution réalisable. Pour ce faire, nous allons procéder par l'absurde. Supposons qu'il existe un sommet  $k$  qui ne respecte pas sa contrainte d'intégrité associée, à savoir :

$$\sum_{x_j/e_k \in x_j} < 1$$

Les variables  $x_j$  de cette contrainte étant définies positives et entières, le cas pris en considération si dessus implique que toutes les variables de l'inéquation sont nulles pour le programme en nombres entiers et donc :

$$\forall x_j/e_k \in x_j < \frac{1}{f}$$

dans le cas de la version relaxée du problème. Or ceci n'est possible que si le nombre de sous-ensemble contenant  $x_k$  est supérieur à  $f$ , ce qui est impossible par définition. Tous les sommets respectent donc leur contrainte d'intégrité et on en déduit que la procédure d'arrondis garantit une solution réalisable.

### 1.2.3 Existence d'un algorithme $f$ -approché

Considérons l'algorithme 1

---

**Algorithm 1** Approximation couverture par ensemble

---

- 1: Exprimer le problème en programmation linéaire en nombres entiers
  - 2: Résoudre la version relaxée
  - 3: Réaliser la procédure d'arrondis
- 

En utilisant la procédure d'arrondis étudiée plus haut, on sait que pour chaque clause, il existe  $x_k$  tel que  $x_k \geq \frac{1}{f}$ , au pire des cas<sup>2</sup>, chaque  $x_k$  est multiplié par  $f$ , ce qui implique que la solution obtenue est au plus  $f$  fois plus grande que la solution optimale. Il s'agit donc d'un algorithme  $f$ -approché.

### 1.2.4 Cas ou $f = 2$

Si  $f = 2$  alors le problème devient une couverture minimum par sommets.

---

2. cas similaire à celui de l'exercice 1

### 1.3 Sur le problème du couplage maximum de poids minimum

#### 1.3.1 Modélisation du problème

Soit un graphe  $G = (V, E)$  avec  $V$  l'ensemble de ses sommets et  $E$  l'ensemble de ses arêtes. Soit  $\{\forall(i, j) \in E, X_{(i,j)}\}$  un ensemble de variables booléennes qui indiquent le choix de l'arête  $(i, j)$  correspondante dans le couplage. Soit  $P_{(i,j)}$  le poids de l'arête  $(i, j)$ .

La première modélisation intuitive est la suivante :

*Maximiser*

$$\sum_{(i,j) \in E} (X_{(i,j)} \times H) - \sum_{(i,j) \in E} (X_{(i,j)} \times P_{(i,j)})$$

*Sous contraintes*

$$\begin{aligned} \forall i \in V, \sum_{(i,j) \in E} X_{(i,j)} + \sum_{(j,i) \in E} X_{(j,i)} &\leq 1 \\ \forall(i, j) \in E, X_{(i,j)} &\in \{0, 1\} \\ \forall(i, j) \in E, P_{(i,j)} &\geq 0 \end{aligned}$$

avec  $H$  une constante très grande. Cependant, après quelques instants de réflexion, nous pouvons imaginer une modélisation plus élégante :

*Minimiser*

$$\sum_{(i,j) \in E} (X_{(i,j)} \times P_{(i,j)})$$

*Sous contraintes*

$$\begin{aligned} \forall i \in V, \sum_{(i,j) \in E} X_{(i,j)} + \sum_{(j,i) \in E} X_{(j,i)} &= 1 \\ \forall(i, j) \in E, X_{(i,j)} &\in \{0, 1\} \\ \forall(i, j) \in E, P_{(i,j)} &\geq 0 \end{aligned}$$

En effet, les contraintes forcent le couplage à être maximum tandis que la fonction objectif le force à tendre vers le poids minimum.

#### 1.3.2 Modélisation du problème appliquée au graphe de la figure 2

*Minimiser*

$$\epsilon \times X_{ab} + \epsilon \times X_{bc} + \epsilon \times X_{ac} + M \times X_{ae} + M \times X_{cd} + M \times X_{bf} + \epsilon \times X_{df} + \epsilon \times X_{de} + \epsilon \times X_{fe}$$

*Sous contraintes*

$$\begin{aligned} X_{ab} + X_{ac} + X_{ae} &= 1 \\ X_{ab} + X_{bc} + X_{bf} &= 1 \\ X_{ac} + X_{bc} + X_{cd} &= 1 \\ X_{cd} + X_{de} + X_{df} &= 1 \end{aligned}$$



$$\begin{aligned}
X_{ae} + X_{de} + X_{df} &= 1 \\
X_{ef} + X_{df} + X_{bf} &= 1 \\
\forall (i, j) \in E, X_{(i,j)} &\in \{0, 1\} \\
\epsilon &\geq 0 \\
M &\geq 0
\end{aligned}$$

### 1.3.3 Solution optimale entière $z(ILP)$

Sur un exemple de cette taille, il est facile de trouver une solution à la main. Il y a plusieurs solutions optimales de poids total  $M + 2\epsilon$  sur cet exemple ; l'une d'entre elles est le couplage  $\{(a, b), (c, d), (e, f)\}$ .

La résolution de ce PLNE par *glpsol* (solveur de *GLPK*) donne bien la même solution.

### 1.3.4 Solution optimale $z(LP)$ pour le programme relaxé

Relaxer le programme revient à transformer la contrainte d'intégrité des  $X_{(i,j)}$  en la contrainte suivante :

$$\forall (i, j) \in E, 0 \leq X_{(i,j)} \leq 1$$

En rentrant le PL relaxé dans *glpsol*, nous obtenons le couplage de poids total  $3\epsilon$  :

$$\{X_{ab} = 0.5; X_{ac} = 0.5; X_{bc} = 0.5; X_{df} = 0.5; X_{ef} = 0.5; X_{de} = 0.5\}$$

### 1.3.5 Conclusion sur la pertinence de la formulation

La solution trouvée au PLNE étant optimale, la formulation du problème semble être pertinente.

