La méthode de branch and bound

1. Introduction

Pour plusieurs problèmes, et en particulier les problèmes d'optimisation combinatoire, l'ensemble de leurs solutions est fini et dénombrable. Il est donc possible, en principe, d'énumérer toutes ces solutions, ensuite prendre celle qui nous arrange. Malheureusement il n'est guère évident d'effectuer cette énumération à cause du nombre prohibitif de solutions.

La méthode de branch and bound (procédure par séparation et évaluation) est une méthode générique de résolution des problèmes d'optimisation combinatoire. Elle procède par énumération de toutes les solutions du problème en question, mais d'une manière intelligente.

2. Principe de la méthode

La procédure par séparation et évaluation s'appuie sur la construction d'une arborescence dont la racine représente l'ensemble de toutes les solutions du problème. Chaque nœud est muni d'une borne inférieure pour un problème de minimisation et d'une borne supérieure pour un problème de maximisation.
Grâce à l'utilisation des bornes, cette méthode se dote de la capacité d'exclure des solutions partielles tôt. Bien entendu, dans le pire cas, on retombe toujours sur l'exploration explicite de toutes les solutions du problème.
Essentiellement la méthode repose sur les points suivants :

- **Construction d'une solution heuristique**

Toute solution représente une borne supérieure pour un problème de minimisation et une borne inférieure pour un problème de maximisation. Donc une solution initiale de bonne qualité, peut servir dans l'élimination de plusieurs branches de l'arborescence et par conséquent réduire l'espace de recherche.

- **Séparation**

Elle permet de construire d'une manière récursive une arborescence en partitionnant chaque ensemble en plusieurs sous-ensembles généralement disjoints. La racine représente l'ensemble de toutes les solutions du problème. Chaque nœud interne représente une solution partielle et chaque feuille représente une solution totale.

- **Evaluation**

L'évaluation consiste à associer pour chaque nœud crée, une borne inférieure (LB) pour un problème de minimisation et une borne supérieure (UB) pour un problème de maximisation. Inversement on doit avoir pour chaque problème au moins une borne supérieure(UB) pour un problème de minimisation et une borne inférieure (LB) pour un problème de maximisation. C'est en confrontant les bornes inférieures avec les bornes supérieures qu'on arrive à élaguer certaines branches de l'arborescence et donc profiter d'avantage. En effet, pour un problème de minimisation si la borne inférieure est supérieure ou égale à la borne supérieure, il est impossible de trouver dans cette branche une solution mieux que la meilleure solution candidate.

- **Exploration**

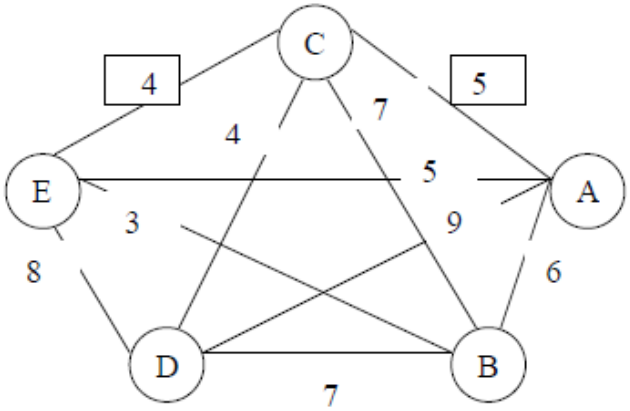
L'exploration consiste à fixer un protocole donnant l'ordre de visite des différentes branches comme les méthodes de parcours dans les graphes (l'exploration en profondeur d'abord (DFS) ou en largeur d'abord (BFS)). On peut choisir par exemple de commencer par explorer les branches les plus prometteuses c'est-à-dire celles qui possèdent la borne inférieure la plus petite pour un problème de minimisation. D'une façon générale se sont des heuristiques qui nous guident vers le choix d'une telle ou telle branche.

3. Illustration de la méthode sur quelques problèmes

Le problème du voyageur de commerce

- Soit un graphe G=(V,E). Un cycle est hamiltonien si et seulement si tous les sommets de G apparaissent une et seule fois dans ce cycle.
- Soit un graphe G=(V,E) valué. Le problème du voyageur de commerce consiste à trouver un cycle hamiltonien dont la somme des poids est minimale.

Soit le graphe suivant :



Borne inférieure

Soit le cycle hamiltonien suivant : $v_1, v_2, \ldots, v_n, v_{n+1} = v_1$. Il est facile de monter que son coût est $\geq \frac{1}{2}\sum_{i=1}^{n}(a\text{rête} - v_{i1} + a\text{rête} - v_{i2})$ où $a\text{rête} - v_{i1}$ et $a\text{rête} - v_{i2}$ désignent respectivement deux arêtes adjacentes au sommet i ayant le plus petit poids. Donc cette expression représente une borne inférieure LB.

Soit **E**, le sommet de départ.

$LB_E = \frac{1}{2}\{ (3+4)+(4+4)+(5+5)+(3+6)+(4+7) \} = 22.5$

Les prochains sommets dans le cycle peuvent être : A, B, C ou D. Pour chacune des ces solutions partielles, une borne est calculée.

