# Chapitre 6 : programmation linéaire en nombres entiers

Dans de nombreux problèmes pratiques, les valeurs des variables de décision sont contraintes de ne prendre que des valeurs entières.

- ✓ Par exemple, pour minimiser la main-d'œuvre nécessaire dans un projet, le nombre des ouvriers doivent être une valeur entière

La première solution intuitive et l'arrondissement de la valeur. Mais, en arrondissant une valeur réelle à une valeur entière, plusieurs problèmes fondamentaux surgissent comme

- ✓ Les solutions arrondies peuvent ne pas être réalisables
- ✓ Même si les solutions sont réalisables, la fonction objectif donnée par le les solutions arrondies peuvent ne pas être les meilleures
- ✓ Enfin, même si les deux conditions ci-dessus sont satisfaites, vérifier toutes les solutions arrondies sont coûteuses en calcul ($2^n$ solutions possibles à considérer pour un problème de n variable)

Pour pallier ce problème, une nouvelle technique de résolution exacte est nécessaire qu'on appelle la programmation entière

## 6.2 Les types des problèmes de la programmation entière

- **Programmation entière pure :**

Toutes les variables sont limitées pour prendre uniquement des valeurs entières

- **Programmation discrète :**

Toutes les variables sont limitées pour ne prendre que valeurs discrètes.

- **Programmation mixte en nombres entiers ou discrets :**

Seules certaines variables sont limitées à prendre des nombres entiers ou valeurs discrètes

- **Programmation en nombres binaire (0-1 programmation)**

Les variables sont contraintes de prendre des valeurs de zéro ou 1

## 6.3 programmation linéaire en nombres entiers

C'est une extension de la programmation linéaire où les variables de décision doivent prendre des valeurs entières.

**Exemple 6.1**

Prenons l'exemple d'un fabricant d'additifs alimentaires qui produit des mélanges d'additifs pour les athlètes. Dans notre exemple, le mélange alimentaire contient deux ingrédients actifs. D'un autre côté, le sachet de ce mélange alimentaire doit contenir une quantité minimale de chacun des quatre éléments nutritifs ci-dessous

| Nutriment | A | B | C | D |
|---|---|---|---|---|
| Milligramme | 9000 | 5000 | 2000 | 200 |

Les ingrédients sont achetés sous comprimés indivisibles dont les valeurs nutritives et le coût suivants :

| | A | B | C | D | Cout/ comprimé |
|---|---|---|---|---|---|
| Ingrédient 1(Milligramme/comprimé) | 100 | 80 | 40 | 10 | 40 |
| Ingrédient 2(Milligramme/ comprimé) | 200 | 150 | 20 | - | 60 |

Quelles devraient être les quantités de comprimés d'ingrédients actifs de de mélange alimentaire afin de minimiser le cout de fabrication ?

Le modèle mathématique de ce problème est le suivant :

**Variables**

Afin de résoudre ce problème, il est préférable de penser en termes d'un sachet de mélange alimentaire. Ce sachet est composé de trois parties - l'ingrédient 1, l'ingrédient 2 , donc :

x1 = nombre de comprimé d'ingrédient 1 dans un sachet de mélange alimentaire

x2 = nombre de comprimé d'ingrédient 2 dans un sachet de mélange alimentaire

où x1 , x2 $\in IN^2$ .

**Contraintes**

Contraintes nutritionnelles

100x1 + 200x2> = 9000 (nutriment A)

80x1 + 150x2> = 5000 (nutriment B)

40x1 + 20x2> = 2000 (nutriment C)

10x1> = 200 (nutriment D)

**Objectif**

Vraisemblablement pour minimiser les coûts, c.-à-d.

Minimiser 40x1 + 60x2

**Exemple 6.2 :**

Supposons que nous souhaitons investir 14 000 $. Nous avons identifié quatre opportunités d'investissement. L'investissement 1 nécessite un investissement de 5000 $ et a un gain de 8 000 $; l'investissement 2 nécessite 7 000 $ et a gain de 11 000 $; l'investissement 3 nécessite 4 000 $ et a un gain de 6 000 $; et l'investissement 4 nécessite 3 000 $ et a un gain de 4 000 $. Dans lequel des Investissements devrions-nous placer notre argent de manière à maximiser notre gain total?

