Chapitre 6 : La méthode de branch and bound

1.Introduction

Pour plusieurs problèmes, en particulier les problèmes d'optimisation, l'ensemble de leurs solutions est fini (en tous les cas, il est dénombrable). Il est donc possible, en principe, d'énumérer toutes ces solutions, et ensuite de prendre celle qui nous arrange. 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 de *branch and bound* (*procédure par évaluation et séparation progressive*) consiste à énumérer ces solutions d'un 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 recherche. De ce fait, on arrive souvent 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 ce faire, cette méthode se dote d'une fonction qui permet de mettre une borne sur certaines solutions pour soit les exclure soit les maintenir comme des solutions potentielles. Bien entendu, la performance d'une méthode de branch and bound depend, entre autres, de la qualité de cette fonction (de sa capacité d'exclure des solutions partielles tôt).

2. L'algorithme général

Par convenance, on représente l'exécution de la méthode de branch-and-bound à travers une arborescence. La racine de cette arborescence représente l'ensemble de toutes les solutions du problème considéré. Dans ce qui suit, nous résumons la méthode de branch-and-bound sur des problèmes de minimisation.

Pour appliquer la méthode de branch-and-bound, nous devons être en possession :

1. d'un moyen de calcul d'une borne inférieure d'une une solution partielle
2. d'une stratégie de subdiviser l'espace de recherche pour créer des espace de recherche de plus en plus petits.
3. d'un moyen de calcul d'une borne supérieure pour au moins une solution.

La méthode commence par considérer le problème de départ avec son ensemble de solutions, appelé la racine. Des procédure de bornes inférieures et supérieures sont appliquées à la racine. Si ces deux bornes sont égales, alors une solution optimale est trouvée, et on arrête là. Sinon, l'ensemble des solutions est divisée en deux ou plusieurs sous-problèmes, devenat ainsi des enfants de la racine. La méthode est ensuite appliquée récursivement à ces sous-problèmes, engendrant ainsi une arborescence. Si une solution optimale est trouvée pour une sous-problème, elle est réalisable, mais pas nécessairement optimale, pour le problème départ. Comme elle est réalisable, elle peut être utilisée pour éliminer toute sa descendance : si la borne inférieure d'un

nœud dépasse la valeur d'une solution déjà connue, alors on peut affirmer que la solution optimale globale ne peut être contenue dans le sous-ensemble de solution représenté par ce nœud. La recherche continue jusqu'à ce que tous les nœuds sont soit explorés ou éliminés.

3. Illustration de la méthode sur quelques problèmes

La moyen le plus efficace, de comprendre un nouveau concept, est l'illustration par l'exemple. Cette section est donc dévolue à la résolution de quelques problèmes par la méthode de Branch-and-Bound

Application 3.1 : le voyageur de commerce

Nous avions déjà vu ce problème dans les chapitre précédents. Rappleons tout de même sa définition.

Définition 3.1 : 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.

Définition 3.2: 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 donc le graphe de la Figure 6.1:

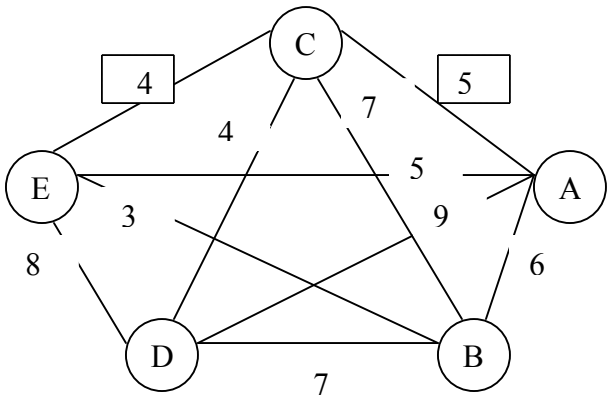

Figure 6.1

Cet graphe peut être bien entendu représenté par sa matrice D d'adjacence tel que D[i,j] représente le poids de l'arc (i,j).

Résolvons ce problème en commençant par exemple à partir du sommet **E**.