Par exemple pour le sommet D, $LB_{\{E,D\}} = \frac{1}{2}\{ (8+3)+(4+4)+(5+5)+(3+6)+(4+8) \} = 25$. Car l'arête (E,D) est sélectionnée

On fait la même chose pour les autres sommets, donc on aura:

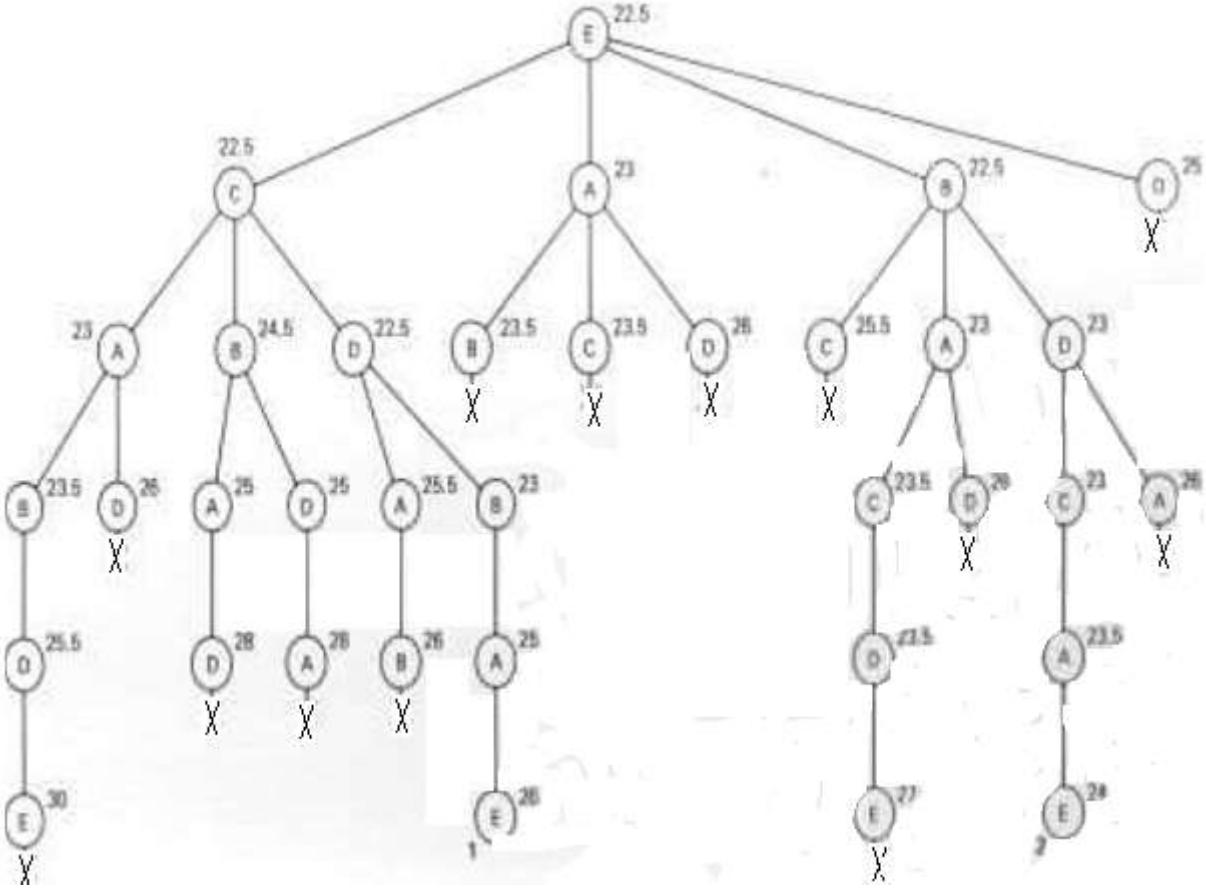
Pour le sommet C, $LB_{\{E, C\}} = 22.5$

Pour le sommet A, $LB_{\{E, A\}} = 23$

Pour le sommet B, $LB_{\{E, B\}} = 22.5$

Parmi ces quatre valeurs, nous allons explorer le sommet ayant la plus petite borne, soit le sommet C, donc les prochains sommets peuvent être : {A, B, D}.
Pour le sommet A, $LB_{\{E, C, A\}} = 23$
Pour le sommet B, $LB_{\{E, C, B\}} = 24.5$
Pour le sommet D, $LB_{\{E, C, D\}} = 22.5$

En gardant le même principe, quand une solution complète est trouvée, la meilleure solution candidate est modifiée si nécessaire. Puisque aucune solution initiale n'est proposée, alors La première solution trouvée qui a pour coût 26, représente la meilleure solution candidate.
En continuant l'exploration, on aura l'arborescence suivante :



Le problème d'affectation

Soient *n* personnes à affecter à *n* tâches. Le coût d'affectation de *i* à la tâche *j* est noté c_{ij}. Le problème consiste à affecter chaque personne à une seule tâche en minimisant le coût total. Bien entendu, une tâche ne peut être affectée qu'à une seule personne.
Soit la matrice des coûts suivante :

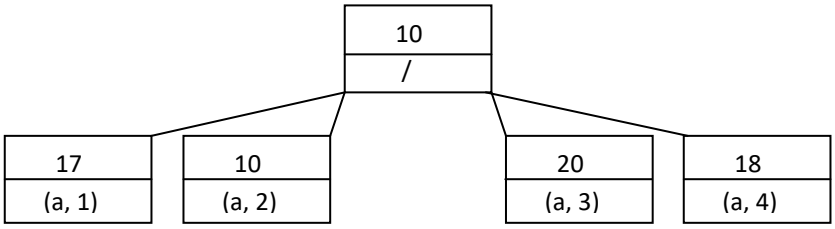
	Tâche 1	Tâche 2	Tâche 3	Tâche 4
Personne a	9	2	7	8
Personne b	6	4	3	7
Personne c	5	8	1	8
Personne d	7	6	9	4

Choisissons au hasard une solution, par exemple on va prendre l'affectation suivante:
(1, b), (2, c), (3, d) et (4, a).
On aura un coût total égal à 6 + 8 + 9 + 8 = 30.
Cette affection représente la meilleure solution candidate de départ pour l'algorithme.