## 1.4 Sur le problème de la coupe maximum

### 1.4.1 Complexité

A chaque itération de l'algorithme, la valeur de la coupe maximale augmente au minimum d'une unité. Or la valeur de la coupe maximale étant bornée par le nombre d'arêtes, on obtient donc que l'algorithme effectue au plus  $m$  opérations. On a donc un algorithme en  $O(m)$ .

### 1.4.2 Un algorithme 2-approché

Considérons  $(Y_1, Y_2)$  la coupe renvoyée par l'algorithme, nous chercherons dans un premier temps à montrer que chaque sommet dans  $Y_1$  admet au moins autant d'arêtes dans  $Y_2$  que dans  $Y_1$ .

Pour ce faire, considérons un sommet  $v \in Y_1$ , supposons que ce sommet possède plus d'arêtes dans  $Y_1$  que dans  $Y_2$ , nous noterons  $a_1$  le nombre d'arêtes incidentes à  $v$  dans  $Y_1$  et  $a_2$  le nombre d'arêtes adjacentes à  $v$  dans  $Y_2$ . Déplacer  $v$  de  $Y_1$  vers  $Y_2$  reviendrait à diminuer la coupe de  $a_2$  et augmenter celle-ci de  $a_1$ , or d'après l'hypothèse de départ  $a_2 < a_1$ , on observerait une augmentation de la valeur de la coupe, ce qui est impossible si  $(Y_1, Y_2)$  est une coupe retournée par l'algorithme. On en déduit donc que l'hypothèse de départ est fausse.

Autrement dit, pour un sommet  $v \in V$ , en notant  $d_v$  le degré de  $v$ ,  $v_c$  le nombre d'arêtes adjacentes à  $v$  traversant la coupe et  $v_s$  le nombre d'arêtes adjacentes à  $v$  ne la traversant pas, on peut écrire :

$$\begin{aligned} v_c + v_s &= d_v \quad \text{or on a } v_c \geq v_s \\ \Rightarrow v_c &\geq \frac{d_v}{2} \end{aligned}$$

Si l'on généralise sur l'ensemble du graphe, on peut en déduire :

$$\begin{aligned} |(Y_1, Y_2)| &= \frac{1}{2} \sum_{v \in V} v_c \\ &\geq \frac{1}{2} \sum_{v \in V} \frac{d_v}{2} \\ &= \frac{m}{2} \end{aligned}$$

On a donc :  $|(Y_1, Y_2)| \geq \frac{m}{2}$ , de plus comme vu précédemment, la valeur de la coupe maximale (que nous noterons  $OPT$ ) est bornée par le nombre d'arêtes. On peut donc en déduire :

$$\frac{OPT}{|(Y_1, Y_2)|} \leq \frac{m}{\frac{m}{2}} = 2$$

L'algorithme donné est donc bien un algorithme 2-approché.

### 1.4.3 Atteindre la borne

## 1.5 Sur le problème de partition

## 1.6 Sur le problème du sac à dos simple

### 1.6.1 Construction d'un algorithme approché

#### Complexité de l'algorithme

La complexité de l'algorithme, une fois les nombres triés, est en  $O(n)$ . Cependant, il est possible de démontrer que la complexité des algorithmes de tris basés sur des fonctions de comparaisons ne peut être inférieures à  $O(n \log n)$ . la complexité de cet algorithme est donc similaire à la complexité de l'algorithme de tri utilisé.

#### Minoration de $cost(T)$

Dans un premier temps nous montrerons que l'existence d'un indice  $j$  pour lequel  $cost(T) + w_{j+1} > b$  n'est pas systématique, mais que si ce dernier n'existe pas, le problème n'existe pas non plus.

En effet, si cet indice n'existe pas, on a alors :

$$\sum_{i=1}^n w_i \leq b$$

On en déduit donc que la solution optimale est donnée par  $T = \{1, \dots, n\}$ . Ce qui en soit même n'a aucun intérêt<sup>3</sup>. Nous admettons donc l'existence de cet indice  $j$ .

Supposons, à l'indice  $j$ , que  $cost(T) \leq \frac{b}{2}$ . Par définition de l'indice  $j$ , on a  $cost(T) + w_{j+1} > b$ , ce qui implique  $w_{j+1} > \frac{b}{2}$  et donc :

$$\begin{aligned} w_{j+1} &> cost(T) \\ \Rightarrow w_{j+1} &> \sum_{i=1}^j w_i \\ \Rightarrow w_{j+1} &> w_i \end{aligned}$$

Or les éléments  $w_i$  étant triés, on a  $w_j \geq w_{j+1}$ . L'hypothèse de départ est donc fausse.

#### Performance relative de l'algorithme

Notons  $OPT$  la solution optimale au problème du sac à dos, et  $A$  la solution approchée donnée par l'algorithme. Rappelons, par définition du problème, que  $OPT \leq b$ . La performance relative nous est donnée par :

$$\frac{OPT}{A} \leq \frac{b}{\frac{b}{2}} = 2$$

La performance relative de l'algorithme est donc 2.