Soit la borne v pour ce sommet (on verra plus loin comment la trouver !) c'est-à-dire la valeur de toutes les solutions incluant le sommet E vont avoir un coût $\geq v$.

Le prochain sommet du cycle que nous recherchons est soit A, B, C ou D. Pour chacune de ces solutions partielles, nous calculons une nouvelle borne (signifiant que toutes les solutions comprenant cette solution partielle va avoir un coût ≥ à cette nouvelle borne).

Initialement, nous pouvons mettre le coût de la meilleure solution trouvée à un très grand nombre ou bien de le trouver par une quelconque autre méthode (aléatoire ou autre).

L'idée de cette méthode la suivante : toute solution partielle dont le coût est plus grand que celui de la meilleure solution trouvée jusqu'à présent va être exclue de la recherche. Ce processus est continué jusqu'à avoir une solution complète. Si le coût de cette solution complète est inférieure à celui de la meilleure solution que nous avons déjà, alors nous remplaçons ce coût comme étant celui de la meilleure solution courante.

Il existe plusieurs manières de décider quelle solution partielle à explorer en premier. Celle que nous allons décrire ci-dessous est celle qui consiste à choisir toujours la solution ayant la plus petite borne. Mais il existe d'autres manières de procéder. Les unes se valent que les autres.

La partie cruciale de branch and bound est la qualité de la fonction F. Si elle est trop simple, probablement que l'exclusion des solutions partielles se fera à des moment avancés. Cela a pour effet de rallonger encore les temps d'exécution des ces algorithmes.

Une fonction F peut être comme suit :

Lemme Soit le cycle hamiltonien suivant : $v_1, v_{2,}, ..., v_n, v_{n+1} = v_1$. Il est facile de voir que, quelque soit la solution, son coût est $\geq \frac{1}{2}\sum_{i=1}^{n}(arc-v_{i_1} + arc-v_{i_2})$ où $arc-v_{i_1}$ et $arc-v_{i_2}$ désignent deux arcs adjacents au sommet *i* ayant **le plus petit poids**.

Preuve : Quelque soit le cycle hamiltonien, il doit y avoir deux arcs incidents à chaque sommet du graphe : un arc sortant ce sommet et un arc entrant ce sommet. Le poids de l'arc sortant le sommet i est donné par D[i,j] , pour un certain $j \neq i$. Le poids de l'arc entrant est aussi donné par D[k,i], pour un certain $k \neq j$, car les deux arcs devant être distincts. Par conséquent, quelque soit un cycle hamiltonien, la somme des poids de deux arc incidents à un sommet i doit être supérieure à celle des plus petits arcs deux arcs incidents à i. En sommant sur les i, on compte par deux fois chacun des arcs. D'où le résultat du lemme.

Appliquons cette borne sur le graphe ci-dessus. En commençant à partir de E, on aura donc:

Le coût est ≥ ½{(3+4)+(4+4)+(5+5)+(3+6)+(4+7)}= 22.5

Les prochain sommets dans le cycle peuvent être : A, B, C ou D. Pour chacune des ces solutions partielles, une borne est calculée. Par exemple pour D, la nouvelle borne est :

½{(8+3)+(4+4)+(5+5)+(3+6)+(4+8)}= 25.

Pour calculer cette borne, on applique la fonction ci-dessus en tenant compte que pour le sommet E, l'arc (E,D) doit être pris, et pour le sommet D, l'arc (D,E) doit être pris. La valeur 25 représente donc la plus petite valeur d'un cycle hamiltonien incluant les arcs (E,D) et (D,E) ; les arcs n'étant pas orientés dans ce graphe.

Ceci est fait pour chacun des sommets. On obtient successivement : 22.5 pour le sommet **C,** 23 pour le sommet **A** et 22.5 pour le sommet **B**. Parmi ces quatre valeurs, nous allons explorer le sommet ayant la plus petite valeur, soit le sommet **C,** avant d'explore le second sommet ayant la plus petite borne, etc. Il est utile de signaler qu'il existe d'autre manières de procéder dans le choix de la prochaine solution partielle à explorer. Toutefois, chacune des ces méthodes a ses propres avantages et inconvénients. À titre d'exemple, nous pouvons citer l'exploration en profondeur d'abord (DFS) et la méthode en largeur d'abord (BFS).

