RAIRO. RECHERCHE OPÉRATIONNELLE

JACQUES CARLIER
BRUNO LATAPIE

Une méthode arborescente pour résoudre les problèmes cumulatifs

RAIRO. Recherche opérationnelle, tome 25, n° 3 (1991), p. 311-340

<http://www.numdam.org/item?id=RO_1991__25_3_311_0>

Recherche opérationnelle/Operations Research
(vol. 25, n° 3, 1991, p. 311 à 340)

UNE MÉTHODE ARBORESCENTE POUR RÉSOUDRE LES PROBLÈMES CUMULATIFS (*)

par Jacques CARLIER (1) et Bruno LATAPIE (1)

Résumé. – *Le but de cet article est de décrire une procédure minimisant la durée totale d'ordonnancements devant respecter des contraintes potentielles et des contraintes cumulatives. Le problème correspondant étant NP-difficile, cette procédure est arborescente. Pour la construire, nous associons à chaque tâche l'intervalle durant lequel elle pourra s'exécuter. La séparation consiste à choisir une tâche et à remplacer son intervalle d'exécution par deux intervalles recouvrant l'intervalle initial. Pour avoir des évaluations par défaut efficaces, nous générons, à partir du problème cumulatif initial, des problèmes à m machines. Enfin, l'évaluation par excès utilise l'algorithme de Jackson.*

Des tests sur des jeux d'essai classiques et sur des exemples générés aléatoirement montrent l'efficacité de cette méthode.

Mots clés : Ordonnancement; méthode arborescente; contraintes de ressources; gestion de projet.

Abstract. – *This paper presents a procedure for scheduling in a minimal makespan a project subject to precedence and cumulative constraints. Because of the NP-hardness of this problem, a branch and bound method is proposed. An execution interval is associated with each task. Branching consists of choosing a task and splitting its interval of execution into two intervals. Lower bounds are computed by m-machines problems, upper bounds are based on Jackson's algorithm.*

Tests on classical benchmarks and randomly generated problems show the effectiveness of this method.

Keywords : Scheduling; branch and bound method; resource constraints; project scheduling.

INTRODUCTION

Les problèmes d'ordonnancements cumulatifs incluent à la fois des contraintes potentielles et des contraintes de ressources. Ils sont NP-difficiles

(*) Reçu mai 1990.
(1) U.R.A.-C.N.R.S. n° 817, HEUDIASYC-HEUristique et DIAgnostic des SYstèmes Complexes, Université de Technologie de Compiègne, B.P. n° 649, 60206 Compiègne Cedex, France.

Recherche opérationnelle/Operations Research, 0399-0559/91/03 311 30/$ 5.00

[GAREY 79]. On rencontre dans la littérature des méthodes approchées, comme les méthodes sérielles [ALVAREZ 88] et des méthodes exactes [DAVIS 75], [PATTERSON 76], [CHRISTOFIDES 87], [HERROELEN 90]. Les méthodes sérielles présentent l'avantage d'une grande simplicité de mise en œuvre. Mais du fait même de leur principe, elles peuvent systématiquement écarter la solution optimale [GRAHAM 69]. Par ailleurs, elles ne contiennent pas d'évaluation préalable de distance à l'optimum.

Les méthodes exactes ont l'inconvénient de ne pouvoir traiter que des exemples de taille réduite si on vise l'optimalité. Elles sont toutefois plus satisfaisantes dans la mesure où elles permettent, même pour des exemples de grande taille, de construire des solutions approchées dont on connaît une bonne évaluation de la distance à l'optimum. Le but de cet article est de présenter une méthode exacte de type arborescent généralisant la méthode des intervalles [CARLIER 87 B] aux problèmes cumulatifs. Son principe consiste à associer à chaque tâche un intervalle d'exécution qui sera éventuellement recouvert par deux intervalles au cours du parcours arborescent. L'évaluation par défaut est obtenue en s'appuyant sur les problèmes à une et à m machines, l'évaluation par excès emploie l'algorithme de Jackson. La séparation est choisie de manière à faire augmenter l'évaluation par défaut. Nous avons implémenté la méthode en utilisant les stratégies en profondeur d'abord et en largeur d'abord. La méthode des intervalles est donc fort différente des méthodes classiques basées sur la programmation linéaire en nombres entiers ou sur la génération d'ordonnancements actifs. Elle donne d'excellents résultats que nous rapportons.

Dans cette présentation, nous rappelons au paragraphe 1 les contraintes des problèmes cumulatifs et les différentes méthodes de résolution publiées. Nous présentons au paragraphe 2 les problèmes à une et à m machines qui permettent d'établir une évaluation par défaut. Ensuite, nous analysons au paragraphe 3 les différents aspects de la méthode des intervalles. Enfin, nous rapportons au paragraphe 4 les résultats obtenus par cette méthode en remarquant qu'ils se comparent favorablement à ceux établis auparavant dans la littérature par des algorithmes exacts.

1. PRÉSENTATION DES PROBLÈMES CUMULATIFS

Ordonnancer, c'est programmer l'exécution d'une réalisation en allouant des ressources aux tâches et en fixant leur date d'exécution tout en cherchant à atteindre un objectif, par exemple, la minimisation de la durée de l'ordonnancement. On rencontre couramment les problèmes d'ordonnancement dans les entreprises et en particulier en informatique.

Ces problèmes de formulation assez simple, sont le plus souvent NP-difficiles. Ils se scindent, par ordre de difficulté croissante, en plusieurs catégories interdépendantes :

- ordonnancements à contraintes potentielles : on ne tient compte que de la précédence entre les tâches;
- problème à une machine :
  des tâches doivent être effectuées sur une même machine,
- problème à m machines :
  on dispose de m machines pour exécuter un ensemble de tâches,
- ordonnancements à contraintes cumulatives :
  on doit satisfaire aux contraintes potentielles et aux contraintes de ressources telles que les machines, le personnel, la mémoire.

Les problèmes à une machine ou à m machines, qui sont des problèmes cumulatifs particuliers, sont NP-difficiles; les problèmes cumulatifs sont donc eux-mêmes NP-difficiles.

1.1. Définition du problème

Dans un problème cumulatif, un ensemble de tâches, utilisant des ressources dont la disponibilité est limitée, sont soumises à des contraintes potentielles. Les quantités de ressources nécessaires pour l'exécution de chaque tâche sont connues. Les contraintes cumulatives résultent du fait que les quantités de ressources disponibles sont limitées et souvent inférieures aux quantités de ressources nécessaires pour exécuter les tâches prêtes si on ne tient compte que des contraintes potentielles. La fonction économique est la durée. Dès qu'une tâche est commencée, on exige qu'elle ne soit pas interrompue avant sa fin : l'ordonnancement est dit sans préemption. On suppose que, pour chaque tâche, la durée d'exécution est fixée et on recherche un ordonnancement de durée minimale.

Ces problèmes sont l'extension de ceux résolus par les méthodes P.E.R.T. et potentiels-tâches au cas où les tâches requièrent des ressources. Ils sont donc d'un grand intérêt pratique mais sont très difficiles.

Pour établir un ordonnancement, on doit satisfaire les contraintes suivantes :

$$\operatorname{Min} t_{n+1} \tag{1}$$

$$t_j - t_i \geqq v_{ij}, \quad v_{ij} \in \mathbb{R}, \qquad \forall (i,j) \in U \tag{2}$$

$$\sum_{i \in S(t)} a_{ik} \leqq b_k, \quad \forall k \in K, \quad \forall t \in \mathbb{N}, \qquad S(t) = \{ i \in X / t_i \leqq t < t_i + p_i \} \tag{3}$$

où :

X : l'ensemble des tâches à exécuter;

U : l'ensemble des arcs du graphe conjonctif;

K : l'ensemble des ressources;

t_i : la date de début de placement de la tâche i dans l'ordonnancement ($i=0, \ldots, n+1$);

p_i : la durée de la tâche i;

v_{ij} : la valuation de l'arc (i, j). Sur les exemples que nous considérons, $v_{ij}=p_i$ sauf pour $i=0$. v_{0j} sera la date de disponibilité de la tâche j;

$S(t)$: l'ensemble des tâches en cours d'exécution à l'instant t;

a_{ik} : la quantité de ressource k *nécessaire* à la tâche i pour son exécution;

b_k : la quantité de ressource k *disponible* durant tout l'ordonnancement.