---

3. Le problème du sac à dos, dans cette configuration n'est plus NP-Complet

### 1.6.2 Construction d'un schéma d'approximation

#### Complexité de l'algorithme

De façon très grossière, la construction de l'ensemble des sous-ensembles  $S_k$  de  $k$  éléments se fait en  $k \times n^k$  opérations. On a donc une construction des sous-ensembles réalisée en :

$$\sum_{i=1}^k i n^i$$

. Soit une construction en  $O(n^k)$ .

L'ensemble étant déjà trié, l'algorithme glouton s'exécute en  $O(n)$ , d'où une exécution totale de l'algorithme en  $O(n^{k+1})$ .

## 1.7 Programmation dynamique

a

### 1.7.1 Sur le problème de la partition

Condition nécessaire sur la somme des poids des  $n$  objets

Récurrence

Complexité

Jeux d'essais

### 1.7.2 Le problème du sac à dos

Justification des formules

Exemple

Complexité

### 1.7.3 Le problème du voyageur de commerce

Exemple

Complexité

## 1.8 Sur le produit matriciel

a

1.8.1 Nombre d'opérations nécessaires pour un produit

1.8.2 Nombre de parenthésages possibles d'un produit de  $k$  matrices

1.8.3 Récurrence

1.8.4 Exemple

## 1.9 Résolution numérique

a

1.10 Seuil d'approximation pour le problème Bin Packing

a

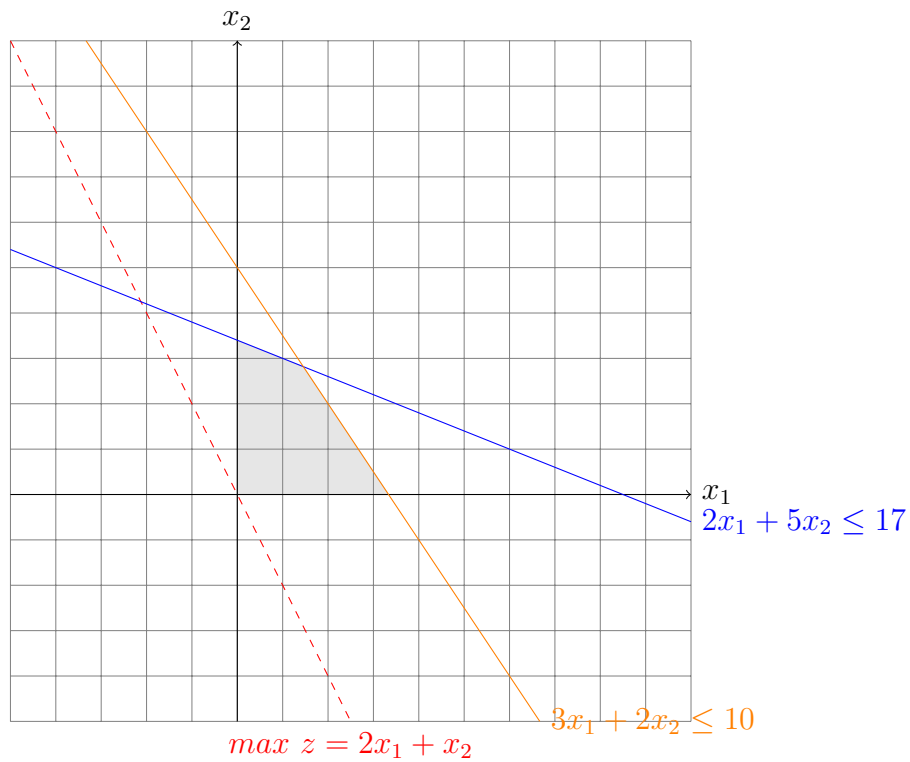
1.11 Seuil d'approximation pour le problème de la coloration de sommets (resp. d'arêtes)

a

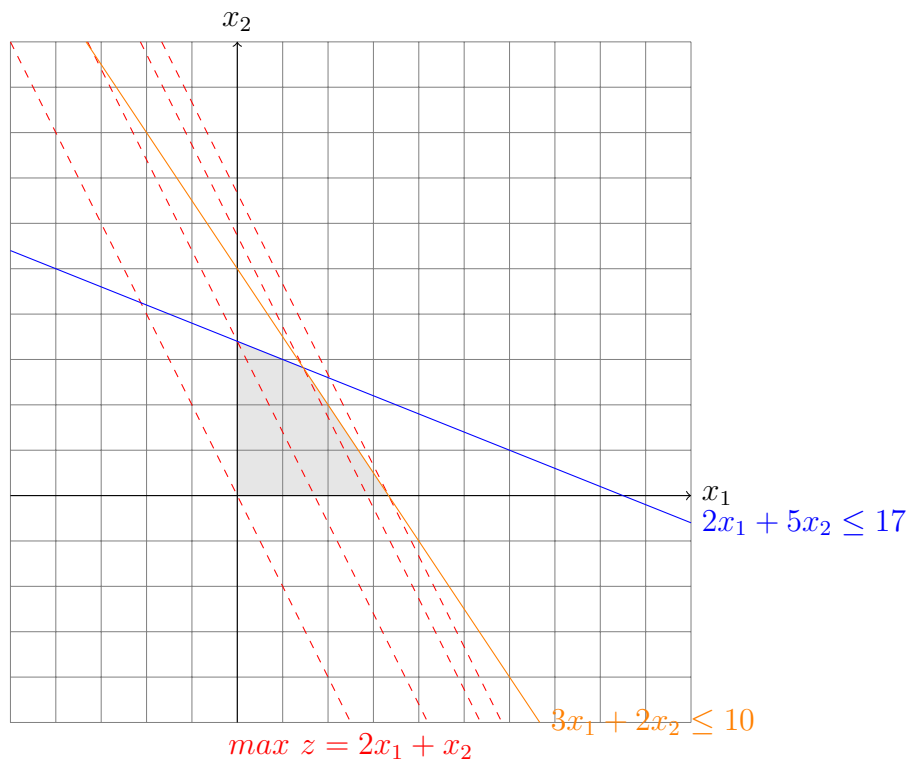
1.12 Comparaison de *branch and bound* et *branch and cut*