Ce processus est ensuite répété. Quand une solution complète est trouvée, la meilleure solution courante est modifiée si cela est nécessaire. Par exemple, dans notre exemple, la première solution trouvée a pour coût 26 remplaçant ainsi l'ancienne valeur que nous avons initialisée à un grand nombre.

L'idée de Branch and Bound peut être résumée comme suit :

Quand un sommet a une borne plus grande que cette valeur, il n'y a plus lieu de le considérer étant donnée que cette solution partielle ne mènera pas à la solution minimale.

Ce processus de génération de solutions partielles, avec le calcul de leur borne, génère une arborescence. Pour l'exemple ci-dessus, l'arborescence obtenue est la suivante :

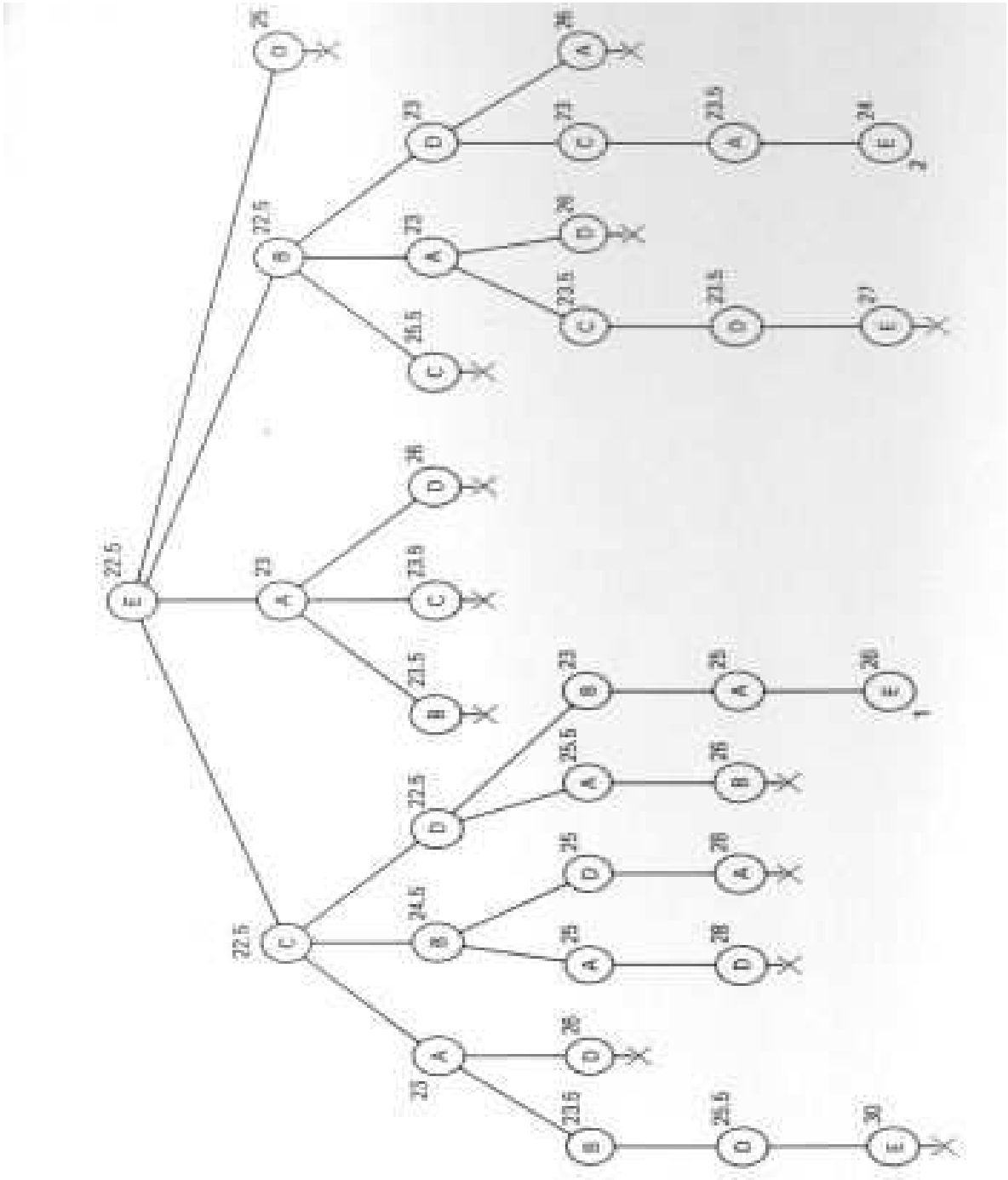

Figure 6.2.

Remarquez qu'à égalité des bornes, on choisit au hasard. On prend en premier le sommet C, on génère les sommets A, B et D. On ensuite, on explore le sommet D. Ensuite B, A et E. à ce stade, nous avons une solution complète (ECDBAE) de valeur 26. Ensuite, on explore le sommet A qui va donner le sommet B, de valeur 26. Il n'y pas lieu d'explorer encore cette solution partielle car sa borne inférieure vaut déjà 26.

Application 2 : problème d'affectation

Soient *n* personnes à affecter à *n* tâches. Le coût de l'affectation de *i* à la tâche *j* est noté par c_{ij} .