Les tâches sont numérotées de 0 à $n+1$, la tâche de début notée 0 et la tâche de fin appelée $n+1$ sont deux tâches fictives. p_i, b_k et a_{ik} sont des constantes fixées au début du problème. Les contraintes de précédence données par la relation (2) imposent qu'une tâche j ne peut s'exécuter que lorsque toutes les tâches i la précédant dans le graphe sont terminées. Les contraintes de ressources définies par (3) sont satisfaites si la demande totale des tâches s'exécutant à l'instant t n'excède pas la disponibilité de la ressource. L'objectif donné par (1) est de minimiser la durée totale de l'ordonnancement, c'est-à-dire la date de début de placement de la tâche fictive $n+1$.

La figure 1 ci-dessous rapporte un exemple à 16 tâches et 2 ressources. Les sommets du graphe représentent les tâches à exécuter. Les tâches 0 et 17 sont les deux tâches fictives. Les arcs du graphe conjonctif sont valués par la durée des tâches. Cependant, les valuations des arcs (0, 1), (0, 5), (0, 9), (0, 13) sont respectivement les dates de disponibilité des tâches 1, 5, 9 et 13. Dans cet exemple, une tâche utilise une seule ressource mais cette condition ne fait pas partie des hypothèses. En effet, nous avons traité de nombreux exemples où les tâches utilisent plusieurs ressources.

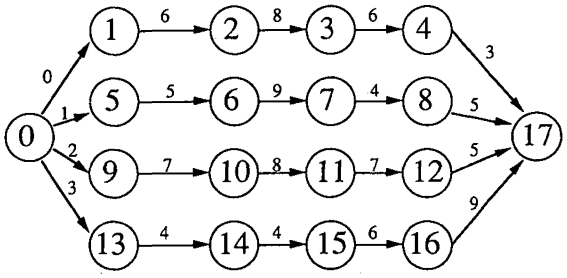
Préalablement à toute étude, nous devons introduire un graphe en mémoire et calculer ses caractéristiques. Le graphe que nous utilisons sera toujours sans circuit.

1.2. Entrée des données et calcul des caractéristiques du graphe

Pour chaque tâche i de durée p_i, on calcule r_i et q_i par l'algorithme de Bellman.

Nous rappelons, par définition, que :

$r_i=l(0,i)$ et $q_i=l(i,n+1)-p_i$ où $l(i,j)$ est la valeur maximale d'un chemin allant de i à j dans le graphe conjonctif G. r_i s'interprète comme la date de



R1

i	1	2	5	6	9	10	13	14
a_{i1}	3	4	1	6	4	3	3	5

R2

i	3	4	7	8	11	12	15	16
a_{i2}	6	4	3	7	3	6	5	4

Les arcs du graphe conjonctif sont valués par la durée des tâches.

La disponibilité de chaque ressource est constante et égale à 10.

Figure 1. – Exemple de problème cumulatif.

disponibilité (encore appelée date au plus tôt) de la tâche i et q_i représente la longueur du chemin critique résiduel à partir de la tâche i. q_i sera appelée durée de latence.

De façon plus formelle :

$l(0,j) = \underset{i \in U^-(j)}{\text{Max}} \; (l(0,i) + p_i)$ où $U^-(j)$ désigne l'ensemble des prédécesseurs de j

et

$$l(i,n+1) = \underset{j \in U^+(i)}{\text{Max}} \; (P_j + l(j,n+1))$$

où $U^+(i)$ désigne l'ensemble des successeurs de i.

1.3. Stratégies de résolution

1.3.1. *Méthodes sérielles*

La démarche la plus courante pour établir un ordonnancement consiste à simuler l'état d'avancement des diverses tâches et à arbitrer dans le temps les

conflits qui apparaissent pour l'utilisation des ressources nécessaires à l'exécution des tâches. Ces conflits sont résolus à l'aide de *règles de priorités*, qui permettent de sélectionner, parmi les tâches en attente, une tâche exécutable par une ressource qui se libère. On aboutit ainsi à une construction ordonnée dans le temps d'un ordonnancement. Ces techniques utilisent assez bien les ressources disponibles puisque l'on exécute une tâche, dès que cela est possible.

Cependant, des exemples [GRAHAM 69], [CARLIER 88 A] montrent que l'optimum n'appartient pas toujours au domaine des solutions générées par les méthodes sérielles. De plus, les méthodes sérielles ne sont pas stables en ce sens que si l'on augmente la quantité de ressources disponibles, la durée de l'ordonnancement peut s'accroître.

Bien entendu, ces méthodes ne constituent que des heuristiques: on ne peut juger de la distance entre la solution obtenue et l'optimum.

On établit donc une méthode arborescente qui a pour objet de calculer une solution optimale du problème. Nous décrivons brièvement le principe de ces méthodes.

1.3.2. *Principe des méthodes arborescentes*

Le principe d'une méthode arborescente est d'explorer l'ensemble des solutions réalisables en le scindant en des sous-ensembles de plus en plus petits de façon à isoler une solution optimale. Pour définir une telle procédure, nous devons élaborer une règle de séparation qui permettra de construire l'arborescence, choisir une règle de parcours de l'arborescence et préciser les évaluations que nous utilisons. Notre séparation consiste à remplacer l'intervalle d'exécution possible d'une tâche par deux intervalles recouvrant l'intervalle initial. Notre évaluation par excès est fondée sur la règle de Jackson.

L'efficacité de ces méthodes dépend fortement de la qualité des évaluations par défaut. La nôtre est fondée sur la génération de problèmes à m machines obtenus en relâchant certaines contraintes du problème cumulatif.

2. GÉNÉRATION DE PROBLÈMES À m MACHINES ET ÉVALUATION PAR DÉFAUT

Le problème à m machines est par définition le problème cumulatif particulier pour lequel on a une seule ressource ($|K|=1$), la disponibilité de cette ressource vaut m ($b_k=m$), la demande de chaque tâche est 0 ou 1 ($a_{ik}=0$ ou 1) et les contraintes potentielles se réduisent à des dates de disponibilité r_i et

à des durées de latence q_i. Nous notons (r_i, p_i, q_i) les données d'un problème à m machines.

Pour la génération de problèmes à m machines, nous proposons deux niveaux de relaxation des contraintes. Dans le premier, nous ne conservons que les contraintes dues à une ressource. Dans le second, nous autorisons en outre la duplication des tâches. La durée optimale du problème à m machines de données (r_i, p_i, q_i) est une évaluation par défaut de la durée optimale du problème cumulatif [CARLIER 84].

Dans ce paragraphe, nous expliquons la génération des problèmes à m machines qui sont des outils permettant l'efficacité de notre méthode. Nous en déduisons l'évaluation par défaut $G'(J)$ que nous utilisons. Le lecteur peut, dans une première lecture, passer cette partie un peu technique et lire directement le paragraphe suivant où la méthode globale est expliquée.

2.1. Génération

2.1.1. *Premier niveau de relaxation*

Soit $k \in K$ une ressource et F l'ensemble des tâches utilisant cette ressource. Nous allons calculer des sous-ensembles E de F vérifiant la condition suivante :

$$\forall i_1, i_2, \ldots, i_{m+1} \in E, \quad a_{i_1,k} + a_{i_2,k} + \ldots + a_{i_{m+1},k} > b_k.$$

Cette condition exprime le fait que pas plus de m tâches de l'ensemble E ne pourront s'exécuter simultanément.

E est initialisé par :

$$E = \{ i \mid a_{ik} > (b_k/(m+1)) \}.$$ E définit un problème à m machines.

Ensuite, *on cherche à ajouter à E des tâches j de* $F \setminus E$ *telles que* :

$$a_{j,k} + a_{i_1,k} + \ldots + a_{i_m,k} > b_k$$

où :

$$a_{i_1,k} = \min_{i \in E} a_{ik}$$

$$a_{i_2,k} = \min_{i \in E - \{i_1\}} a_{ik}$$

$$\vdots$$

$$a_{i_m,k} = \min_{i \in E - \{i_1, i_2, \ldots, i_{m-1}\}} a_{ik}.$$

Chaque nouvel ensemble E définit un problème à m machines.