Nous considérons le programme linéaire suivant :

$$PL_0 \begin{cases} \max z(x_1, x_2) = 2x_1 + x_2 \\ 2x_1 + 5x_2 \leq 17 \\ 3x_1 + 2x_2 \leq 10 \\ x_1, x_2 \geq 0 \end{cases}$$

1.12.1 Polytope associé à  $PL_0$ 

## 1.12.2 Résolution graphique



Nous obtenons grâce à la résolution graphique :

$$\begin{cases} x_1 = \frac{10}{3} \\ x_2 = 0 \\ z = \frac{20}{3} \end{cases}$$

### 1.12.3 Résolution par la méthode du simplexe

Programme linéaire :

$$PL_0 \begin{cases} \max z(x_1, x_2) = 2x_1 + x_2 \\ 2x_1 + 5x_2 \leq 17 \\ 3x_1 + 2x_2 \leq 10 \\ x_1, x_2 \geq 0 \end{cases}$$

Forme standard :

$$PL_0 \begin{cases} \max z(x_1, x_2) = 2x_1 + x_2 \\ 2x_1 + 5x_2 + y_1 = 17 \\ 3x_1 + 2x_2 + y_2 = 10 \\ x_1, x_2, y_1, y_2 \geq 0 \end{cases}$$

Tableaux :

		2	1	0	0
0	$y_1 = 17$	2	5	1	0
0	$y_2 = 10$	3	2	0	1
	$z = 0$	-2	-1	0	0

$y_2$  sort de la base ;  $x_1$  rentre dans la base ; le pivot devient  $a_{1,2} = 3$ .

		2	1	0	0
0	$y_1 = \frac{31}{3}$	0	$\frac{11}{3}$	1	$-\frac{2}{3}$
2	$x_1 = \frac{10}{3}$	1	$\frac{2}{3}$	0	$\frac{1}{3}$
	$z = \frac{20}{3}$	0	$\frac{1}{3}$	0	$\frac{2}{3}$

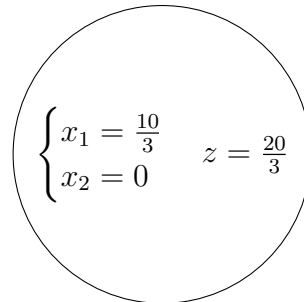
Tous les coûts réduits sont positifs ou nuls donc le simplexe est fini. La solution correspond bien à celle trouvée graphiquement :

$$\begin{cases} x_1 = \frac{10}{3} \\ x_2 = 0 \\ z = \frac{20}{3} \end{cases}$$

### 1.12.4 Recherche d'une solution à valeur entière

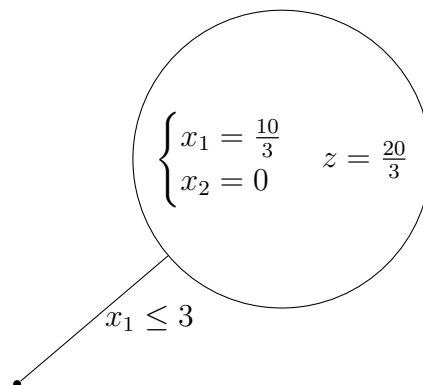
#### Branch and bound

Pour la méthode du Branch and bound, nous partons de la solution optimale réelle trouvée précédemment.



$$\begin{cases} x_1 = \frac{10}{3} \\ x_2 = 0 \end{cases} \quad z = \frac{20}{3}$$

Puis nous ajoutons la contrainte  $x_1 \leq 3$  pour forcer  $x_1$  à être entier.



$$\begin{cases} x_1 = \frac{10}{3} \\ x_2 = 0 \end{cases} \quad z = \frac{20}{3}$$

$x_1 \leq 3$

Nous calculons alors les tableaux du simplexe avec la nouvelle contrainte.

		2	1	0	0	0
0	$y_1 = 17$	2	5	1	0	0
0	$y_2 = 10$	3	2	0	1	0
0	$y_3 = 3$	1	0	0	0	1
	$z = 0$	-2	-1	0	0	0

$y_3$  sort de la base ;  $x_1$  rentre dans la base ; le pivot devient  $a_{1,3} = 1$ .

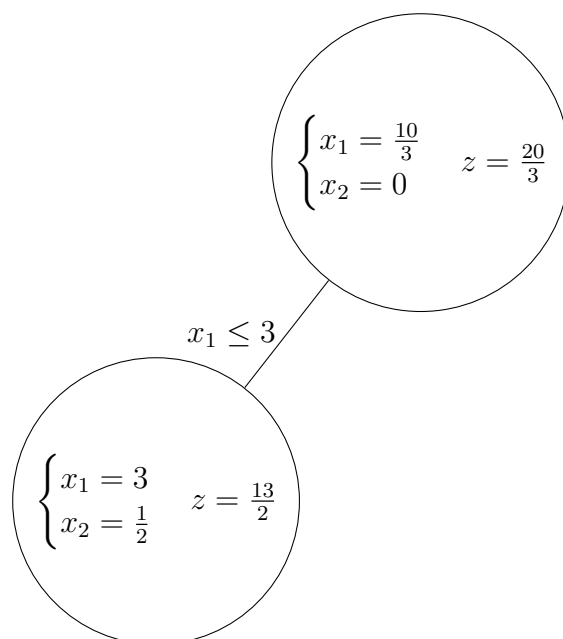


		2	1	0	0	0
0	$y_1 = 11$	0	5	1	0	-2
0	$y_2 = 1$	0	2	0	1	-3
2	$x_1 = 3$	1	0	0	0	1
	$z = 6$	0	-1	0	0	2