Le problème consiste à affecter chaque personne à une seule tâche de telle manière à minimiser le coût total . Bien entendu, une tâche ne peut être faite que par une seule personne.

Illustration. Soit la matrice suivante des coûts

	tâche 1	**tâche 2**	**tâche 3**	**tâche 4**	
	9	2	7	8	personne **a**
C =	6	4	3	7	personne **b**
	5	8	1	8	personne **c**
	7	6	9	4	personne **d**

Si par exemple, on décide de faire l'affectation suivante

(1, a) ; (2, c) ; (3, b) et (4,d); on aura le coût total suivant : 1 + 8 + 3 + 4 = 16.

Par contre, si on faisait l'affectation suivante :

(1, b) ; (2, c) ; (3, d) et (4,a); on aura le coût total suivant 6 + 8 + 9 + 8 = 30.

L'application de la méthode de branch-and-bound nécessite une fonction de borne inférieure.

Une des ces fonctions de borne inférieure pourrait être celle-ci :

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. Dans notre cas,

$$2 + 3 + 1 + 4 = 10.$$

On commence par la racine qui correspond à l'état de « pas d'élément encore choisi de la matrice des coûts ». La valeur de la borne inférieure est alors égale à 10. Les nœuds au premier niveau de l'arbre correspondent au 4 jobs de la première ligne de la matrice à partir du moment que chacun d'eux est un chois potentiel de la première composante de la solution. Le plus prometteur parmi eux est le nœud 2 car ayant la plus petite borne inférieure, comme illustré par la Figure 6.3.

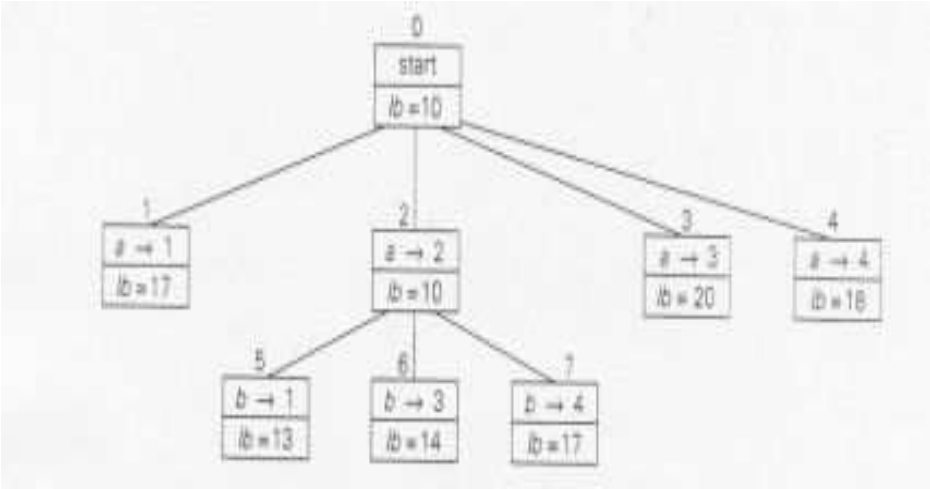

Figure 6.3.