Comme en programmation linéaire, notre première étape est de décider de nos variables.
Cela peut être beaucoup plus difficile en programmation en nombres entiers car il y a des moyens très intelligents pour utiliser les restrictions d'intégrité. Dans ce cas, nous utiliserons une 0-1 variable $x_j$ pour chaque investissement. Si $x_j$ est égal à 1, nous ferons l'investissement j. Si c'est 0, nous ne ferons pas l'investissement. Cela conduit au problème de la programmation en 0-1 :

Le modèle mathématique de ce problème est le suivant :
Maximiser $8x_1 + 11x_2 + 6x_3 + 4x_4$
Sc.
$5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14$
$x_j \in \{0,1\}$ $j = 1, \ldots 4$

C'est un problème linéaire en nombres binaires

### 5.4 La résolution d'un problème linéaire en nombres entiers

Plusieurs techniques ont été développées pour résoudre les problèmes de la programmation linéaires en nombres entiers. Parmi ces techniques on peut citer

- La méthode branch and bound (par évaluation et séparation progressive)
- La méthode des plans coupants
- La génération des colonnes

On s'intéresse plus particulièrement à la méthode branch and bound. Cette méthode n'est pas une technique de résolution spécifiquement limitée aux problèmes de programmation entière. C'est une *approche de résolution* qui peut être appliquée à un certain nombre de types de problèmes. L'approche branch and bound est basée sur le principe que :

L'ensemble total de solutions réalisables peut être divisé en sous-ensembles de solutions plus petits. Celles-ci des sous-ensembles plus petits peuvent alors être évalués systématiquement jusqu'à ce que la meilleure solution soit trouvée.
Contrairement, à la méthode purement énumérative, cette technique utilise des heuristiques pour s'écarter des solutions incomplètes qui ne conduisent pas à la solution que l'on souhaite. De ce fait, on arrive généralement à obtenir la solution recherchée en des temps raisonnables. Bien entendu, dans le pire cas, on retombe toujours sur l'élimination explicite de toutes les solutions du problème. Pour appliquer la méthode de branch-and-bound, nous devons être en possession :

- Le calcul d'une borne inférieure (LB): la borne inférieure permet d'éliminer certaines solutions de l'arbre de recherche afin de faciliter l'exploration du reste de l'arbre et de faire converger la recherche vers la solution optimale.

- Le calcul d'une borne supérieure (UB): la borne supérieure peut être une heuristique qui permet de couper certaines branches de l'arbre de recherche.
- Utilisation des règles spécifiques au problème : certains problèmes possèdent des contraintes à satisfaire pour n'importe quelle solution réalisable, il est possible d'établir des règles qui permettent d'élaguer des branches de l'arbre de recherche.
- Stratégie de recherche : Il existe plusieurs techniques pour l'exploration de l'arbre de recherche. On peut utiliser soit une exploration en profondeur d'abord (Depth First Search -DFS), soit une exploration en largeur d'abord (Breadth First Search-BFS), soit encore une recherche qui utilise une file de priorité.
- Choix de la fonction d'évaluation des nœuds : Lors d'une application de branch and bound, on a besoin d'avoir une fonction qui permet d'évaluer chaque nœud traité afin de couper les branches inutiles. De même, un nœud peut être coupé dans trois cas possibles :
    - Dans le premier cas, on arrive à un stade où la valeur de la borne inférieure d'un nœud courant est plus grande ou égale à la valeur de la borne supérieure qu'on avait établie auparavant.
    - Dans le deuxième cas, la solution n'est pas réalisable, l'un des critères d'évaluation n'a pas été respecté.
    - Le troisième cas est le cas où on a obtenu une solution réalisable, tous les critères sont valides, mais la solution obtenue est supérieure à la borne inférieure.