Le mécanisme ci-dessus peut être formalisé par l'algorithme général suivant :

Premier niveau de relaxation pour la génération de problèmes à m machines.
Début
– Introduire le nombre MAXSOL de problèmes à générer; m := 1
Répéter
Pour chaque ressource k du problème cumulatif ***Faire***
– Détermination de l'ensemble F des tâches utilisant la ressource k
Si F ≠ ∅ ***Alors***
– E = ensemble des tâches de F telles que $a_{ik} > b_k/(m+1)$ et tri de ces tâches par ordre des a_{ik} croissants
Si E = ∅ ***Alors***
– Pas de conflit d'ordre m + 1.
Sinon
{ Recherche si on peut ajouter à E des tâches de F\E }
AjouttachesEapartirF(E,F)
Fsi
Fp
Si nombre de problèmes à m machines < MAXSOL ***Alors*** m := m + 1 ***Fsi***
Jusqu'à nombre de problèmes générés = MAXSOL
Fin

AjouttachesEapartirF(E,F)
{ Recherche d'ajout à E des tâches de F\E dont la demande est inférieure ou égale à $b_k/(m+1)$ }
Début
– Recherche dans F\E de la tâche i de demande a_{ik} minimale.
Si on n'a pas trouvé de tâche i ***Alors***
Les tâches de E constituent un problème à m machines
Sinon
Si (a_{ik} + somme des m tâches de E de plus petites demandes $> b_k$) ***Alors***
E := E ∪ { i }
{ Appel récursif avec E }
AjouttachesEapartirF(E,F)
{ Appel récursif avec E\{i} }
AjouttachesEapartirF(E\{i}, F\{i})
Sinon
AjouttachesEapartirF(E,F\{i})
Fsi
Fsi
Fin

Pour illustrer ce processus, supposons que nous voulions déterminer les problèmes à une machine de l'exemple à une ressource ci-dessous :

Tâche i	1	2	3	4
Demande a_{i1}	3	6	5	4

, $b_1 = 8$

- Calcul de $b_1/2=4$, $F=\{1,2,3,4\}$.
- Constitution de E et tri par ordre de demande croissante $E=\{3,2\}$.
- On a : $i=4$ et $E=\{2,3,4\}$.

Le premier appel récursif avec E nous donne :

2 3 4 *est un problème à une machine.*

Le second appel récursif avec $E\setminus\{4\}$ fournit :

2 3 *est un problème à une machine.*

2.1.2. *Second niveau de relaxation*

La méthode consiste à reprendre chaque problème à m machines généré précédemment et à dupliquer certaines tâches de sorte que pas plus de m tâches ne puissent s'exécuter simultanément. Les tâches correspondantes sont contenues dans un ensemble E. Soient $a_{i_1,k}, a_{i_2,k}, \ldots, a_{i_p,k}$ les demandes des p tâches de l'ensemble E avec :

$$a_{i_1,k}=\min_{i\in E} a_{ik}$$

$$a_{i_2,k}=\min_{i\in E-\{i_1\}} a_{ik}$$

$$\vdots$$

$$a_{i_p,k}=\min_{i\in E-\{i_1,i_2,\ldots,i_{p-1}\}} a_{ik}.$$

Chaque problème à m machines vérifie la relation suivante :

$$\sum_{j=1}^{m+1} a_{i_j,k} > b_k.$$

Nous allons, pour le second niveau de relaxation, effectuer une partition des demandes des tâches tout en satisfaisant la condition ci-dessus. Pour cela, on convient de choisir la tâche l de plus grande demande a_{lk} que l'on note sous la forme (r_l, p_l, q_l, a_{lk}), on crée une nouvelle tâche $p+1$ et on pose :

$$\text{tâche } l : \left(r_l, p_l, q_l, \left\lceil \frac{a_{lk}}{2} \right\rceil\right),$$

$$\text{tâche } p+1 : \left(r_l, p_l, q_l, \left\lceil \frac{a_{lk}}{2} \right\rceil\right).$$

Ensuite, si $M_{m+1} = \{1, 2, \ldots, p, p+1\} = EU\{p+1\}$ désigne le nouvel ensemble de tâches et si $\sum_{j=1} a_{i_j, k} > b_k$ où :

$$a_{i_1, k} = \min_{i \in M} a_{ik}$$

$$a_{i_2, k} = \min_{i \in M - \{i_1\}} a_{ik}$$

$$\vdots$$

$$a_{i_{m+1}, k} = \min_{i \in M - \{i_1, i_2, \ldots, i_m\}} a_{ik}$$

on réitère le processus de partition de la tâche de plus grande demande. Dans le cas contraire, on enregistre le problème à m machines sans tenir compte des dernières modifications opérées. On obtient ainsi l'algorithme général suivant :

Second niveau de relaxation
Début
Pour chaque problème à m machines ***Faire***
générée par le premier niveau de relaxation
continue : = vrai
Répéter
Prendre la tâche i de plus grande demande a_{ik}, cette tâche est représentée par le quadruplet (r_i, p_i, q_i, a_{ik})
- Remplacer (r_i, p_i, q_i, a_{ik}) par :
$(r_i, p_i, q_i, [a_{ik}/2])$ puis créer une nouvelle tâche l $(l \geqq n+1)$ de quadruplet : $(r_i, p_i, q_i, [a_{ik}/2])$

Si le problème à m machines comporte plus de m tâches ***Alors***
Si somme des (m + 1) tâches de plus petites demandes $\leqq b_k$ ***Alors***
- Enregistrer dans une structure le problème à m machines sans tenir compte des modifications de la tâche i
continue : = faux

Fsi
Sinon
- Enregistrer le problème à m machines en tenant compte des modifications de la tâche

Fsi
jusqu'à non continue
Fp
Fin

Reprenons l'exemple extrait du paragraphe précédent. Supposons que nous voulions obtenir les problèmes à deux machines. Par le premier niveau de relaxation, l'ensemble constitué des tâches

1, 2, 3, 4 *est un problème à deux machines*

L'algorithme du second niveau décrit ci-dessus nous conduit à :

- prendre la tâche 2 de plus grande demande;
- mettre à jour sa demande;
- créer une nouvelle tâche 5 de demande 3.

On a alors :

Tâche i	1	2	3	4	5
Demande a_{i1}	3	3	5	4	3

Comme $3+3+3>8$, on continue le processus. Selon le même procédé, on obtient :

Tâche i	1	2	3	4	5	6
Demande a_{i1}	3	3	3	4	3	2

Or, $2+3+3>8$ est faux donc :

1 2 5 4 3 *est un problème à deux machines*

Dans notre cas, on ne garde que les problèmes à m machines issus du second niveau de relaxation. Les problèmes à m machines sont enregistrés dans une structure d'arbre binaire; le critère de placement tient compte de l'évaluation par défaut de chaque problème. Ainsi, pour sélectionner les problèmes d'évaluation par défaut maximale, il suffit d'effectuer un parcours *in-order* (Gauche Racine Droite) de l'arbre.

Nous allons donc étudier comment calculer l'évaluation par défaut de chaque problème à m machines, ces évaluations serviront pour la résolution du problème cumulatif.

2.2. Calcul de l'évaluation par défaut pour un problème à m machines

Nous proposons ci-dessous une évaluation par défaut associée à un ensemble J, le calcul de J sera explicité ultérieurement. Ensuite, nous introduisons une autre évaluation qui servira pour la méthode arborescente de résolution du problème cumulatif.

I représente l'ensemble des tâches du problème à m machines.

PROPOSITION 1 : (démonstration dans [CARLIER 82 A, 84]) : *Si J n'est pas vide* ($J \subset I$),

$$G(J) = \operatorname{Min}_{i \in J} r_i + \frac{\sum_{i \in J} p_i}{m} + \operatorname{Min}_{i \in J} q_i$$

est une évaluation par défaut de la durée optimale de l'ordonnancement.

PROPOSITION 2 : (démonstration dans [CARLIER 84]) : *Soient J un ensemble tel que* $\operatorname{card}(J) \geqq m$ *et* $i_1, i_2, \ldots, i_m$ (resp. $j_1, j_2, \ldots, j_m$) *les tâches de J telles que* :

$$r_{i_1} = \operatorname{Min}_{i \in J} r_i \qquad (\text{resp. } q_{j_1} = \operatorname{Min}_{j \in J} q_j)$$

$$r_{i_2} = \operatorname{Min}_{i \in J - \{i_1\}} r_i \qquad (\text{resp. } q_{j_2} = \operatorname{Min}_{j \in J - \{j_1\}} q_j)$$

$$\ldots$$

$$r_{i_m} = \operatorname{Min}_{i \in J - \{i_1, \ldots, i_{m-1}\}} r_i \qquad (\text{resp. } q_{j_m} = \operatorname{Min}_{j \in J - \{j_1, \ldots, j_{m-1}\}} q_j)$$

$$G'(J) = \frac{1}{m}(r_{i_1} + \ldots + r_{i_m} + \sum_{i \in J} p_i + q_{j_1} + \ldots + q_{j_m})$$

est une évaluation par défaut de la durée optimale de l'ordonnancement.