En continuant, on génère les nœuds comme illustrés par la Figure 6.4. jusqu'à arriver à la solution (2,a) ; (1,b) ; (3,c) et (4,d) engendrant un coût de valeur 13. Remarquons que, quand on arrive à noeud 8, on choisit le couple (c,3). Cela nous laisse obligatoirement avec le couple (4,d). C'est la première solution complète qu'on génère à ce stade de calculs. À partir de là, soit on génère une solution dont le coût est supérieur à la solution courante (comme c'est le cas pour le nœud 9) soit on ignore les nœuds car ne pouvant pas contenir une solution meilleure que celle dont on dispose (comme c'est le cas des nœuds 1, 3,4, 6 et 7).
Notons que c'est par pur hasard que nous avons généré du premier coup la solution optimale.

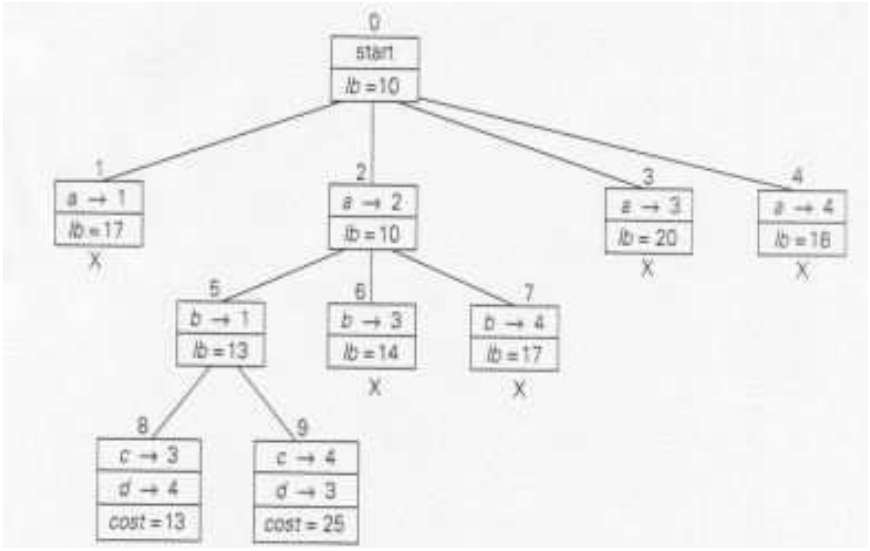

Figure 6.4.

Application 3 : 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 finalement 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.

Illustration Soit 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 1, 2, 3 et 4, alors le temps d'accomplissement de toutes les tâches est égal à ??, comme illustré par le diagramme de Gantt suivant : (le donner en cours)

Si, par contre, on les exécute dans l'ordre 4, 3, 1 et 2, alors le temps d'accomplissement de toutes les tâches est égal à ??, comme illustré par le diagramme de Gantt suivant (le donner en cours).

Avant de procéder à la résolution proprement dite, on aura besoin de calculer le makespan pour une solution donnée.