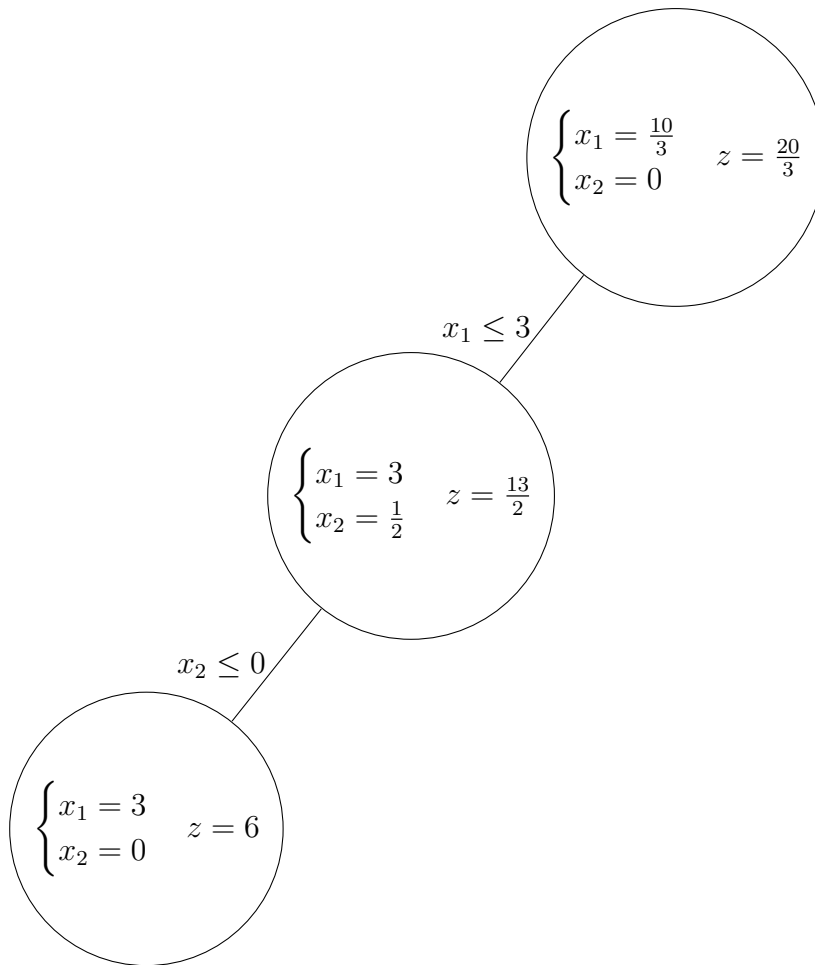
$y_2$  sort de la base ;  $x_2$  rentre dans la base ; le pivot devient  $a_{2,2} = 2$ .

		2	1	0	0	0
0	$y_1 = \frac{17}{2}$	0	0	1	$-\frac{5}{2}$	2
1	$x_2 = \frac{1}{2}$	0	1	0	$\frac{1}{2}$	$-\frac{3}{2}$
2	$x_1 = 3$	1	0	0	0	1
	$z = \frac{13}{2}$	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$

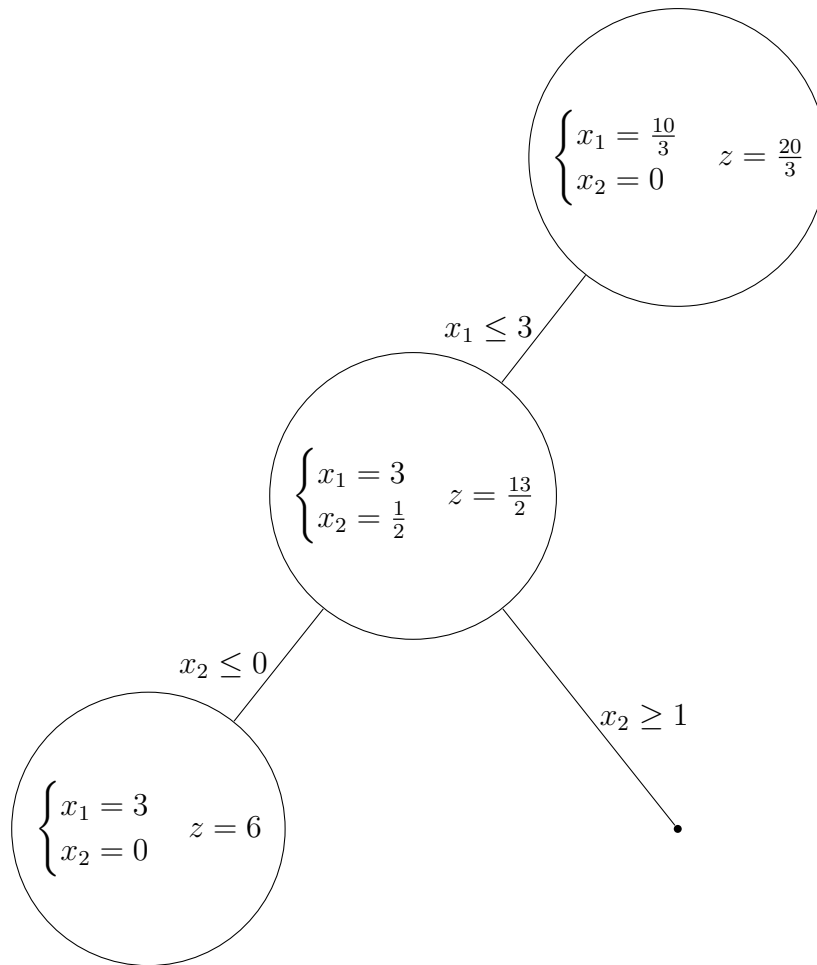
Le simplexe est fini, nous ajoutons le résultat à l'arbre de branch and cut.