Pour pouvoir calculer une évaluation par défaut, il faut choisir un sous-ensemble J de l'ensemble I des tâches.

Remarquons que $\operatorname{Max}_{i \in I}(r_i + p_i + q_i)$ est également une évaluation par défaut.

2.2.1. *Choix de l'ensemble* $J (J \subset I)$

Lors de l'utilisation de la méthode arborescente pour le problème cumulatif, il est préférable de choisir un ensemble J maximisant l'évaluation par défaut.

D'après J. CARLIER [CARLIER 84, 88B], il est possible de calculer un ensemble J maximisant $G(J)$ en employant la solution préemptive du problème à une machine. On associe à la donnée (r_i, p_i, q_i) d'un problème à m machines, le triplet (mr_i, p_i, mq_i) d'un problème à une machine.

Définition du problème à une machine

L'étude du problème à une machine apparaît lorsque chaque tâche requiert une ressource dont la disponibilité est égale à l'unité. On doit ordonnancer n tâches, en une durée minimale, en respectant les conditions suivantes: une tâche i est disponible à la date r_i, a une durée p_i et un laps de temps q_i doit s'écouler entre la fin de la tâche i et l'achèvement de l'ordonnancement. Le problème est NP-difficile [CARLIER 82A].

Pour le problème à m machines (r_i, p_i, q_i), l'évaluation par défaut $G(J)$ vaut:

$$G(J) = \operatorname*{Min}_{i \in J} r_i + \frac{\sum_{i \in J} p_i}{m} + \operatorname*{Min}_{i \in J} q_i.$$

La même évaluation $f(J)$ pour le problème à une machine associé (mr_i, p_i, mq_i) est:

$$f(J) = m \operatorname*{Min}_{i \in J} r_i + \sum_{i \in J} p_i + m \operatorname*{Min}_{i \in J} q_i.$$

Or,

$$G(J) = \frac{1}{m}\left(m \operatorname*{Min}_{i \in J} r_i + \sum_{i \in J} p_i + m \operatorname*{Min}_{i \in J} q_i\right) = \frac{1}{m} f(J).$$

Avec l'algorithme de Jackson préemptif, que nous allons décrire, on calcule un ensemble J maximisant $f(J)$ et donc $G(J)$.

2.2.2. *Principe de l'algorithme de Jackson préemptif et du calcul de J*

La solution préemptive de Jackson est obtenue en appliquant l'algorithme de liste donnant priorité à la tâche disponible de q_i maximal. La tâche est exécutée jusqu'à son terme ou est interrompue lorsqu'une tâche plus prioritaire devient disponible. On met à jour la date t et on réitère le processus jusqu'à ce que toutes les tâches soient placées [CARLIER 88B].

Ensuite, on calcule la durée de la solution de Jackson, ce qui revient à déterminer la tâche t_{maxi} maximisant $(r_i + p_i + q_i)$.

PROPOSITION 3 [CARLIER 84] : *Soit I l'ensemble des tâches. Pour toute tâche j, il existe $J \subset I$ tel que :*

- $j \in J$;
- $C_j = \operatorname*{Min}_{k \in J} r_k + \sum_{k \in J} p_k$ (C_j: *adresse de fin de placement de la tâche j;*)
- $\forall k \in J,\ k \neq j,\ q_k > q_j$.

L'algorithme général de calcul de J est :

Début
- *Établissement de la solution préemptive de Jackson*
- *Calcul effectif de J*

Fin

Établissement de la solution préemptive de Jackson

Principales variables :

cpt : compteur de tâches placées
t_{maxi} : tâche maximisant $r_i + p_i + q_i$
C_j : adresse de fin de placement de la tâche j

Début

Prochainedatelibre : $= \underset{i \in I}{\text{Min}}\, r_i$
cpt : = 0
Répéter
- Déterminer la tâche j disponible la plus prioritaire ($r_j \leqq$ Prochainedate libre et q_j maximale) parmi toutes les tâches non encore totalement placées ($p_j \neq 0$)
- Déterminer la date de disponibilité $datet_i$ immédiatement supérieure à r_j parmi toutes les tâches non encore totalement placées ($datet_i = +\infty$ si toutes les tâches sont disponibles)

Si $datet_i \geqq$ Prochainedatelibre $+ p_j$ *Alors*
{On peut complètement placer la tâche j}
Prochainedatelibre: = Prochainedatelibre $+ p_j$
p_j: = 0; cpt: = cpt + 1
- Mise à jour éventuelle de la tâche t_{maxi}

Sinon
{Placement partiel de la tâche j}
p_j : = $p_j - datet_i$ + Prochainedatelibre
Prochainedatelibre: = $datet_i$

Fsi

Jusqu'à cpt = nombre de tâches à placer

Fin

Calcul effectif de J

Début
- Choix de la tâche j maximisant $C_i + q_i$, $i \in I$
- Constitution d'un tableau T de tâches k tel que $q_k > q_j$ et tri de ces tâches par ordre décroissant des valeurs de C_k

{Détermination de l'ensemble J en ajoutant les éléments de T un par un à J et en calculant : $C = \underset{k \in J}{\text{Min}}\, r_k + \sum_{k \in J} p_k$. L'ensemble J est constitué quant $C = C_j$}

J : = {j}; bcl : = 1; sortie : = faux;
Répéter
{Calcul de C}
$r_{kmin} = +\infty$
Somp : = 0
Pour chaque tâche $k \in J$ ***Faire***
Somp : = Somp + p_k
Si $r_k < r_{kmin}$ ***Alors*** $r_{kmin} := r_k$ ***Fsi***
Fp
sortie : = r_{kmin} + Somp = C_j

```
    Si non sortie Alors
        J := J ∪ {T [bcl]}
        bcl := bcl + 1
    Fsi
Jusqu'à sortie ou bcl > nombre d'éléments de T
Si non sortie Alors J := {j} Fsi
Fin
```

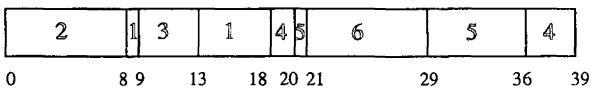
L'application de l'algorithme de calcul de J à l'exemple ci-dessous, extrait de [CARLIER 88B], donne :

$j=5$, $C_5=36$, $T=[6, 1, 3, 2]$ d'où $J=\{5\}$ et $C=20+8=28$. On réitère le processus avec $J=\{5, 6\}$, on a $C=20+8+8=36=C_5$; on obtient finalement : $J=\{5, 6\}$.

L'étude des problèmes à m machines et à une machine permet de calculer une évaluation par défaut du problème cumulatif. Nous pouvons maintenant présenter la procédure arborescente basée sur la méthode des intervalles.

i	1	2	3	4	5	6
r_i	4	0	9	15	20	21
p_i	6	8	4	5	8	8
q_i	20	25	30	9	14	16

L'algorithme de Jackson conduit à :



De plus :

i	1	2	3	4	5	6
C_i	18	8	13	39	36	29
$C_i + q_i$	38	33	43	48	50	45

Figure 2. – Illustration de l'algorithme de Jackson préemptif.

3. LA MÉTHODE DES INTERVALLES

Il s'agit d'une méthode arborescente. Il faut donc spécifier:

- l'évaluation par défaut d'un nœud N [notée $f$défaut (N)];
- l'évaluation par excès (notée f excès);
- le mode de séparation d'un nœud;
- la stratégie de parcours de l'arbre.

L'évaluation par défaut utilise les problèmes à m machines: on choisit, pour l'ensemble des problèmes sélectionnés, celle maximisant $G'(J)$. Il faut aussi choisir une évaluation par excès du problème cumulatif.