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 $\alpha_{i(k)}, \beta_{i(k)}, \gamma_{i(k)}$ la date 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 :

$$\alpha_{i(k)} = a_{i(1)}; \alpha_{i(k)} = \alpha_{i(k-1)} + a_{i(k)};$$

$$\beta_{i(1)} = a_{i(1)} + b_{i(1)}; \beta_{i(k)} = \max\{\alpha_{i(k)}, \beta_{i(k-1)}\} + b_{i(k)};$$

$$\gamma_{i(1)} = a_{i(1)} + b_{i(1)} + c_{i(1)}; \alpha_{i(k)} = \max\{\beta_{i(k)}, \gamma_{i(k-1)}\} + c_{i(k)}.$$

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

$$C_{\max} = \alpha_{i(k)} + \sum_{i \in U} a_i + (b_{i(n)} + c_{i(n)})$$

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

$$C_{\max} \geq \alpha_{i(k)} + \sum_{i \in U} a_i + \min_{i \in U}(b_i + c_i)$$

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

$$C_{\max} = \beta_{i(k)} + \sum_{i \in U} b_i + c_{i(n)})$$

En choisissant le plus court temps d'exécution sur la machine 3, quelque soit la solution, on a alors :

$$C_{\max} = \beta_{i(k)} + \sum_{i \in U} b_i + \min_{i \in U} c_{i(n)}$$

Similairement, en considérant que la machine 3 est continue dans son exécution. On obtient la borne suivante :

$$C_{\max} \geq \gamma_{i(k)} + \sum_{i \in U} c_i$$

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

$$C_{\max} \geq \max\left\{\alpha_{i(k)} + \sum_{i \in U} a_i + \min_{i \in U}\{b_i + c_i\}; \beta_{i(k)} + \sum_{i \in U} b_i + \min_{i \in U}\{c_i\}; \gamma_{i(k)} + \sum_{i \in U} c_i\right\}$$

Munis de cette borne inférieure, on peut passer à la résolution de notre problème. À la racine de l'arbre, aucune tâche n'est exécutée, la borne inférieure est donc :

$$C_{\max} \geq \max\left\{\alpha_{i(k)} + \sum_{i \in U} a_i + \min_{i \in U}\{b_i + c_i\}; \sum_{i \in U} b_i + \min_{i \in U}\{c_i\}; \sum_{i \in U} c_i\right\} = \{12 + 2 + 2; 23 + 2; 19\} = 14.$$

Autrement dit, aucune solution ne peut avoir un makespan inférieur à 14.

- Si on met la tâche 1 en position 1, on obtient :

A = {1} ; U={2,3,4}

$\alpha_1 = 1; \beta_1 = 9; \gamma_1 = 13$

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}

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

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 = $\max\{???9+10;13+11+2;18+10\} = 28$.

- Si on met la tâche 4 en position 4, on obtient :

A = {1,2,3,4} ; U={}

$\alpha_4 = 9+3 = 12; \beta_4 = \max\{12,15\}+9 = 24; \gamma_4 = \max\{24,26\}+2 = 28$

Par conséquent, le makespan est donc égal à 28.

On vient de trouver que la valeur du makespan, de notre première solution, est 28, comme le montre la Figure 6.5.

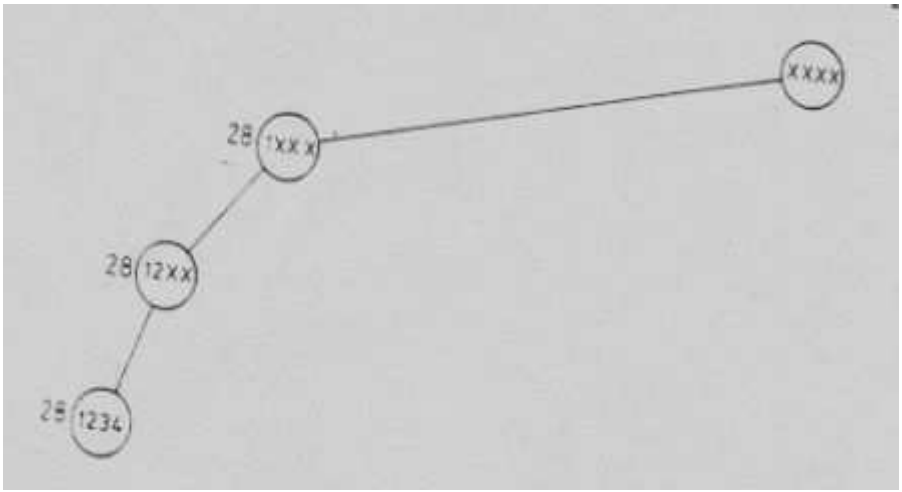

Figure 6.5.