Quand l'approche Branch and Bound est appliquée à un problème de programmation en nombres entiers, elle est utilisée en conjonction avec l'approche de programmation linéaire non entière normale. Nous présentons cette méthode à l'aide de l'exemple suivant :

**Exemple :** Le propriétaire d'un atelier d'usinage prévoit l'agrandissement de ses activités en achetant de nouveaux machines : presses et tours. Le propriétaire a estimé que chaque presse achetée augmentera le profit de 100$ par jour et chaque tour augmentera le profit de 150$ par jour. Le nombre de machines que le propriétaire peut acheter est limité par le coût des machines et la surface au sol disponible dans le magasin. Les prix d'achat de la machine et les exigences d'espace sont comme suit

| Machine | Espace demandée (m2) | Prix d'achat |
|---|---|---|
| Presse | 15 | 8000$ |
| Tour | 30 | 4000$ |

Le propriétaire a un budget de 40000 $ pour l'achat de machines et 200 mètres carrés de surface disponible. Le propriétaire veut savoir combien de chaque type de machine à acheter pour maximiser l'augmentation quotidienne des bénéfices.

Le modèle de la programmation linéaire pour un problème de programmation entière est formulé exactement de la même manière que les exemples de programmation linéaire vu dans le chapitre de la programmation linéaire. La seule différence est que dans ce problème, les variables de décision ont des valeurs entières car le propriétaire ne peut pas acheter une fraction, ou une partie, d'une machine. Le modèle de programmation linéaire en nombres entiers est comme suit :

$\max Z = 100x_1 + 150x_2$

SC

$8{,}000x_1 + 4{,}000x_2 \leq 40{,}000$ ( budjet)
$15x_1 + 30x_2 \leq 200$ ( surface)
$x_1, x_2 \geq 0$ et entiers ( ou $x_1, x_2 \in IN^2$)

où

$x_1$ = nombre des machines presses
$x_2$ = nombre des machines tours

Les variables de décision dans ce modèle représentent des nombres de machines entières. Le fait que les deux variables de décision, $x_1$ et $x_2$, peut supposer que toute valeur entière supérieure ou égale à zéro est ce qui donne à ce modèle sa désignation de modèle entier total.

Avant de résoudre ce problème, nous présentons dans ce qui suit les étapes de l'algorithme de Branch and bound pour la programmation linéaire en nombres entiers.

**<u>Algorithme Méthode</u> de résolution générale d'un problème en PLNE par B&B**

1. Initialisation :
   - Calculer un LB par une heuristique qui donne des solutions entières
   - Calculer un UB par la méthode du simplex

2. choisir une variable xi et une constante xi* et ajouter la contrainte suivante $xi \geq xi^*+1$ ($xi \leq xi^*$)
3. diviser le problème selon ces deux contraintes ( voir exemple)
2. Pour chaque noeud i calculer UBi de ce noeud par la méthode du simplexe

Si (UBi < LB)
  Séparer ce noeud
Sinon (si UBi est fractionné)
Aller à 2
Sinon (si UBi est entier)
Mettre LB=UBi
Aller à 2

Nous commençons la méthode Branch and bound en résolvant d'abord le problème comme un problème linéaire régulier où les contraintes d'intégralité sont enlevées (c'est-à-dire que les contraintes d'integrité sont relaxées).

Le modèle de programmation linéaire pour le problème et la solution relaxée optimale est

$$\text{maximize } Z = 100x_1 + 150x_2$$
$$\text{s.c}$$
$$8{,}000x_1 + 4{,}000\,x_2 \leq 40{,}000$$
$$15x_1 + 30x_2 \leq 200$$
$$x_1, x_2 \geq 0$$

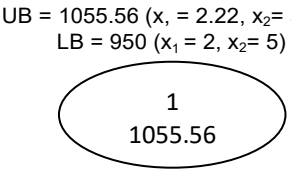
Cette opération est connue sous le nom « **relaxation linéaire** »