3.1. Évaluation par excès

Dans la méthode des intervalles, l'évaluation par excès est basée sur la construction d'un ordonnancement. A chaque nœud de l'arborescence, nous appliquons la règle de Jackson au problème cumulatif et affectons des ressources aux tâches. Il s'agit donc de la méthode sérielle donnant priorité à la tâche de plus grand q_i.

3.2. Séparation en un nœud de l'arborescence

Les données initiales d'un problème cumulatif sont composées de trois paramètres; pour chaque tâche i, on a sa durée p_i, sa date de disponibilité r_i et sa durée de latence q_i. Nous en introduisons un quatrième, la date échue d_i en posant:

$$d_i = f\text{excès} - q_i, \qquad i \in I.$$

Nous justifions cette relation. Dans la méthode arborescente, nous cherchons des solutions optimales. Notons f^* la durée optimale. Une durée de latence est alors équivalente à une date échue $d_i = f^* - q_i$. f^* étant inconnue, nous *l'évaluons par excès avec f excès*. Remarquons également que $f$défaut (N) *permet d'évaluer par défaut* f^*. Ceci justifie les opérations d'ajustement que l'on effectue en chacun des nœuds de l'arborescence en posant:

$$\begin{aligned} q_i &:= \text{Max } (q_i, f\text{défaut } (N) - d_i) \\ d_i &:= \text{Min } (d_i, f\text{excès} - q_i) \end{aligned}$$

L'introduction de d_i permet d'associer à chaque tâche l'intervalle $[r_i, d_i]$ où elle devra s'exécuter. Ceci nous permet de définir le principe de séparation. Il consiste à remplacer l'intervalle associé à une tâche par deux intervalles

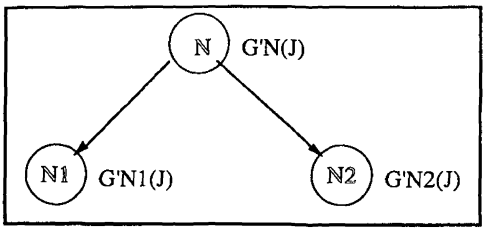
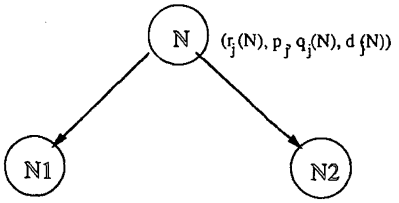


Figure 3. – Illustration du phénomène de séparation.

dont l'union redonne l'intervalle initial (*fig.* 3). Nous choisissons ensuite la tâche dont l'intervalle sera séparé.

3.2.1. *Séparation effective*

A un nœud N, on a, pour le problème cumulatif, les quantités $(r_i(N), p_i, q_i(N), d_i(N))$; on choisit une tâche j (*voir* ci-dessous) et on introduit deux nouveaux nœuds $N1$ et $N2$. On a :



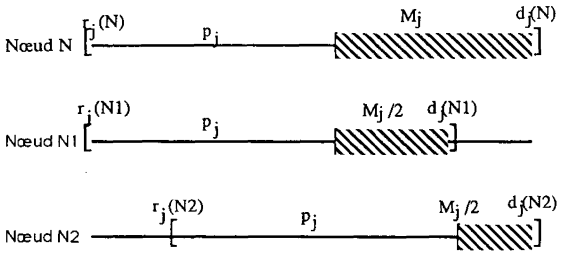
$(r_j(N1), p_j, q_j(N1), d_j(N1))$ $\quad$ $(r_j(N2), p_j, q_j(N2), d_j(N2))$

Nous appelons *marge de la tâche j la quantité* $M_j = d_j(N) - r_j(N) - p_j$, les seuls paramètres modifiés entre $N1$, $N2$ et N sont $d_j(N1)$ et $r_j(N2)$, d'où :

$$r_j(N2) = r_j(N) + [M_j/2]$$

$$d_j(N1) = r_j(N2) + p_j - 1.$$

Sous forme d'intervalles, les problèmes associés aux nœuds N, $N1$ et $N2$ peuvent se représenter par :



Les modifications des valeurs de r_i, q_i et d_i sont transmises à chacun des problèmes à m machines sélectionnés. L'intervalle est choisi de telle sorte que l'évaluation par défaut progresse.

3.2.2. *Choix de la tâche dont l'intervalle sera séparé*

Le nœud N est séparé en deux nœuds fils $N1$ et $N2$. La tâche j est choisie de telle sorte que l'évaluation par défaut associée à J soit la plus grande possible en $N1$ et en $N2$. On rappelle la valeur de $G'(J)$:

$$\text{pour} \quad |J| \geqq m, \; G'(J) = \frac{1}{m}(r_{i_1} + \ldots + r_{i_m} + \sum_{i \in J} p_i + q_{j_1} + \ldots + q_{j_m}).$$

Le schéma de la figure 3 illustre le phénomène de séparation. Le but est de maximiser la quantité :

$$G'_{N2}(J) - G'_N(J) + G'_{N1}(J) - G'_N(J) \quad \text{sur } j \in J \; (^1).$$

Notons : $\lambda_j = G'_{N2}(J) - G'_N(J)$, $\mu_j = G'_{N1}(J) - G'_N(J)$.

Calculons la quantité λ_j.

(1) $G'_{N1}(J)$ et $G'_{N2}(J)$ sont des abus d'écriture dans la mesure où $N1$ et $N2$ dépendent du choix de j. Cet abus d'écriture permet d'alléger les notations.

Deux cas :

si $j \notin \{i_1, \ldots, i_m\}$

$$G'_{N2}(J) = \frac{1}{m}(r_{i_1} + \ldots + r_{i_m} + \sum_{i \in J} p_i + q_{j_1} + \ldots + q_{j_m})$$

$$G'_N(J) = G'_{N2}(J) \quad \text{donc } \lambda_j = 0.$$

si $j \in \{i_1, \ldots, i_m\}$

On a :

$$G'_N(J) = \frac{1}{m}(r_{i_1} + \ldots + r_j + \ldots + r_{i_m} + \sum_{i \in J} p_i + q_{j_1} + \ldots + q_{j_m}).$$

Au nœud $N2$, deux cas peuvent se produire :

(i) *le nouveau* $r_j \in \{i_1, \ldots, i_m\}$

Dans ce cas,

$$G'_{N2}(J) = \frac{1}{m}(r_{i_1} + \ldots + r_j + \ldots + r_{i_m} + \sum_{i \in J} p_i + q_{j_1} + \ldots + q_{j_m}).$$

On a alors :

$$\lambda_j = \frac{1}{m}[r_j(N2) - r_j(N)].$$

(ii) *le nouveau* $r_j \notin \{i_1, \ldots, i_m\}$ (c'est-à-dire qu'il n'est pas parmi les m plus petites valeurs de r_i).

Comme

$$G'_{N2}(J) = \frac{1}{m}(r_{i_1} + \ldots + r_{i_m} + r_{i_{m+1}} + \sum_{i \in J} p_i + q_{j_1} + \ldots + q_{j_m})$$

$$\lambda_j = \frac{1}{m}[r_{i_{m+1}}(N) - r_j(N)].$$

Finalement :

$$\lambda_j = \frac{1}{m}[\text{Min}(r_j(N2) - r_j(N), r_{i_{m+1}}(N) - r_j(N))].$$

Un calcul similaire pour μ_j, à partir des q_j, nous conduit à :

$$\mu_j = \frac{1}{m}[\text{Min}\,(q_j(N1) - q_j(N), q_{i_{m+1}}(N) - q_j(N))]$$

La tâche j est choisie de telle sorte que $\lambda_j + \mu_j$ *soit maximale.*

Cependant si $\lambda_j + \mu_j = 0$, on prend celle de plus grande marge $M_j = d_j - r_j - p_j$ parmi les tâches de $\{i_1, i_2, \ldots, i_m, j_1, j_2, \ldots, j_m\}$.

Une fois fixé le principe de séparation, il reste à choisir le nœud pendant qui sera séparé. Nous avons implémenté les deux stratégies les plus employées : l'exploration en profondeur (S.E.S.) et en largeur d'abord (S.E.P.).

3.3. Procédure par Séparation et Évaluation Progressive (S.E.P.)

Elle sépare le nœud de l'arborescence d'évaluation par défaut minimale. Cela revient à réaliser une exploration en largeur d'abord. Elle est assez coûteuse en place mémoire mais reste intéressante car elle permet de trouver de bonnes solutions réalisables. Il faut disposer en mémoire d'une liste des nœuds non encore explorés. L'algorithme se termine lorsque l'évaluation par défaut devient supérieure ou égale à celle par excès ou lorsque toutes les tâches ont une marge nulle.