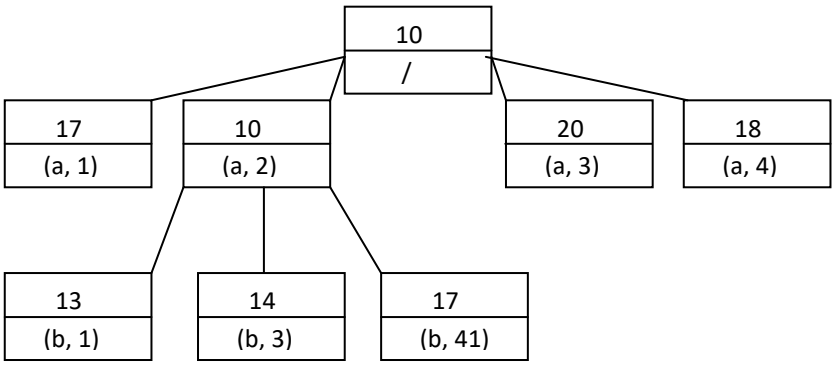
Borne inférieure
Il est clair que, quelque soit la solution, son coût ne peut pas être plus petit que la somme des plus petits éléments de chacune des lignes de la matrice des coûts.

Au départ aucune tache n'est affectée.
$LB_{\{\ \}} = 2 + 3 + 1 + 4 = 10$.

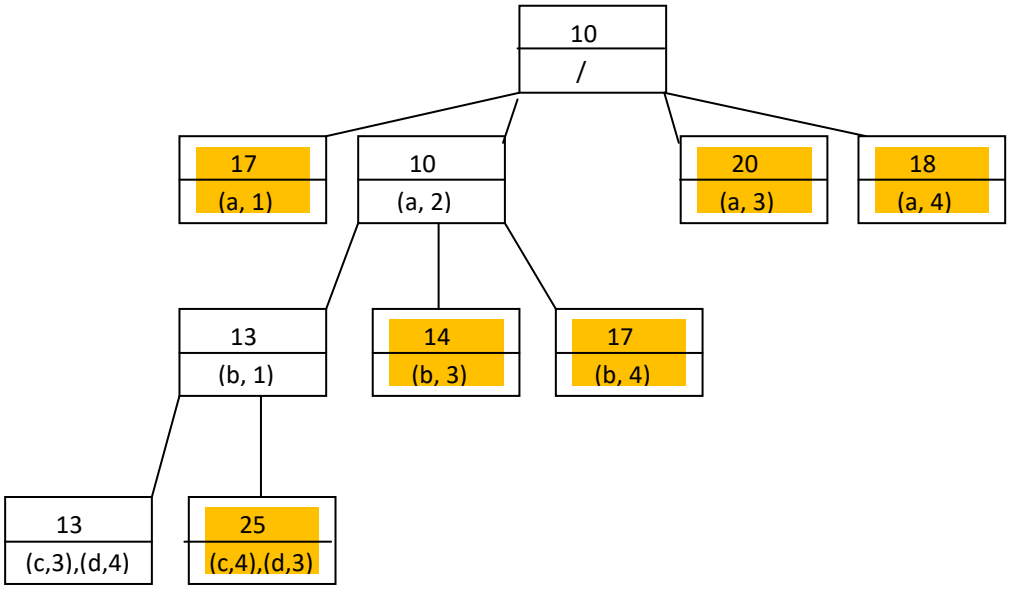
On va affecter la personne à l'une des taches suivantes : {1, 2, 3, 4}
$LB_{\{ (a, 1) \}} = 9 + 3 + 1 + 4 = 17$.
$LB_{\{ (a, 2) \}} = 2 + 3 + 1 + 4 = 10$.
$LB_{\{ (a, 3) \}} = 7 + 4 + 5 + 4 = 20$.
$LB_{\{ (a, 4) \}} = 8 + 3 + 1 + 6 = 18$.



Selon l'heuristique qui choisit le nœud portant la plus petite borne, l'affectation qu'on va sélectionner est : la tache 2 à la personne a.
Donc les taches qu'on peut affecter à la personne b sont : {1, 3, 4}
$LB_{\{ (a, 2), (b, 1) \}} = 2 + 6 + 1 + 4 = 13$.
$LB_{\{ (a, 2), (b, 3) \}} = 2 + 3 + 5 + 4 = 14$.
$LB_{\{ (a, 2), (b, 4) \}} = 2 + 7 + 1 + 7 = 17$.