La solution donnée par le simplexe

$$x_1 = 2.22,\ x_2 = 5.56, \text{ and } Z = 1055.56\ \$$$

On voit clairement que cette solution n'est pas acceptable vu que $x_1$, $x_2$ sont non entiers

La méthode Branch and bound utilise un arbre composé de nœuds et de branches pour la résolution du problème. Le premier nœud de l'arbre illustré à la figure suivante contient la solution de la programmation linéaire **relaxée** trouvée précédemment. Nous mettons la borne supérieur UB= 1055.56 parce qu'il n y'aura pas une solution meilleure que celle-ci, et nous mettons LB=950 la valeur de la solution arrondie, et la solution optimale entière sera entre ces deux bornes.




Maintenant que les solutions réalisables possibles ont été réduites à des valeurs entre les limites supérieure et inférieure, nous devons tester les solutions dans ces limites pour déterminer la meilleure. La première étape de la méthode de Branch and bound consiste à créer *deux* sous-ensembles de solution de la solution relaxée actuelle. Ceci est accompli en observant la solution relaxée

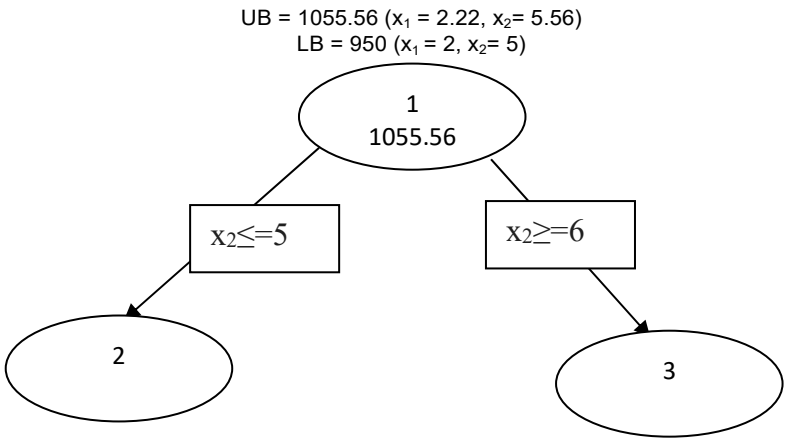
$x_1^* = 2.22,\ x_2^* = 5.56$

Les deux variables sont fractionnées, alors on va choisir une, mais laquelle ?
L'heuristique ici est de choisir la variable ayant la plus grande fraction, ici c'est la variable $x_2$. Mais, le choix de n'importe quelle variable va donner une solution optimale, la chose qui change est le temps d'exécution.

Donc on fait un branchement sur la variable $x_2$. Parce que $x_2$ doit être une valeur entière dans la solution optimale, les contraintes suivantes peuvent être développées.
$x_2 \leq \text{int}(x_2^*)+1 \Rightarrow x_2 \geq 6$
$x_2 \geq \text{int}(x_2^*) \Rightarrow x_2 \leq 5$

Ces deux nouvelles contraintes représentent les deux sous-ensembles de solutions pour notre approche de résolution. Chacune de ces contraintes sera ajoutée séparément à notre modèle, ce qui donnera deux

autres modèle qui seront ensuite résolus normalement par la méthode du simplexe pour déterminer une solution relaxée. Sur l'arbre de résolution on obtient deux nouveaux nœuds.




D'abord, la résolution du nœud 2 consiste à résoudre le système suivant :

maximize $Z = 100x_1 + 150x_2$

s.c

$8{,}000x_1 + 4{,}000\ x_2 \leq 40{,}000$

$15x_1 + 30x_2 \leq 200$

$x_2 \leq 5$

$x_1, x_2 \geq 0$

La solution optimale pour ce problème est $x_1 = 2.5$ , $x^2 = 5$, et $z = 1000$.