$x_1$  a bien une valeur entière mais  $x_2$  est devenu réel, nous ajoutons donc la contrainte  $x_2 \leq 0$ . Comme  $x_2 \geq 0$  par définition, nous obtenons la solution suivante :



Nous obtenons une feuille de l'arbre et une solution entière admissible. Nous devons maintenant développer le reste de l'arbre pour vérifier s'il est possible d'obtenir une meilleure solution entière. Posons donc la contrainte suivante :  $x_2 \geq 1$ .



Afin de trouver une solution, nous appliquons la phase 1 du simplexe dont voici les tableaux.

		0	0	0	0	0	0	-1
0	$y_1 = 17$	2	5	1	0	0	0	0
0	$y_2 = 10$	3	2	0	1	0	0	0
0	$y_3 = 3$	1	0	0	0	1	0	0
-1	$y_5 = 1$	0	1	0	0	0	-1	1
	$z = -1$	0	-1	0	0	0	1	0

$y_5$  sort de la base ;  $x_2$  rentre dans la base ; le pivot devient  $a_{2,4} = 1$ .

		0	0	0	0	0	0	-1
0	$y_1 = 12$	2	0	1	0	0	5	-5
0	$y_2 = 8$	3	0	0	1	0	2	-2
0	$y_3 = 3$	1	0	0	0	1	0	0
0	$x_2 = 1$	0	1	0	0	0	-1	1
	$z = 0$	0	0	0	0	0	0	1

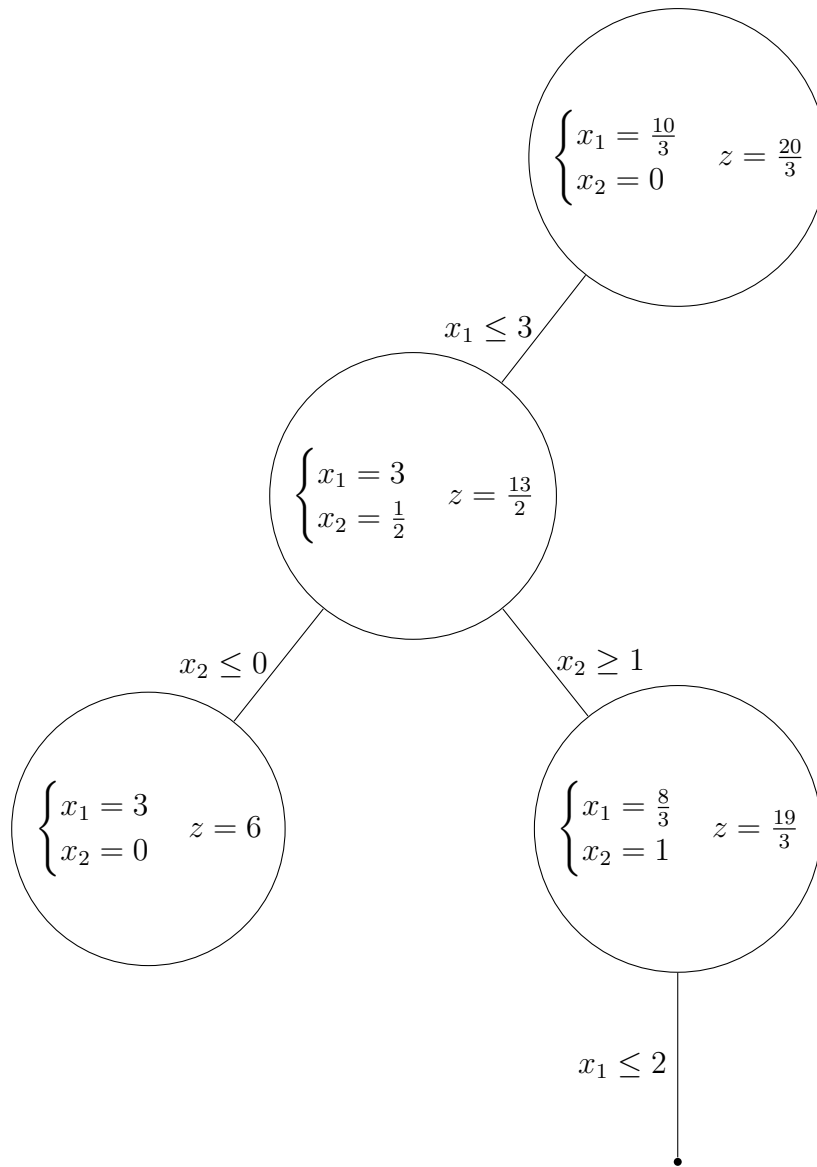
$z = 0$  donc la phase 1 du simplexe est finie. Passons à la phase 2...

		2	1	0	0	0	0
0	$y_1 = 12$	2	0	1	0	0	5
0	$y_2 = 8$	3	0	0	1	0	2
0	$y_3 = 3$	1	0	0	0	1	0
1	$x_2 = 1$	0	1	0	0	0	-1
	$z = 1$	-2	0	0	0	0	-1

$y_2$  sort de la base ;  $x_1$  rentre dans la base ; le pivot devient  $a_{1,2} = 3$ .