Ce qu'on vient de faire jusqu'ici, c'est d'avoir traversé l'arborescence de calculs en profondeur d'abord. Cela nous permet, à partir de maintenant, d'avoir une borne supérieure du makespan. Si, dans l'exploration de cette arborescence, on obtient des bornes inférieures plus grande que 28, on peut les ignorer de la recherche, puisque ces solutions partielles ne peuvent contenir la solution optimale.

En continuant de la sorte, on obtient la Figure 6.6. suivante : une arborescence complètement explorée.

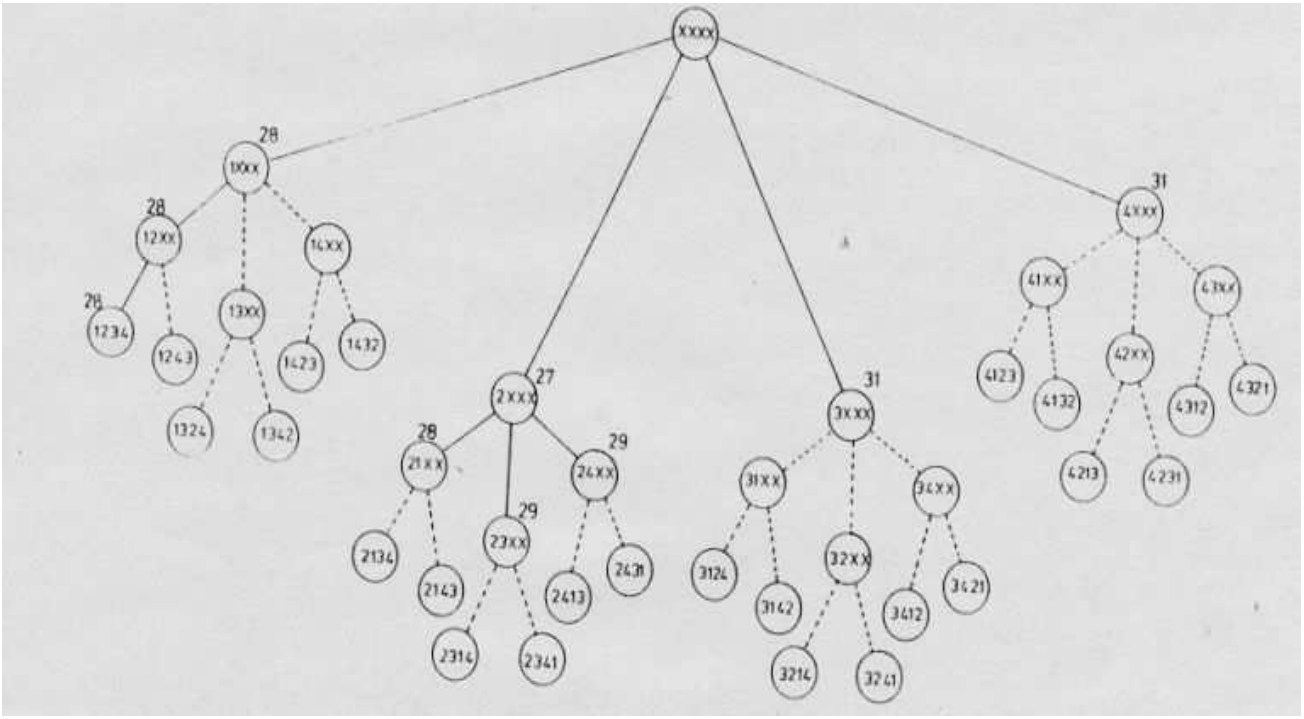

Figure 6.6.

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.

Rérérences:

1. Azmoodeh, M. (1988): Abstract data types and algorithms, McMillan.
2. French, S. (1982): *Sequencing and scheduling*, Ellis Horwood Ltd., Chichester,
3. Levitin, A. (2003): Introduction to the design and analysis of algorithms, Addison Wesley.