Selon l'heuristique qui choisit le nœud portant la borne 10, on sélectionne les affectations suivantes : (a, 2) et (b, 1).
Donc les taches qu'on peut affecter à la personne c sont : {3, 4}.
$LB_{\{ (a, 2), (b, 1), (c, 3), (d, 4) \}} = 2 + 6 + 1 + 4 = 13$.
$LB_{\{ (a, 2), (b, 1), (c, 4), (d, 3) \}} = 2 + 6 + 8 + 9 = 25$.
En suivant le même principe, on aura l'arborescence ci-dessous, après élagage des sous arbres en jaune :



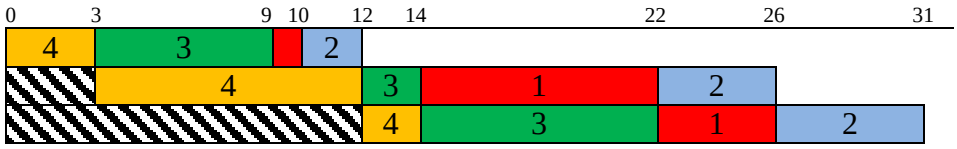
Le problème du Flow shop à trois machines

Soient à exécuter n tâches sur 3 machines. Chaque tâche doit s'exécuter sur la machine 1, ensuite sur la machine 2 et enfin sur la machine 3. Le temps d'exécution de la tâche *i* sur la machine 1 est noté par a_i , sur la machine 2 par b_i et sur la machine 3 par c_i . On désire trouver une permutation d'exécution de ces *n* tâches sur les trois machines de tel manière à minimiser le temps total d'accomplissement, appelé makespan.

Prenons l'exemple suivant de 4 tâches sur 3 machines avec les temps d'exécution suivants :

	a_i	b_i	c_i
Tâche 1	1	8	4
Tâche 2	2	4	5
Tâche 3	6	2	8
Tâche 4	3	9	2

Si on décide d'exécuter les tâches dans l'ordre **4**, **3**, **1** et **2**, alors le temps d'accomplissement de toutes les tâches est égal à **31**, comme illustré par le diagramme de Gantt suivant :



D'une façon générale, soit $A = \{ i(1), i(2),...,i(k) \}$ l'ensemble des premières tâches déjà exécutées dans cet ordre, et U l'ensemble des tâches non encore exécutées, à un instant donné. Pour j = 1, 2,…, k , on pose $\boldsymbol{\alpha}_{i(k)}, \boldsymbol{\beta}_{i(k)}, \boldsymbol{\gamma}_{i(k)}$, les dates de fin de la tâche *i(k)* sur la *machine 1*, *la machine 2* et *la machine 3*, respectivement. Ces trois variables sont calculées récursivement comme suit :

$\boldsymbol{\alpha}_{i(1)} = a_{i(1)}, \quad \boldsymbol{\alpha}_{i(k)} = \boldsymbol{\alpha}_{i(k-1)} + a_{i(k)}$

$\boldsymbol{\beta}_{i(1)} = a_{i(1)} + b_{i(1)}, \quad \boldsymbol{\beta}_{i(k)} = Max\{\boldsymbol{\alpha}_{i(k)}, \boldsymbol{\beta}_{i(k-1)}\} + b_{i(k)}$

$\boldsymbol{\gamma}_{i(1)} = a_{i(1)} + b_{i(1)} + c_{i(1)}, \quad \boldsymbol{\gamma}_{i(k)} = Max\{\boldsymbol{\beta}_{i(k)}, \boldsymbol{\gamma}_{i(k-1)}\} + c_{i(k)}$

Pour calculer une borne inférieure à ce problème, nous allons considérer trois possibilités les plus favorables qui peuvent se présenter. Autrement dit, nous allons déterminer le plus court temps pour exécuter les jobs dans l'ensemble U des tâches non encore exécutées.

Il est clair que dans toute solution, l'exécution des tâches sur la machine 1 est continuelle. Considérons la dernière tâche, disons *i(n)* , d'une solution donnée. Le meilleur qui puisse arriver à cette tâche est de ne pas attendre sur la machine 2 et la machine 3. Autrement dit, le makespan C_{max} est alors :

$$c_{max} = \boldsymbol{\alpha}_{i(k)} + \sum_{i \in U} a_i + \left(b_{i(n)} + c_{i(n)}\right)$$

Par conséquent, si on choisit les plus courts temps d'exécution sur la machine 2 et 3, alors quelque soit la solution, on aura :

$$c_{max} \geq \boldsymbol{\alpha}_{i(k)} + \sum_{i \in U} a_i + min_{i \in U}\{b_i + c_i\}$$

Similairement, en considérant que la machine 2 est continue dans son exécution. On a alors

$$c_{max} = \boldsymbol{\beta}_{i(k)} + \sum_{i \in U} b_i + c_{i(n)}$$

En choisissant le plus court temps d'exécution sur la machine 3, quelque soit la solution, on a alors: $c_{max} \geq \boldsymbol{\beta}_{i(k)} + \sum_{i \in U} b_i + min_{i \in U}\{c_i\}$

Similairement, en considérant que la machine 3 est continue dans son exécution. On obtient la borne suivante : $c_{max} \geq \boldsymbol{\gamma}_{i(k)} + \sum_{i \in U} c_i$

Par conséquent, quelque soit la solution, son makespan ne peut être mieux que la valeur des trois expressions ci-dessus. Autrement dit, nous avons bien:

$$c_{max} \geq \max\{\boldsymbol{\alpha}_{i(k)} + \sum_{i \in U} a_i + min_{i \in U}\{b_i + c_i\}, \quad \boldsymbol{\beta}_{i(k)} + \sum_{i \in U} b_i + min_{i \in U}\{c_i\}, \quad \boldsymbol{\gamma}_{i(k)} + \sum_{i \in U} c_i \}.$$

Passons maintenant à la résolution de notre problème mais en faisant un parcours en profondeur d'abord.

Au départ nous avons une borne supérieure égale à 31.