Ensuite, la solution du nœud 3 est trouvée par la résolution du modèle avec la contrainte $x2 \geq 6$

maximize $Z = 100x_1 + 150x_2$

s.c

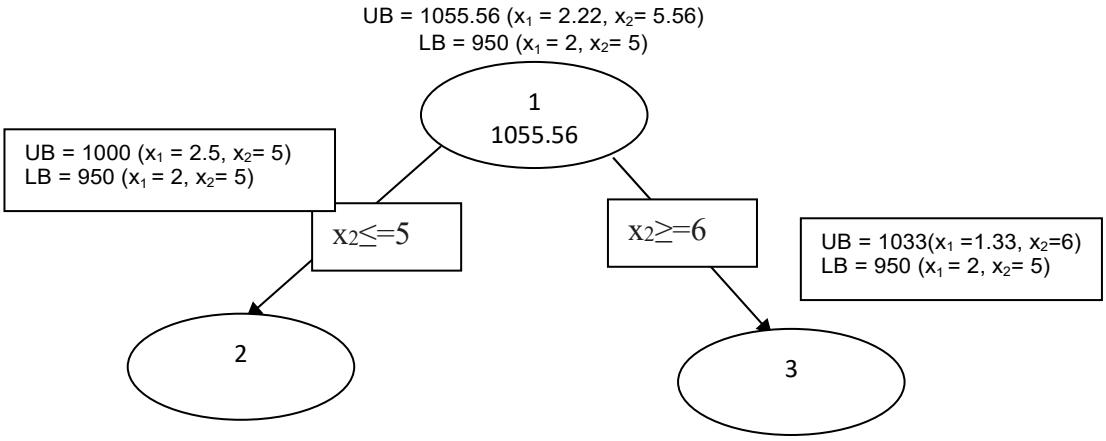
$8{,}000x_1 + 4{,}000\ x_2 \leq 40{,}000$

$15x_1 + 30x_2 \leq 200$

$x_2 \geq 6$

$x_1, x_2 \geq 0$

La solution optimale pour ce modèle est x1=1.33 x2=6, et z=1033.33




Vu qu'on n'a pas encore trouvé une solution optimale, nous continuons le branchement. Nous pouvons choisir le nœud 2 ou le nœud 3 pour faire ce branchement. Cependant si on branche le nœud 2, il est possible que la plus grande valeur de la fonction objective 1000, alors pour le nœud 3 elle est 1033. Donc on va choisir le nœud 3 pour le branchement.
On va répéter le même traitement du nœud 1 pour le nœud 3. On a une seule variable fractionnée $x_1^* = 1.33$

Donc on va créer deux sous problèmes en ajoutant respectivement les deux contraintes suivantes :
$X_1 \leq int(x_1^*)+1 \Rightarrow x_1 \geq 2$
$X_1 \geq int(x_1^*) \Rightarrow x_1 \leq 1$

Le modèle du nœud 4

maximize $Z = 100x_1 + 150x_2$

s.c

$8{,}000x_1 + 4{,}000\, x_2 \leq 40{,}000$

$15x_1 + 30x_2 \leq 200$

$x_2 \geq 6$

$x_1 \leq 1$

$x_1, x_2 \geq 0$

La solution optimale pour ce modèle est $x_1=1$, $x_2=6.17$, et $z=1025$.