3.4. Procédure par Séparation et Évaluation Séquentielle (S.E.S.)

Elle sépare le nœud de l'arborescence le plus proche du dernier nœud considéré. Cela revient à réaliser une exploration en profondeur d'abord. Ce type de procédure est en général assez facile à mettre en œuvre par des procédures récursives et est utilisé assez souvent.

L'emploi de la récursivité permet d'éviter l'usage d'une pile explicite pour le « backtracking ». Le retour arrière s'effectue lorsque l'évaluation par défaut d'un nœud devient supérieure ou égale à l'évaluation par excès ou quand toutes les tâches du nœud ont une marge nulle. De plus, la stratégie est peu coûteuse en place mémoire.

Un algorithme général peut s'écrire comme suit :

Méthode des intervalles par S.E.S.

Début

- Création d'un nœud N correspondant au problème initial avec fdéfaut(N) : = 0
- Calcul de la solution de Jackson pour le nœud N afin d'obtenir une évaluation par excès fexcès
- Calcul des dates échues d_i pour le nœud N
- Résolution Intervalles Profondeur(N, fexcès)

Fin

Résolution Intervalles profondeur(N, fexcès)

N : nœud courant
fexcès : évaluation par excès
Début
- Ajustement des q_i et des d_i pour le nœud N
- Recopie des valeurs de r_i, q_i et d_i du nœud N dans chacun des problèmes à m machines
- Calcul de la solution de Jackson pour le nœud N (on obtient une évaluation f)

Si f < fexcès ***Alors***

fexcès : = f
- Afficher la nouvelle solution trouvée

Fsi

Si fdéfaut(N) ≧ fexcès ***Alors***
- Suppression du nœud N

Sinon
- Choix de la tâche j à séparer et calcul de la nouvelle évaluation par défaut gmax
- Mise à jour éventuelle de la valeur par défaut en posant :
  fdéfaut(N) : = Max(gmax, fdéfaut(N))

Si fdéfaut(N) < fexcès ***et*** au moins une des tâches n'a pas une marge nulle ***Alors***
- Création de deux nœuds N1 et N2 identiques à N
- Suppression du nœud N
- Modification de paramètres des nœuds N1 et N2 (respectivement d_j et r_j)
- Application éventuelle de l'algorithme de Bellman pour calculer les nouvelles valeurs de r_i et de q_i

{ Appel récursif pour N1 }
Résolution Intervalles Profondeur (N1,fexcès)
{ Appel récursif pour N2 }
Résolution Intervalles Profondeur(N2,fexcès)

Sinon
- Suppression du nœud N

Fsi

Fsi

Fin

3.5. Exemple numérique

Nous allons appliquer la nouvelle méthode de résolution à l'exemple de la figure 4. Il comporte 6 tâches, 1 ressource. Les tâches sont numérotées de 1 à 6. Les deux tâches fictives sont 0 et 7. Les nombres au-dessus de chaque nœud du graphe correspondent aux durées respectives des tâches. La demande en ressource de chaque tâche est indiquée au-dessous du nœud. La disponibilité de la ressource est 9.

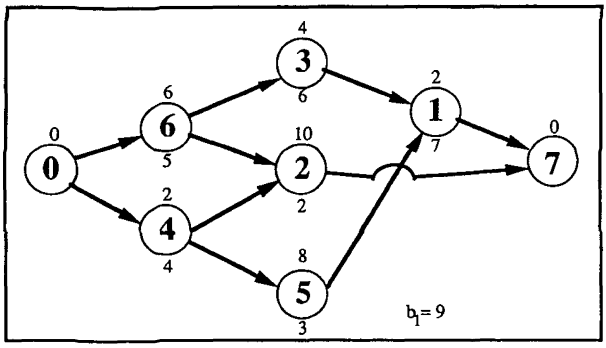


Figure 4. – Exemple d'application.

Préalablement, nous déterminons les paramètres (r_i, q_i) par l'algorithme de Ballman. Nous obtenons :

Tâche i	r_i	q_i
1	10	0
2	6	0
3	6	2
4	0	10
5	2	2
6	0	10

On cherche ensuite à générer les problèmes à m machines par les deux niveaux de relaxation. Le premier niveau nous fournit :

1 3 6 : **problème à une machine,**

1 3 4 5 6 : **problème à deux machines,**

1 2 3 4 5 6 : **problème à trois machines.**

Le second niveau de relaxation reprend chaque problème à m machines généré précédemment et crée éventuellement de nouvelles tâches en partitionnant les demandes. On obtient ainsi :

1 3 6 : **problème à une machine,**

1 3 4 5 6 8 : **problème à deux machines,**

Les demandes des tâches associées à ce problème ont été mises à jour comme suit :

Tâche i	1	3	4	5	6	8
a_{i1}	4	6	4	3	5	3

1 2 3 4 5 6 8 9 10 : problème à trois machines.

Les demandes des tâches sont :

Tâche i	1	2	3	4	5	6	8	9	10
a_{i1}	4	2	3	4	3	3	3	3	2

Finalement, les problèmes à m machines sont :

1 3 6 : problème à une machine,

1 3 4 5 6 8 : problème à deux machines,

1 2 3 4 5 6 8 9 10 : problème à trois machines.

On remarquera que la génération des problèmes à m machines n'a lieu qu'une seule fois au début de l'algorithme.

On applique maintenant la méthode des intervalles; on choisit ici la stratégie en profondeur d'abord. On détermine la solution de Jackson pour le problème initial (*fig.* 5).

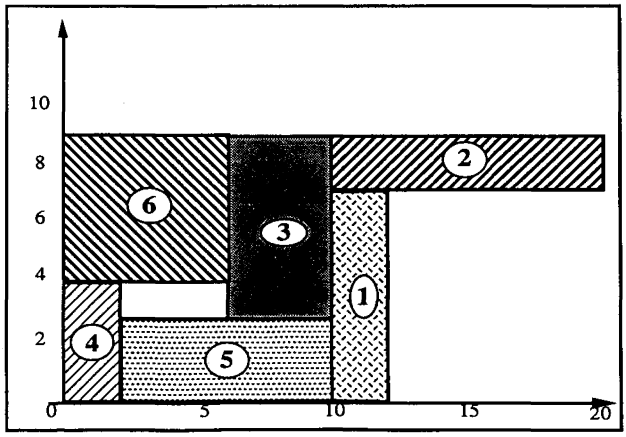


Figure 5. – Diagramme de Gantt de la solution de Jackson associée au nœud N1.

On calcule ensuite les valeurs de d_i pour chaque tâches :

Tâche i	1	2	3	4	5	6
d_i	20	20	18	10	18	10

Le nœud **N1** correspond au problème initial, la solution de Jackson est l'évaluation par excès de ce nœud (*fig.* 7). On détermine la tâche j à séparer; on trouve $j=2$. On sépare en créant deux nœuds **N2** et **N3** identiques au nœud **N1** exceptés les paramètres (q_2, d_2) pour le nœud **N2** et r_2 pour le nœud **N3**. Après mise à jour, on a : $(q_2, d_2)=(3, 17)$ et $r_2=8$.

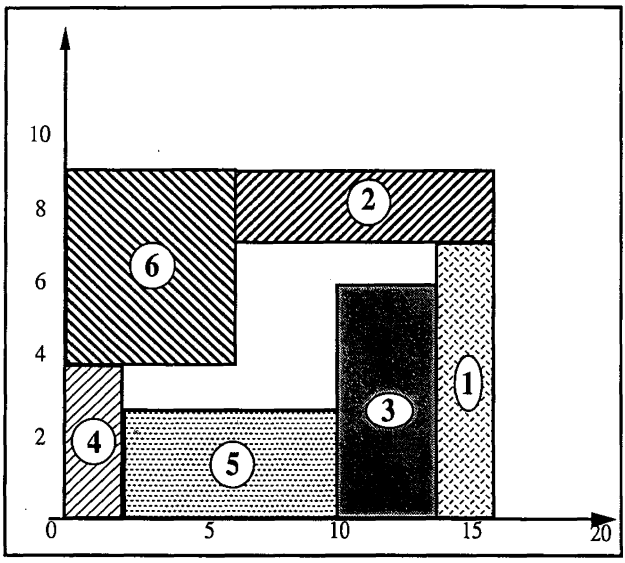
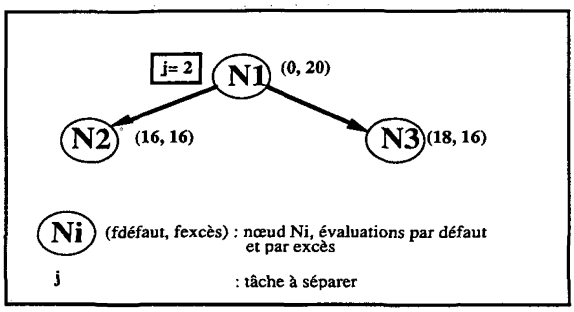


Figure 6. – Diagramme de Gantt de la solution de Jackson associée au nœud N2.