À la racine de l'arbre, aucune tâche n'est exécutée, la borne inférieure est donc :

$$c_{max} \geq \max\{\alpha_{i(k)} + \sum_{i \in U} a_i + min_{i \in U}\{b_i + c_i\}, \beta_{i(k)} + \sum_{i \in U} b_i + min_{i \in U}\{c_i\}, \gamma_{i(k)} + \sum_{i \in U} c_i \} = \{ 12+9, \ 23+2, \ 19 \} = 25.$$

Autrement dit, aucune solution ne peut avoir un makespan inférieur à 25.

- Si on met la tâche 1 en position 1, on obtient: A = {1}, U = {2, 3, 4}

$\alpha_1 = 1, \quad \beta_1 = 9, \quad \gamma_1 = 13$

borne inférieure = max{ 1+11+ 9, 9 +15 + 2, 13 +15 }= 28 .

- Si on met la tâche 2 en position 2, on obtient: A = {1, 2}, U={3, 4}

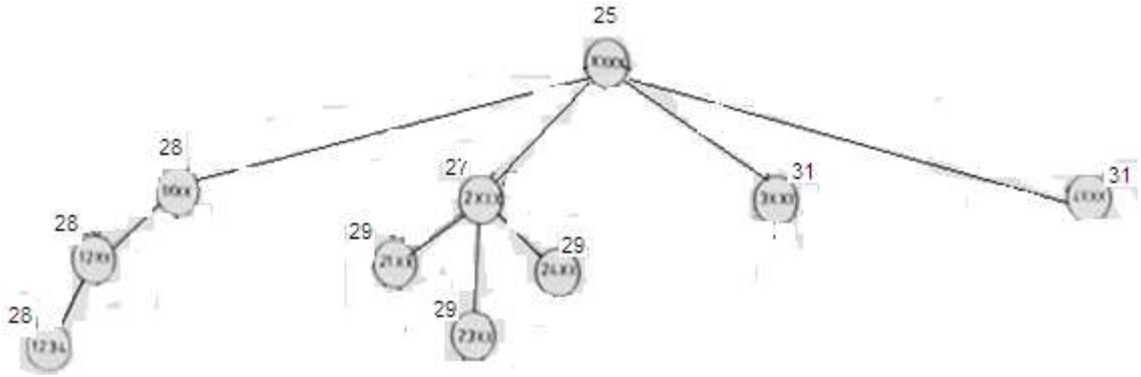
$\alpha_2 = 1 + 2 = 3, \quad \beta_2 = max\{3, 9\} + 4 = 13, \quad \gamma_2 = max\{13, 13\} + 5 = 18$

borne inférieure = max{ 3 + 9 +10, 13 +11+ 2, 18 +10 }= 28 .

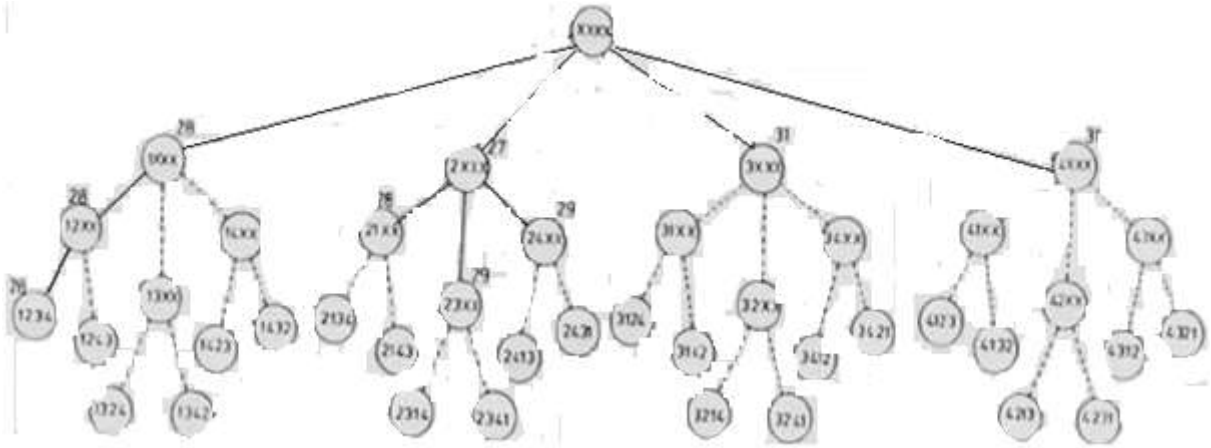
- Si on met la tâche 3 en position 3, on obtient: A = {1, 2, 3}, U={4}
$\alpha_3 = 3 + 6 = 9,$ $\beta_3 = max\{9, 13\} + 2 = 15,$ $\gamma_3 = max\{15, 18\} + 8 = 26$
borne inférieure = max{ 9 +3+11, 15 +9+ 2, 26 +2 }= 28 .

- Si on met la tâche 4 en position 4, on obtient: A = {1, 2, 3, 4}, U={}
$\alpha_4 = 9 + 3 = 12,$ $\beta_4 = max\{12, 15\} + 9 = 24,$ $\gamma_4 = max\{24, 26\} + 2 = 28$

En suivant le même principe et après élagage de certaines branches, on aura l'arborescence ci-dessous avec un makespan = 28.



La figure suivante nous montre l'arborescence complète avec les branches élaguées en pointillés.