Le modèle du nœud 5

maximize $Z = 100x_1 + 150x_2$

s.c

$8{,}000x_1 + 4{,}000\, x_2 \leq 40{,}000$

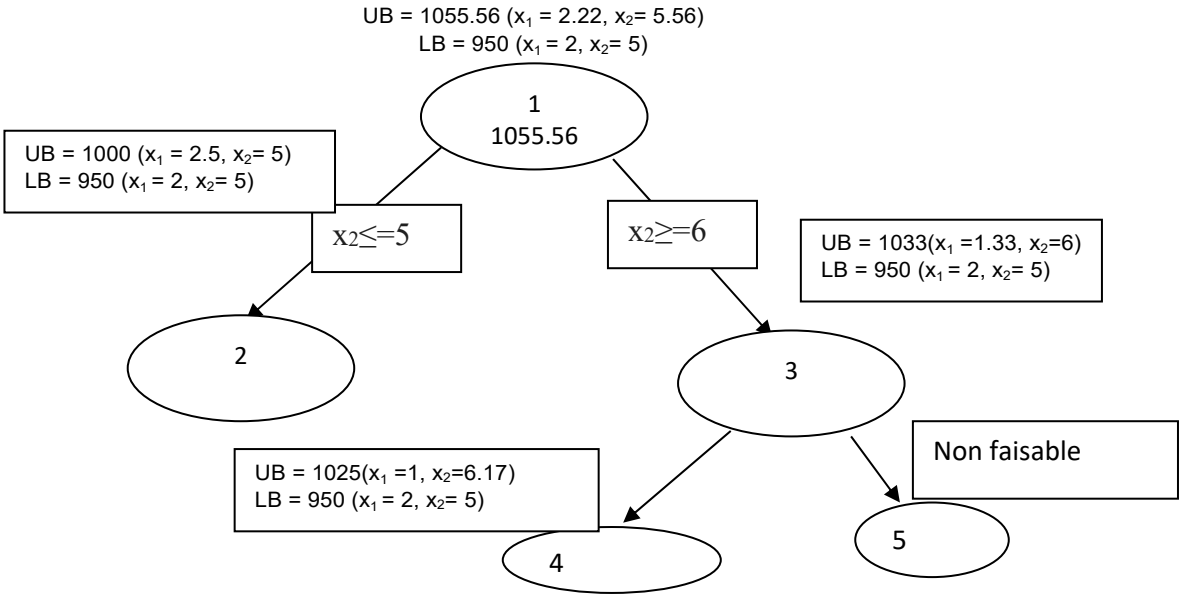
$15x_1 + 30x_2 \leq 200$

$x_2 \geq 6$

$x_2 \geq 2$

$x_1, x_1 \geq 0$

La résolution du modèle du nœud 5 donne une solution non faisable




Le nœud 4 ne peut pas être final parce que le UB>LB et il y a des variables fractionnées
Donc, on refait le travail pour le nœud 4. Ainsi, on trouve deux sous problèmes selon les nouvelles contraintes crées à partir de la variable $x_2^*=6.17$.

$x_2 \leq int(x_2^*)+1 \Rightarrow x_2 \geq 7$
$x_2 \geq int(x_2^*) \Rightarrow x_2 \leq 6$