On détermine au nœud **N2** une nouvelle solution de Jackson (*fig.* 6) qui améliore la meilleure solution connue. On met à jour l'évaluation par excès du nœud **N3** (*f*excès = 16). Comme *f*défaut (= 18) ≧ *f*excès (= 16), on ne développe pas l'arborescence.

Finalement, la solution optimale 16 a été obtenue au nœud **N2**. L'arborescence est représentée par la figure 7.

Nous allons terminer en donnant les principales structures de données, la méthodologie de génération d'un problème cumulatif aléatoire est les résultats obtenus par la méthode.



Après obtention de la solution de Jackson, on met à jour l'évaluation par excès ($f$excès = 16).

Figure 7. – Arborescence obtenue à partir de l'exemple.

4. STRUCTURES DE DONNÉES, GÉNÉRATION ALÉATOIRE ET RÉSULTATS

4.1. Principales structures de données utilisées

Nous allons préciser les structures pour représenter un problème cumulatif, les problèmes à m machines, une solution et l'arborescence de la méthode des intervalles.

Pour représenter le problème cumulatif initial, on dispose, pour chaque tâche, des ressources nécessaires et de leurs demandes, du quadruplet (r_i, p_i, q_i, d_i) et d'un pointeur vers ses successeurs. Comme on ne connaît pas à l'avance le nombre de successeurs pour un sommet donné, on emploie une liste chaînée composée des successeurs.

Pour pouvoir sélectionner rapidement les problèmes à m machines, on utilise un arbre binaire dont chaque nœud, représentant un problème à m machines, comporte l'évaluation par défaut et des pointeurs vers ses deux fils.

Pour la méthode des intervalles, chaque nœud de l'arborescence est composé uniquement de son évaluation par défaut et des paramètres spécifiques à un nœud (r_i, q_i, d_i). Il n'est pas nécessaire de disposer de pointeurs vers les deux fils car le « backtracking » est automatique au moyen des algorithmes récursifs.

On a enfin une structure permettant de connaître la meilleure solution établie.

4.2. Génération aléatoire et résultats obtenus

Pour tester la validité de la méthode, nous avons utilisé les problèmes de PATTERSON, d'ALVAREZ et des problèmes générés aléatoirement. Ils représentent l'ensemble des problèmes test relatifs aux ordonnancements à contraintes cumulatives aujourd'hui disponibles dans la littérature. Les problèmes de Patterson comportent de 7 à 51 tâches et le nombre de ressources est compris entre 1 et 3. Nous rapportons un échantillon représentatif des résultats. Les problèmes d'Alvarez testés comportent jusqu'à 51 tâches et 6 ressources. La solution optimale n'est pas rapportée car, à notre connaissance, seules des heuristiques ont déjà été testées sur ces problèmes.

Pour générer un problème aléatoire, nous employons la méthode congruentielle de Lehmer [SEDGEWICK 84]. L'utilisateur introduit le nombre de tâches, le nombre de ressources, la valeur maximale pour la durée des tâches, la probabilité qu'un arc appartienne au graphe (ce paramètre influe sur le nombre de contraintes conjonctives du graphe), la disponibilité maximale pour l'ensemble des ressources et la probabilité que les tâches utilisent une ressource donnée.

Nous avons également expérimenté la méthode sur des flow-shops généralisés pour lesquels la disponibilité d'une machine n'est pas égale à 1 mais est une valeur tirée au hasard entre 1 et 3. Les problèmes de flow-shop généralisé ont été fournis par E. Pinson [PINSON 88]. Ils représentent l'ensemble des jeux de données que nous avons pu nous procurer.

Le logiciel, testé sur un ordinateur VAX 8530, représente quelques 6 000 lignes de code source en langage Pascal. L'arborescence est limitée à 300 nœuds au plus.

Nous avons pu comparer nos solutions à l'optimum pour les exemples de Patterson. Toutefois, cela n'a pas été possible pour les exemples aléatoires, les problèmes d'Alvarez et les flow-shops généralisés. En effet, nous ne disposons pas des logiciels basés sur la génération d'ordonnancements actifs. Il apparaît que notre méthode est moins performante, sur les exemples de Patterson, que le meilleur algorithme connu à ce jour [HERROELEN 90]. Toutefois, sur l'ensemble des exemples testés on obtient toujours des solutions proches de la solution optimale; ceci confirme l'intérêt de cette méthode. Compte tenu de la qualité de notre évaluation par défaut, notre méthode devrait être performante quand les ressources sont dominantes.

Nous utilisons pour les tableaux de résultats les notations suivantes :

n : le nombre de tâches;

m : le nombre de ressources;

lap : la liste des améliorations successives obtenues au cours de l'algorithme *en profondeur d'abord*, la notation $x(y)$ indique que la solution x a été établie après avoir développé y nœuds de l'arborescence;

edp : l'évaluation par défaut de l'arborescence pour la stratégie *en profondeur d'abord* obtenue après avoir séparé 50 nœuds;

lal : la liste des améliorations successives obtenues au cours de l'algorithme *en largeur d'abord*, la notation $x(y)$ indique que la solution x a été établie après avoir développé y nœuds de l'arborescence;

edl : l'évaluation par défaut de l'arborescence pour la stratégie *en largeur d'abord* obtenue après avoir séparé 50 nœuds;

jks : la valeur de la première solution trouvée par l'heuristique de Jackson;

cpm : la valeur du chemin critique obtenue en ne tenant compte que des contraintes potentielles;

opt : la valeur de l'optimum ou de la meilleure solution connue.

CONCLUSION

Nous avons développé une *procédure arborescente basée sur la méthode des intervalles* [CARLIER 87B] pour la résolution des problèmes cumulatifs. Cette *technique est efficace car, à chaque nœud, on calcule non seulement une nouvelle évaluation par défaut mais également une évaluation par excès* toutes deux de bonne qualité. De plus, la séparation permet de faire progresser ces évaluations et les données du problème associé à un nouveau nœud ne diffèrent que d'un paramètre par rapport à celles du nœud courant.

Nous avons implémenté les méthodes en largeur d'abord (S.E.P.) et en profondeur d'abord (S.E.S.). *La stratégie en largeur d'abord* fournit parfois la solution optimale plus rapidement que la technique en profondeur d'abord car elle n'envisage que les nœuds « prometteurs »; elle est néanmoins coûteuse en place mémoire. Cependant, *la résolution en profondeur d'abord* ne nécessite qu'une faible place mémoire et semble plus performante dès que le problème est de taille importante. Un avantage est que nous n'introduisons pas de nouvelles contraintes de succession en cours d'algorithme; le graphe conjonctif reste donc invariant pour toute l'arborescence. Nous remplaçons l'intervalle associé à une tâche par deux intervalles dont l'union redonne l'intervalle initial. Du fait même de la méthode, seuls deux paramètres sont modifiés

TABLEAU I

Problème	n	m	PROFONDEUR		LARGEUR		jks	cpm	opt
			lap	edp	lal	edl			
(P)	7	3	---	7	---	7	7	6	7
(P)	9	1	8(3)	8	8(85)	8	10	8	8
(P)	8	2	14(109)	14	14(2)	14	16	14	14
(P)	8	2	18(79)	18	18(244)	18	20	14	18
(P)	22	3	23(2) - 22(16)	19	23(3)	19	25	18	22
(P)	22	3	22(9) - 21(20)	18	20(2)	18	23	18	20
(R)	15	3	42(5) - 40(26)	29	42(8) - 41(28) - 40(244)	30	43	22	40
(P)	27	3	37(5)-36(6)-35(15)-34(26)	26	38(9) - 37(76) - 34(136)	29	39	22	34
(P)	22	3	37(4) - 35(7) - 33(12)	22	38(11)-37(19)-36(25) -35(56)-34(281)	29	39	22	33
(P)	22	3	40(4)	37	41(287)	37	42	37	37 (*)
(P)	27	3	36(3) - 35(40)	22	36(6) - 34(17)	25	37	22	34
(P)	27	3	41(3) - 40(244)	31	41(7)	33	42	29	40
(P)	27	3	32(84)	29	---	29	35	29	32