Les nœuds créées par des arcs en pointillés sont des nœuds qui n'ont pas été explorés lors de la création de l'arborescence car ne pouvant contenir une solution optimales.

Résolution d'un PLNE

Méthode branch and Bound

Prenons l'exemple du programme linéaire suivant :

$$(P)\begin{cases} Max\ Z = x_1 + 4x_2 \\ 5x_1 + 8x_2 \leq 40 \\ -2x_1 + 3x_2 \leq 9 \\ x_1, x_2 \in N \end{cases}$$

Pour résoudre *(P)* on va suivre les étapes suivantes :

1) Résoudre (*P*) à l'aide du simplexe sans tenir compte de la contrainte x_1, x_2 à valeurs entières. Solution optimale : $x = (x_1, x_2) = (1.55, 4.03)$. $z(x) = 17, 67$.
Séparer par rapport à $x1$: $x_1 \leq 1$ ou $x_1 \geq 2$.

2) Ajouter la contrainte $x_1 \leq 1$ à (*P*) et résoudre ce programme à l'aide du simplexe, sans tenir compte de la contrainte x_1, x_2 à valeurs entières. Solution optimale $x =$ $(1, 3.67)$, $z(x) = 15.67$.
Séparer par rapport à x_2 : $x_2 \leq 3$ ou $x_2 \geq 4$.

3) Ajouter à (*P*) les contraintes $x_1 \leq 1$ et $x_2 \leq 3$. La solution optimale de ce programme est à valeurs entières : $x = (1, 3)$, $z(x) = 13$.
Evaluer à 13 (c'est-à-dire ne pas poursuivre le branchement si on obtient des solutions optimales < 13).

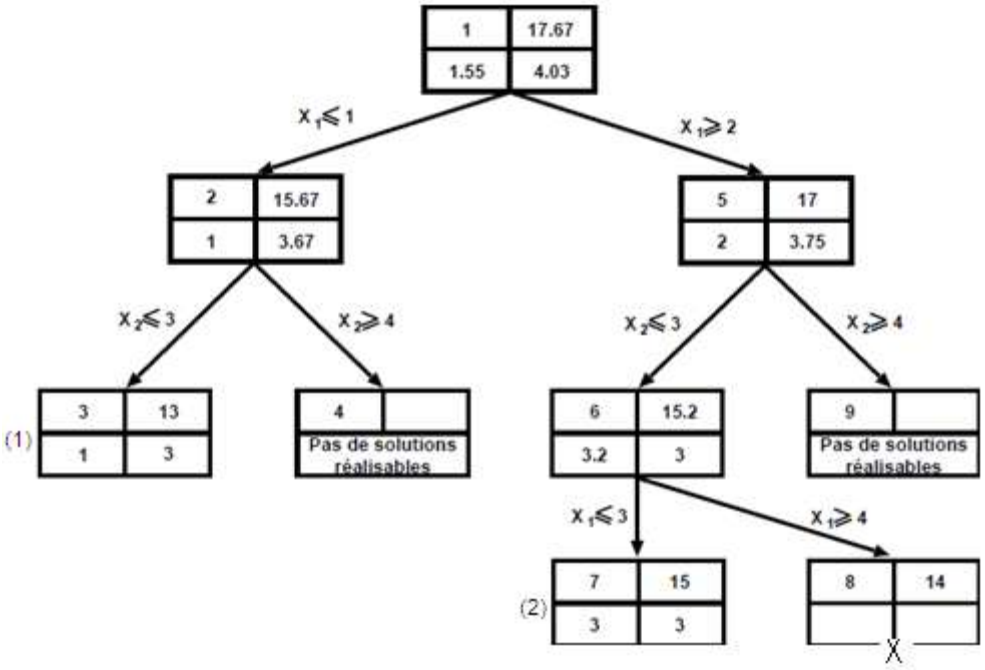
4) Ajouter à (*P*) les contraintes $x_1 \leq 1$ et $x_2 \geq 4$. Ce programme n'a pas de solutions réalisables.
5) Ajouter à (*P*) la contrainte $x_1 \geq 2$. La solution optimale $x = (2, 3.75)$, $z(x) = 17$.
Séparer par rapport à x_2 : $x_2 \leq 3$ ou $x_2 \geq 4$.
6) Ajouter à (*P*) les contraintes $x_1 \geq 2$ et $x_2 \leq 3$. Solution optimale $x = (3.2, 3)$, $z(x)$ $= 15.2$.
Séparer par rapport à x_1 : $x_1 \leq 3$ ou $x_1 \geq 4$.

7) Ajouter à (*P*) les contraintes $2 \leq x_1 \leq 3$ et $x_2 \leq 3$. Solution optimale $x =$ $(3, 3)$, $z(x) = 15$.
Evaluer à 15.

8) Ajouter à (*P*) les contraintes $x_1 \geq 4$ et $x_2 \leq 3$. Cas sans intérêt parce que $z(x) = 14 < 15$ pour la solution optimale x.

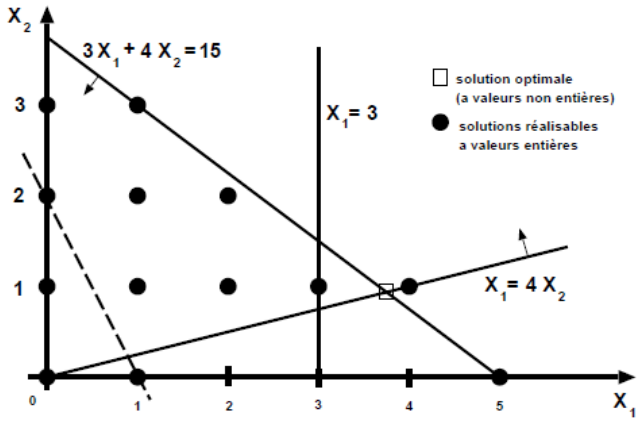
9) Ajouter à (*P*) les contraintes $x_1 \geq 2$ et $x_2 \geq 4$. Pas de solutions réalisables.