Le modèle du nœud 6

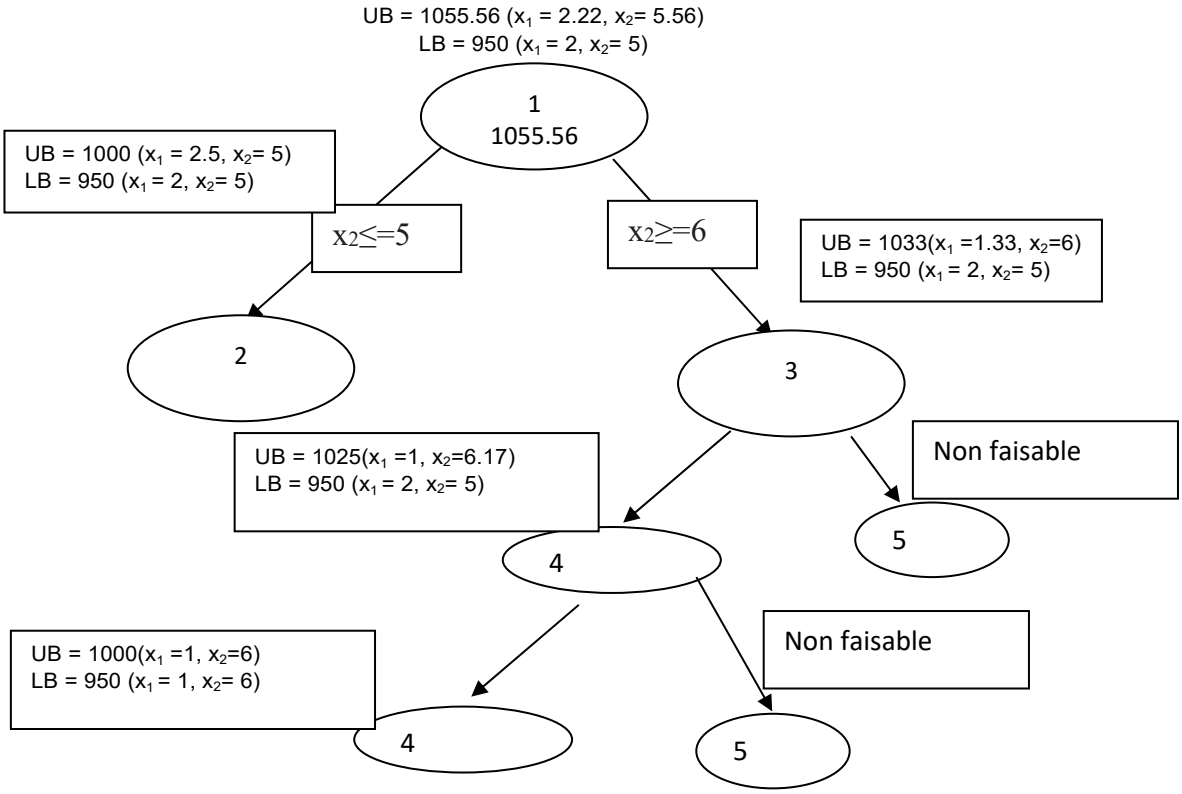
$$\text{maximize } Z = 100x_1 + 150x_2$$
$$\text{s.c}$$
$$8{,}000x_1 + 4{,}000x_2 \leq 40{,}000$$
$$15x_1 + 30x_2 \leq 200$$
$$x_2 \geq 6$$
$$x_1 \leq 1$$
$$x_2 \leq 6$$
$$x_1, x_2 \geq 0$$

La solution optimale pour ce modèle est $x_1=1$, $x_2=6$, et $z=1000$.
Nous remarquons que la solution est entière est que UB6>LB alors le nouveau LB devient 1000, et ce nœud devient une feuille.

Le modèle du nœud 7

$$\text{maximize } Z = 100x_1 + 150x_2$$
$$\text{s.c}$$
$$8{,}000x_1 + 4{,}000x_2 \leq 40{,}000$$
$$15x_1 + 30x_2 \leq 200$$
$$x_2 \geq 6$$
$$x_1 \leq 1$$
$$x_2 \geq 7$$
$$x_1, x_2 \geq 0$$

La résolution du modèle du nœud 5 donne une solution non faisable




Le seul nœud qui reste à explorer est le nœud 2, mais sa borne supérieure UB2=1000≤LB=1000 alors on coupe cette branche et ainsi l'algorithme termine et il nous retourne la solution optimale ($x_1$ =1, $x_2$=6) dont la valeur z=1000.

**Remarques**

- La procédure B&B est un algorithme et non une heuristique : nous avons démontré qu'elle fournit bien la solution optimale du problème qu'elle résout. Bien que généralement, elle donne de bons résultats en terme de vitesse de calcul, il peut arriver qu'elle doive "backtracker" un grand nombre de fois et qu'elle finisse par construire et explorer un arbre gigantesque.
- La programmation linéaire en nombres binaires (0-1 programmation) peuvent également résolu par Branch and bound
- On peut utiliser n'importe qu'elle technique pour calculer la borne supérieure.
- Il existe plusieurs améliorations de l'algorithme branch-and-bound, comme branch-and-cut, branch-and-price que les note généralement par l'expression « *branch and ** ».

**Exercice à faire :** résoudre le *problème de sac à dos* en utilisant la méthode branch and bound

Références :


- Der-San Chen; Robert G. Batson; Yu Dang (2010). Applied Integer Programming: Modeling and Solution. John Wiley and Sons. *ISBN 978-0-470-37306-4*.
- Gerard Sierksma; Yori Zwols (2015). Linear and Integer Optimization: Theory and Practice. CRC Press. *ISBN 978-1-498-71016-9*.