		2	1	0	0	0	0
0	$y_1 = \frac{20}{3}$	0	0	1	$-\frac{2}{3}$	0	$\frac{11}{3}$
2	$x_1 = \frac{8}{3}$	1	0	0	$\frac{1}{3}$	0	$\frac{2}{3}$
0	$y_3 = \frac{1}{3}$	0	0	0	$-\frac{1}{3}$	1	$-\frac{2}{3}$
1	$x_2 = 1$	0	1	0	0	0	-1
	$z = \frac{19}{3}$	0	0	0	$\frac{2}{3}$	0	$\frac{1}{3}$

Nous ajoutons la solution obtenue à l'arbre de branch and cut. Comme dans cette solution,  $x_1$  n'est pas entier, nous ajoutons également la contrainte :  $x_1 \leq 2$ .



Nous lançons alors la résolution de ce nouveau problème par le simplexe.

		2	1	0	0	0	0
0	$y_1 = 12$	2	0	1	0	0	5
0	$y_2 = 8$	3	0	0	1	0	2
0	$y_3 = 2$	1	0	0	0	1	0
1	$x_2 = 1$	0	1	0	0	0	-1
	$z = 1$	-2	0	0	0	0	-1

$y_3$  sort de la base ;  $x_1$  rentre dans la base ; le pivot devient  $a_{1,3} = 1$ .

		2	1	0	0	0	0
0	$y_1 = 8$	0	0	1	0	-2	5
0	$y_2 = 2$	0	0	0	1	-3	2
2	$x_1 = 2$	1	0	0	0	1	0
1	$x_2 = 1$	0	1	0	0	0	-1
	$z = 3$	0	0	0	0	2	-1

$y_2$  sort de la base ;  $y_4$  rentre dans la base ; le pivot devient  $a_{6,2} = 2$ .

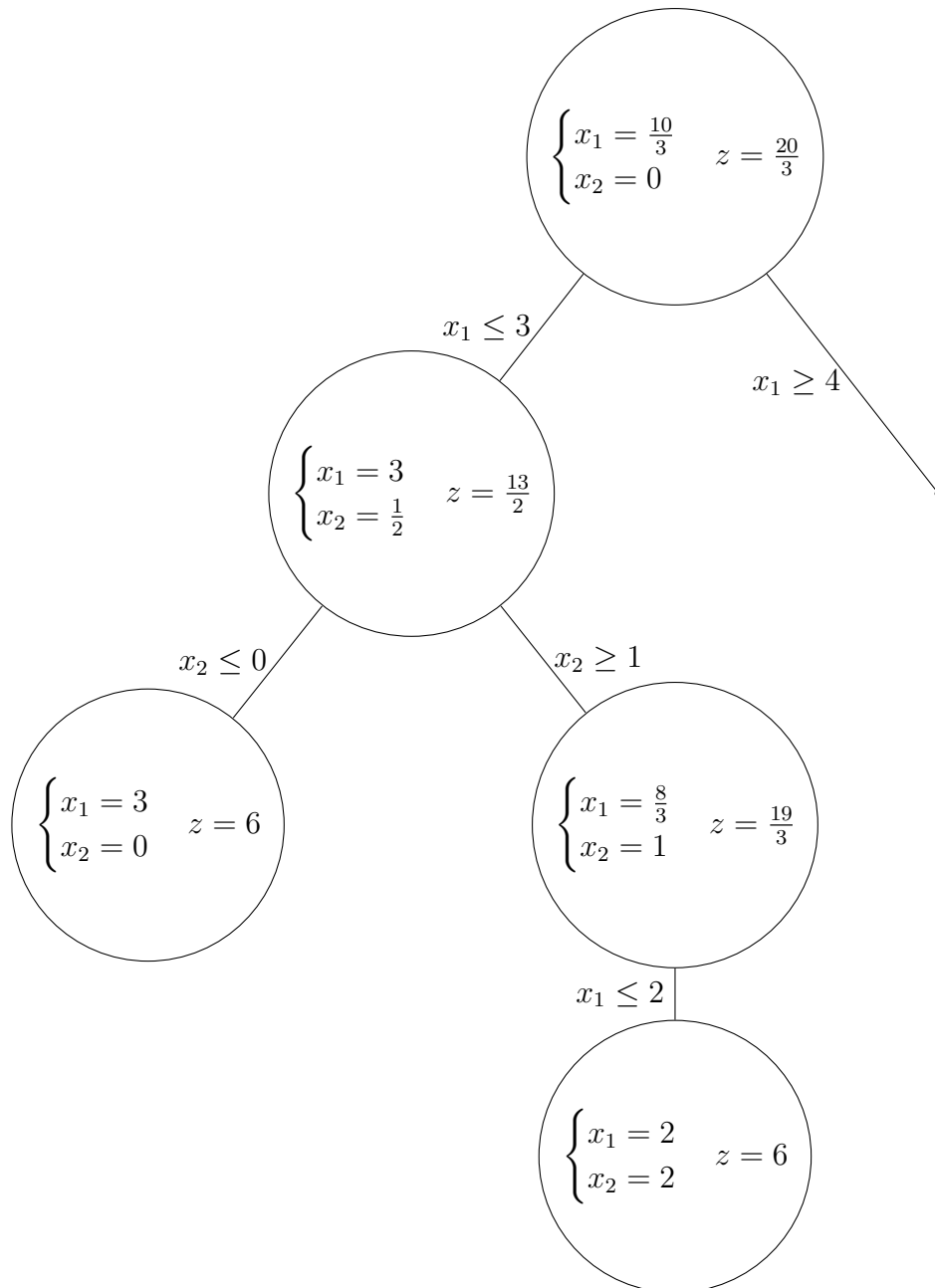
		2	1	0	0	0	0
0	$y_1 = 3$	0	0	1	$-\frac{5}{2}$	$\frac{11}{2}$	0
0	$y_4 = 1$	0	0	0	$\frac{1}{2}$	$-\frac{3}{2}$	1
2	$x_1 = 2$	1	0	0	0	1	0
1	$x_2 = 2$	0	1	0	$\frac{1}{2}$	$-\frac{3}{2}$	0
	$z = 6$	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$	0