La solution optimale à valeurs entières est donc celle trouvée en 7) :
$(x_1^*, x_2^*) = (3, 3)$, avec $z^* = 15$.
L'arborescence qui montre le résultat est la suivante :



La méthode des coupes.

$$(P)\begin{cases} Max\ Z = 2x_1 + x_2 \\ x_1 - 4x_2 \le 0 \\ 3x_1 + 4x_2 \le 15 \\ x_1, x_2 \in N \end{cases}$$



La méthode des coupes consiste à ajouter des contraintes supplémentaires qui permettent d'approcher les solutions réalisables à valeurs entières sans les écarter du domaine des solutions réalisables. Dans l'exemple ci-dessus une telle contrainte est donnée par $x_1 \le 3$ (on coupe à $x_1 = 3$).

Comment trouver de telles contraintes supplémentaires si le programme linéaire contient plus de deux variables ?

Soit $(P)\begin{cases} Max\ Z = cx \\ Ax = b \\ x \ge 0 \end{cases}$, où A est une matrice $m \times n$.

Posons, pour une base J (avec $|J| = m$) et une solution réalisable x,

$Ax = A^J x^J + A^{J^C} x^{J^C} = b$ où $x^J = \left(x_j^J, j \in J\right)$, $x^{J^C} = \left(x_j^{J^C}, j \in J^C\right)$

Puisque A^J est régulier, $x^J + (A^J)^{-1} A^{J^C} x^{J^C} = (A^J)^{-1} b$.

Posons : $\bar{A} = (A^J)^{-1} A^{J^C}$ et $\bar{b} = (A^J)^{-1} b$

On a donc $x^J + \bar{A} x^{J^C} = b$ où $x_i^J + \sum_{j \notin J} \bar{a}_{ij} x_j^{J^C} = \bar{b}_i$ $(j \in J)$ et $\bar{\bar{A}} = \left(\bar{a}_{ij}, i, j = 1, \dots, n\right)$

Soit $\left[\bar{a}_{ij}\right]$ la partie entière de $\bar{a}_{ij}$, $\langle \bar{a}_{ij} \rangle$ la partie fractionnaire de $\bar{a}_{ij} = \left[\bar{a}_{ij}\right] + \langle \bar{a}_{ij} \rangle$

Exemples : $[3.5] = 3 \quad \langle 3.5 \rangle = 0.5$

$[-3.5] = -4 \quad \langle -3.5 \rangle = 0.5$

Il s'ensuit :

$x_i^J + \sum_{j \notin J} \left(\left[\bar{a}_{ij}\right] + \langle \bar{a}_{ij} \rangle\right) x_j^{J^C} = \left[\bar{b}_i\right] + \langle \bar{b}_i \rangle$

$x_i^J + \sum_{j \notin J} \left[\bar{a}_{ij}\right] x_j^{J^C} - \left[\bar{b}_i\right] = \langle \bar{b}_i \rangle - \sum_{j \notin J} \langle \bar{a}_{ij} \rangle x_j^{J^C}$ *(1)*

Soit $\underline{x}$ une solution réalisable de (*P*) à valeurs entières. L'égalité (1) vaut pour *x*, le côté gauche étant à valeurs entières. En plus,

$\underline{x}_i^J + \sum_{j \notin J} \left[\bar{a}_{ij}\right] \underline{x}_j^{J^C} \leq \underline{x}_i^J + \sum_{j \notin J} \bar{a}_{ij} \underline{x}_j^{J^C} = \bar{b}_i$ *(2)*

Comme la partie gauche de (2) est à valeurs entières,

$\underline{x}_i^J + \sum_{j \notin J} \left[\bar{a}_{ij}\right] \underline{x}_j^{J^C} \leq \left[\bar{b}_i\right]$

Comme (1) est valable pour toute solution réalisable, il s'ensuit que :

$$\langle \bar{\boldsymbol{b}} \rangle - \sum_{j \notin J} \langle \bar{\boldsymbol{a}}_{\boldsymbol{ij}} \rangle \underline{\boldsymbol{x}}_{\boldsymbol{j}}^{\boldsymbol{J^C}} \leq \mathbf{0}$$

Proposition *En ajoutant à* (*P*)*, la contrainte :*

$$\sum_{j \notin J} \langle \bar{\boldsymbol{a}}_{\boldsymbol{ij}} \rangle \underline{\boldsymbol{x}}_{\boldsymbol{j}}^{\boldsymbol{J^C}} \geq \langle \bar{\boldsymbol{b}} \rangle$$

On n'écarte pas de solutions réalisables à valeurs entières de l'ensemble des solutions réalisables.

Remarque. Comme l'exemple suivant montre, on arrive à la solution optimale à valeurs entières en ajoutant, à plusieurs reprises si nécessaire, des contraintes supplémentaires suivant la proposition ci-dessus.