(A) : jeu de données d'Alvarez **(R) : jeu de données généré aléatoirement**

(P) : jeu de données de Patterson **(*) : la meilleure solution connue n'a pas été obtenue après 2 heures de temps CPU sur un VAX 8530**

TABLEAU II

Problème	n	m	PROFONDEUR		LARGEUR		jks	cpm	opt
			lap	edp	lal	edl			
(P)	27	3	71(7) - 70(34)	54	70(24) - 64(36)	59	73	31	64
(P)	35	2	---	43	----	43	43	43	43
(R)	36	3	87(2)	71	92(6) - 87(17)	65	93	65	87
(P)	51	3	91(13)	83	92(65)	83	93	83	83(*)
(P)	51	3	87(2) - 85(9) - 82(18) - 81(19)	71	87(3) - 85(6) - 75(46)	71	89	71	75
(R)	52	3	167(2)-164(8)-161(10)	140	174(3) - 169(8)	150	176	34	161
(A)	27	6	53(3) - 46(4) - 45(6)	41	51(3) - 49(11) - 48(284)	41	54	41	45
(A)	27	6	76(2) - 65(5) - 64(84)	51	63(4) - 61(30)	45	81	45	61
(A)	27	6	65(4) - 61(6) - 60(18)	43	61(21)	52	66	43	60
(A)	27	6	79(3) - 78(56)	60	80(4) - 79(98)	56	86	54	78
(A)	27	6	111(26) - 102(47)	63	111(19)-109(23)-107(24)	69	113	52	102
(A)	51	6	108(5)	99	111(2)	98	117	98	108
(A)	51	6	112(3) - 109(5)	94	112(5)	94	117	94	109

(A) : jeu de données d'Alvarez **(R) : jeu de données généré aléatoirement**

(P) : jeu de données de Patterson **(*) : la meilleure solution connue n'a pas été obtenue après 2 heures de temps CPU sur un VAX 8530**

TABLEAU III

Résultats sur des exemples de « Flow-Shop » généralisé.

Problème	n	m	PROFONDEUR		LARGEUR		jks	cpm
			lap	edp	lal	edl		
(F)	20	4	---	13	---	8	14	8
(F)	38	6	67(14) - 66(57)	61	67(14) - 66(97)	65	71	47
(F)	52	5	972(3) - 950(20) - 944(72)	930	980(2) - 944(31)	920	1011	413
(F)	52	5	936(2) - 920(11) - 890(22) - 872(47)	851	890(3)	863	946	394
(F)	52	5	692(2) - 689(21) - 686(43)	673	686(10)	668	694	349
(F)	52	5	780(3) - 768(10) - 742(45)	712	742(8) - 715(39)	683	794	369
(F)	52	5	----	616	----	603	648	380
(F)	77	5	1197(4) - 1157(43)	1101	1157(27)	1016	1237	443

(F) : "flow-shop" généralisé

lors de la séparation d'un nœud de l'arborescence. Ces opérations sont donc très rapides.

Les expériences numériques que nous obtenons sont déjà satisfaisantes. En effet, notre méthode dans sa forme actuelle, se compare favorablement aux méthodes exactes basées sur la P.L.N.E. [PATTERSON 76]. Toutefois, sur les exemples de J. H. Patterson, une stratégie proposée par W. Herroelen et E. Demeulemeester [HERROELEN 90] et fondée sur la génération d'ordonnancements actifs est plus efficace.

Notre démarche pour résoudre les problèmes cumulatifs est originale en particulier pour le calcul de l'évaluation par défaut et le parcours de l'arborescence. La méthode est récente et vraisemblablement améliorable. Nous espérons que les nouvelles directions que nous proposons permettront de stimuler les recherches sur les problèmes cumulatifs.

REMERCIEMENTS

Les auteurs remercient vivement les Professeurs R. Alvarez-Valdes et J. H. Patterson d'avoir communiqué le contenu de leurs jeux de données. Sans leur aimable collaboration, cet article n'aurait pas pu être écrit. Nous remercions également E. Demeulemeester et le Professeur W. Herroelen de nous avoir tenus au courant de leurs travaux.

BIBLIOGRAPHIE

[ALVAREZ 88] R. ALVAREZ-VALDES and J. M. TAMARIT, Computational Comparison of Classical and new Heuristic Algorithms for Resource-constrained Project Scheduling, *International Workshop on Project Management and Scheduling*, Lisbon, juillet 1988.

[CARLIER 78] J. CARLIER, Ordonnancement à contraintes disjonctives, *RAIRO, Rech. Opér.*, 1978, *12*, n° 84, p. 333-350.

[CARLIER 82A] J. CARLIER, One Machine Problem, *Eur. J. Oper. Res.*, 1982, *11*, p. 42-47.

[CARLIER 82B] J. CARLIER, P. CHRETIENNE, Un domaine très ouvert : les problèmes d'ordonnancements, *RAIRO, Rech. Opér.*, 1982, *16*, p. 175-217.

[CARLIER 84] J. CARLIER, Problèmes d'ordonnancements à contraintes de ressources : algorithmes et complexité, *Thèse d'État*, Université de Paris-VI, mai 1984.

[CARLIER 87A] J. CARLIER et E. PINSON, Résolution d'un job-shop 10×10, *Rapport interne UTC GI HEUDIASYC*, 1987.

[CARLIER 87B] J. CARLIER, Scheduling Jobs with Release Dates and Tails on Identical Machines to Minimize Makespan, *European J. Oper. Res.*, 1987, *29*, p. 298-306.

[CARLIER 88A] J. CARLIER et P. CHRETIENNE, Problèmes d'ordonnancements : modélisation, algorithmes et complexité, *Masson*, Paris, 1988.

[CARLIER 88B] J. CARLIER and E. PINSON, The use of Jackson Preemptive Schedule for Solving the Job-Shop Problem, *International Workshop on Project Management and Scheduling*, Lisbon, juillet 1988.

[CHRISTOFIDES 87] N. CHRISTOFIDES, R. ALVAREZ-VALDES and J. M. TAMARIT, Project Scheduling with Resource Constraints: A Branch and Bound Approach, *European J. Oper. Res.*, 1987, *29*, p. 262-273.

[COOPER 76] D. F. COOPER, Heuristics for Scheduling Resource-Constrained Projects: an Experimental Investigation, *Management Sci.*, 1976, *22*, n° 11.

[DAVIS 75] E. DAVIS and J. PATTERSON, A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling, *Management Sci.*, 1975, *21*, n° 8.

[GAREY 79] M. R. GAREY and D. S. JOHNSON, Computers and Intractability: a Guide to the Theory of NP-Completeness, *Freeman*, San Francisco, 1979.

[GRAHAM 69] R. L. GRAHAM, Bounds on Multiprocessing Timing Anomalies, *SIAM J. Appl. Math.*, 1969, *17*, p. 416-429.

[HERROELEN 90] E. DEMEULEMEESTER and W. HERROELEN, A Decision Support System for Resource-Constrained Project Scheduling, *International Workshop on Project Management and Scheduling*, Université de Technologie de Compiègne, France, juin 1990.

[MOCCELLIN 88] J. MOCCELLIN, On Directions in Resource-Constrained Project Scheduling, *International Workshop on Project Management and Scheduling*, Lisbon, juillet 1988.

[PATTERSON 76] J. H. PATTERSON and G. ROTH, Scheduling a Project Under Multiple Resource Constraints: a Zero-One Programming Approach, *Management Sc.*, 1976, *16*, p. 93-108.

[PINSON 88] E. PINSON, Le problème de job-shop, *Thèse de l'Université de Paris-VI*, 1988.

[SEDGEWICK 84] R. SEDGEWICK, Algorithms, *Addison-Wesley*, 1984.