La résolution par le simplexe est finie et nous obtenons une solution entière égale à celle précédemment trouvée. Cette solution est donc une feuille de l'arbre ; nous remontons d'un niveau et ajoutons la contrainte  $x_1 \geq 4$ .

Nous remarquons que cette contrainte force  $x_1$  à violer la deuxième contrainte du problème originel :  $3x_1 + 2x_2 \leq 10$ . En effet  $3 \times 4 > 10$ , cette contrainte ne mène donc à aucune nouvelle solution et l'arbre de branch and cut est terminé.

Nous avons donc trouvé deux solutions optimales entières au problème :

$$\begin{cases} x_1 = 2 \\ x_2 = 2 \\ z = 6 \end{cases} \quad \text{et} \quad \begin{cases} x_1 = 3 \\ x_2 = 0 \\ z = 6 \end{cases}$$



### Coupes de Gomory

La méthode des coupes de Gomory nécessite une solution optimale réelle, nous partons donc de celle trouvée précédemment :

$$\begin{cases} x_1 = \frac{10}{3} \\ x_2 = 0 \\ z = \frac{20}{3} \end{cases}$$

et du dernier tableau du simplexe associé :

		2	1	0	0
0	$y_1 = \frac{31}{3}$	0	$\frac{11}{3}$	1	$-\frac{2}{3}$
2	$x_1 = \frac{10}{3}$	1	$\frac{2}{3}$	0	$\frac{1}{3}$
	$z = \frac{20}{3}$	0	$\frac{1}{3}$	0	$\frac{2}{3}$

Nous avons choisi la deuxième ligne du tableau pour en déduire la contrainte de Gomory suivante :

$$\begin{aligned} < \frac{2}{3} > x_2 + < \frac{1}{3} > y_2 \geq < \frac{10}{3} > \\ \Leftrightarrow \\ \frac{2}{3}x_2 + \frac{1}{3}y_2 \geq \frac{10}{3} \end{aligned}$$

Nous devons maintenant l'exprimer uniquement en fonction de  $x_1$  et  $x_2$ . Remplaçons donc  $y_2$  par son équivalent dans la contrainte 2, cela donne :

$$x_1 \leq 3$$

Nous devons maintenant utiliser le simplexe pour résoudre le problème avec la nouvelle contrainte. L'ayant déjà fait lors de la question précédente, sautons directement au résultat :

$$\begin{cases} x_1 = 3 \\ x_2 = \frac{1}{2} \\ z = \frac{13}{2} \end{cases}$$

et le dernier tableau associé :

		2	1	0	0	0
0	$y_1 = \frac{17}{2}$	0	0	1	$-\frac{5}{2}$	2
1	$x_2 = \frac{1}{2}$	0	1	0	$\frac{1}{2}$	$-\frac{3}{2}$
2	$x_1 = 3$	1	0	0	0	1
	$z = \frac{13}{2}$	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$

Nous choisissons la ligne 2 du tableau pour exprimer la contrainte de Gomory :

$$\begin{aligned} < \frac{1}{2} > y_2 + < -\frac{3}{2} > y_3 \geq < \frac{1}{2} > \\ \Leftrightarrow \\ \frac{1}{2}y_2 + \frac{1}{2}y_3 \geq \frac{1}{2} \end{aligned}$$

Nous devons maintenant l'exprimer uniquement en fonction de  $x_1$  et  $x_2$ . Remplaçons donc  $y_2$  par son équivalent dans la contrainte 2, cela donne :

$$x_2 \leq 2y_3$$



Remplaçons maintenant  $y_3$  par son équivalent dans la contrainte 3, cela donne :

$$2x_1 + x_2 \leq 6$$

Réolvons le problème avec cette contrainte en plus :

		2	1	0	0	0	0
0	$y_1 = 17$	2	5	1	0	0	0
0	$y_2 = 10$	3	2	0	1	0	0
0	$y_3 = 3$	1	0	0	0	1	0
0	$y_4 = 6$	2	1	0	0	0	1
	$z = 0$	-2	-1	0	0	0	0

$y_4$  sort de la base ;  $x_1$  rentre dans la base ; le pivot devient  $a_{1,4} = 2$ .

		2	1	0	0	0	0
0	$y_1 = 11$	0	4	1	0	0	-1
0	$y_2 = 1$	0	$\frac{1}{2}$	0	1	0	$-\frac{3}{2}$
0	$y_3 = 0$	0	$-\frac{1}{2}$	0	0	1	$-\frac{1}{2}$
2	$x_1 = 3$	1	$\frac{1}{2}$	0	0	0	$\frac{1}{2}$
	$z = 6$	0	0	0	0	0	1

Et voici donc la solution finale de la méthode des coupes de Gomory :  $\begin{cases} x_1 = 3 \\ x_2 = 0 \\ z = 6 \end{cases}$  .

## Chapitre 2

# Partie pratique