Exemple.

$$(P) \begin{cases} Max\ Z = 2x_1 + x_2 \\ -x_1 + x_2 \leq 0 \\ 5x_1 + 2x_2 \leq 18 \\ x_1, x_2 \in N \end{cases}$$

Pour résoudre (*P*) on procède par étapes :

1) Résolution avec le simplexe sans tenir compte de la condition « x_1, x_2 à valeurs entières». Solution optimale *x* = (18/7, 18/7, *0, 0).*

Tableau initial

		x_1	x_2	x_3	x_4	
C_B	X_B	2	1	0	0	B
0	x_3	-1	1	1	0	0
0	x_4	5	2	0	1	18
Z_j		0	0	0	0	0
Δ_j		2	1	0	0	

Tableau final

		x_1	x_2	x_3	x_4	
C_B	X_B	2	1	0	0	b
1	x_2	0	1	5/7	1/7	18/7
2	x_1	1	0	-2/7	1/7	18/7
Z_j		2	1	1/7	3/7	54/7
Δ_j		0	0	-1/7	-3/7	

Nouvelle contrainte suivant la proposition ci-dessus :

$\frac{5}{7}x_3 + \frac{1}{7}x_4 \geq \frac{4}{7}$ *(S1)*

La contrainte exprimée en termes de *x*1 et de *x*2 : $x_2 \leq 2$

2) Résolution de (*P*), (*S*1) inclus, par le simplexe. Exige le passage par un programme auxiliaire.

Programme auxiliaire, tableau initial

		x_1	x_2	x_3	x_4	x_5	y	
C_B	X_B	0	0	-5/7	-1/7	1	0	b
0	x_2	0	1	5/7	1/7	0	0	18/7
0	x_1	1	0	-2/7	1/7	0	0	18/7
0	Y	0	0	5/7	1/7	-1	1	4/7
Z_j		0	0	0	0	0	0	-4/7
Δ_j		0	0	-5/7	-1/7	1	0	

La fonction-objectif du programme auxiliaire est donné par :

$Min\ w = y$

$= \frac{4}{7} - \frac{5}{7}x_3 - \frac{1}{7}x_4 + x_5$

Programme auxiliaire, tableau final

		x_1	x_2	x_3	x_4	x_5	
C_B	X_B	0	0	-5/7	-1/7	1	b
0	x_2	0	1	0	0	1	2
0	x_1	1	0	0	1/5	-2/5	14/5
-5/7	x_3	0	0	1	1/5	-7/5	4/5
Z_j		0	0	-5/7	-1/7	-1	-4/7
Δ_j		0	0	0	0	1	

Comme w = 0, le programme (*P*)+(*S*1) possède une solution réalisable. On passe à la deuxième phase.

Tableau final seconde phase

		x_1	x_2	x_3	x_4	x_5	
C_B	X_B	2	1	0	0	0	b
1	x_2	0	1	0	0	1	2
2	x_1	1	0	0	1/5	-2/5	14/5
0	x_3	0	0	1	1/5	-7/5	4/5
Z_j		2	1	0	2/5	1/5	38/5
Δ_j		0	0	0	-2/5	-1/5	

Nouvelle contrainte selon la proposition ci-dessus :

$\frac{1}{5}x_4 + \frac{3}{5}x_5 \geq \frac{4}{5}$ *(S2)*

La contrainte exprimée en termes de *x*1 et de *x*2 :

$x_1 + x_2 \leq 4$

3) Résolution de (*P*), (*S*1) et (*S*2) inclus, exige à nouveau le passage par un programme auxiliaire.

Seconde phase du simplexe, tableau final

		x_1	x_2	x_3	x_4	x_5	x_6	
C_B	X_B	2	1	0	0	0	0	b
1	x_2	0	1	0	-1/3	0	5/3	2/3
2	x_1	1	0	0	1/3	0	-2/3	10/3
0	x_3	0	0	1	2/3	0	-7/3	8/3
0	x_5	0	0	0	1/3	1	-5/3	4/3
Z_j		2	1	0	1/3	0	1/3	22/3
Δ_j		0	0	0	-1/3	0	-1/3	

Nouvelle contrainte selon la proposition ci-dessus :

$x_4 + x_6 \geq 1$ *(S3)*

La contrainte exprimée en termes de *x*1, *x*2 :

$2x_1 + x_2 \leq 7$

4) Résolution de (*P*), (*S*1), (*S*2) et (*S*3) inclus, exige à nouveau le passage par un programme auxiliaire et mène, cette fois, à une solution optimale à valeurs entières, qui est *x* = *t*(3, 1, 2, 0, 1, 1, 0).

Tableau final de la seconde phase du simplexe

		x_1	x_2	x_3	x_4	x_5	x_6	x_7	
C_B	X_B	2	1	0	0	0	0		b
1	x_2	0	1	0	0	0	2	-1/2	1
2	x_1	1	0	0	0	0	-1	1/2	3
0	x_3	0	0	1	0	0	-3	1	2
0	x_5	0	0	0	0	1	-2	1/2	1
0	x_6	0	0	0	1	0	1	-3/2	1
Z_j		2	1	0	0	0	0	1/2	7
Δ_j		0	0	0	0	0	0	-1/2	

Donc la solution optimale à valeurs entières de (P) est : $(x_1^*, x_2^*) = (3, 1)$ avec $z^* * 7$.

Sur la figure ci-après on voit que, en ajoutant successivement ($S1$), ($S2$) et ($S3$), on n'écarte pas de solutions à valeurs entières, et on coupe l'ensemble des solutions réalisables de façon que la solution optimale à valeurs entières apparaît comme point extrémal.