stichting
mathematisch
centrum

MC

AFDELING MATHEMATISCHE BESLISKUNDE BW 126/80 AUGUSTUS
(DEPARTEMENT DE RECHERCHE OPERATIONNELLE)

J.K. LENSTRA, A.H.G. RINNOOY KAN

INTRODUCTION A L'ORDONNANCEMENT DE PLUSIEURS MACHINES

Prépublication

kruislaan 413 1098 SJ amsterdam

Printed at the Mathematical Centre, 413 Kruislaan, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

1980 Mathematics Subject Classification: 90B35, 68C25

INTRODUCTION A L'ORDONNANCEMENT DE PLUSIEURS MACHINES

J.K. LENSTRA

Mathematisch Centrum, Amsterdam

A.H.G. RINNOOY KAN

Université Erasmus, Rotterdam

RESUME

Cet article présente une synthèse des résultats récents dans le domaine de l'ordonnancement de plusieurs machines. La théorie de la complexité de calcul fournit le cadre dans lequel sont présentés ces résultats. Ils concernent d'une part le développement de nouveaux algorithmes polynomiaux, et d'autre part l'application du concept de NP-dureté ainsi que l'analyse des algorithmes d'approximation.

NOTIONS CLEFS: *machines parallèles, tâches, contraintes de précédence, préemption, temps d'achèvement maximum, temps d'achèvement total, complexité de calcul, algorithme polynomial, NP-dureté, optimisation, approximation.*

NOTE. Cet article sera inséré dans le compte rendu du colloque "Regards sur la théorie des graphes", Cerisy-la-Salle, 12-19 juin 1980.

1. INTRODUCTION

Au cours des dernières années, la théorie de l'ordonnancement de plusieurs machines a connu un développement rapide. Ceci est dû en partie au succès spectaculaire de la théorie de la complexité de calcul. L'application de cette théorie a permis de tracer une frontière précise entre deux classes de problèmes d'ordonnancement: les problèmes *bien résolus*, pour lesquels il existe des algorithmes polynomiaux en temps, et les problèmes *NP-durs*, qui sont probablement impossibles à traiter en ce sens qu'il est improbable qu'il existe des algorithmes polynomiaux pour les résoudre. La première classe s'est étendue continuellement grâce au développement de nouveaux algorithmes d'optimisation polynomiaux. En même temps, de nombreux algorithmes d'approximation pour les problèmes de la deuxième classe ont été analysés.

L'organisation de l'article est la suivante. La section 2 donne une courte introduction à la théorie de la complexité de calcul des problèmes combinatoires; un traitement plus détaillé se trouve dans [Karp 1972,1975; Garey & Johnson 1979; Lenstra & Rinnooy Kan 1979]. Les trois sections suivantes présentent une brève revue des résultats disponibles pour les problèmes d'ordonnancement de plusieurs machines. La section 3 concerne un certain nombre de modèles de base pour l'ordonnancement de tâches sur des machines parallèles. On considère à la section 4 le cas particulier de temps d'exécution unitaires et l'influence de contraintes de précédence entre les tâches. La section 5 est dévolue au cas où la préemption (interruption des tâches) est permise et où l'on peut spécifier des dates variables de disponibilité de tâche. La section 6 contient quelques remarques en guise de conclusion.

2. COMPLEXITE DES PROBLEMES COMBINATOIRES

La complexité de calcul d'un problème combinatoire doit évidemment être reliée au comportement des algorithmes élaborés pour le résoudre. Ce comportement est d'habitude mesuré par le *temps de calcul* de l'algorithme (c'est-à-dire le nombre d'opérations élémentaires telles que les additions

et les comparaisons) par rapport à la *taille* du problème (c'est-à-dire du nombre de bits occupés par les données).

Si un problème de taille n peut être résolu par un algorithme avec un temps de calcul $O(p(n))^*$ où p est une fonction *polynomiale*, alors l'algorithme peut être appelé *bon* et le problème *bien résolu*. Ces notions furent introduites par Edmonds [Edmonds 1965] dans le contexte du problème du couplage; son algorithme peut être implémenté de façon à prendre un temps $O(n^3)$ sur des graphes à n sommets. Des algorithmes polynomiaux ont été développés pour une grande variété de problèmes combinatoires [Lawler 1976]. D'autre part, de nombreux problèmes semblables peuvent seulement être résolus à l'aide de méthodes énumératives qui peuvent exiger un temps *exponentiel*.

Confronté à un problème combinatoire, on désirerait naturellement savoir s'il existe un algorithme polynomial ou si, au contraire, toute méthode de résolution doit prendre un temps exponentiel dans le pire des cas. Les résultats de ce dernier type sont encore rares, mais il est souvent possible de montrer que l'existence d'un algorithme polynomial est, à tout le moins, très improbable. On peut arriver à un tel résultat en prouvant que le problème en question est *NP-complet* [Cook 1971; Karp 1972]. Selon la définition formelle donnée ci-dessous, les problèmes NP-complets sont équivalents en ce sens qu'aucun d'entre eux n'est bien résolu et que, si l'un d'entre eux était bien résolu, il en serait de même pour tous. Comme tous les problèmes classiques qui sont notoires pour leur intraitabilité de calcul, tels que les problèmes du voyageur de commerce, de l'ordonnancement général sur machines et de la programmation entière, sont connus pour être NP-complets, la résolution d'un tel problème en temps polynomial serait effectivement très surprenante. Pour les besoins de la pratique, ceci implique qu'en résolvant de tels problèmes on peut tout aussi bien accepter l'inévitabilité d'un mauvais algorithme d'*optimisation* (superpolynomial) ou recourir à l'usage d'un bon algorithme d'*approximation* (polynomial).

La théorie de la NP-complétude concerne en premier lieu les *problèmes de reconnaissance*, qui exigent une réponse par oui ou par non. Un exemple

* La notation $q(n) = O(p(n))$ signifie qu'il existe une constante $c \geq 0$ telle que $|q(n)| \leq c \cdot p(n)$ pour tout $n > 0$.

de tel problème de reconnaissance est le suivant:

PARTITION:

occurrence: entiers positifs $a_1,\ldots,a_t,b$ tels que $\sum_{j=1}^{t} a_j = 2b$;

question: existe-t-il un sous-ensemble $S \subset \{1,2,\ldots,t\}$ tel que $\sum_{j\in S} a_j = b$?

PARTITION peut être résolu par énumération complète en temps $O(2^{t-1})$ ou par programmation dynamique en temps O(tb) [Bellman & Dreyfus 1962], mais ces temps d'exécution sont tous deux exponentiels en la taille du problème, qui est O(t log b).

Une occurrence d'un problème de reconnaissance est *admissible* si la question peut être répondue affirmativement. L'admissibilité est d'habitude équivalente à l'existence d'une *structure* associée qui satisfait une certaine propriété.

Un problème de reconnaissance appartient à la classe *P* si, pour toute occurrence de ce problème, son admissibilité ou sa non-admissibilité peut être déterminée par un algorithme polynomial. Il appartient à la classe *NP* si, pour toute occurrence, on peut déterminer en temps polynomial si une structure donnée affirme son admissibilité. Par exemple, PARTITION est un membre de *NP*, car pour tout $S \subset \{1,\ldots,t\}$ on peut tester si $\sum_{j\in S} a_j = b$ en temps O(t). Il est évident que $P \subset NP$.

Un problème P' est dit *réductible* au problème P (notation $P' \propto P$) si pour toute occurrence de P' on peut construire en temps polynomial une occurrence de P telle que la résolution de l'occurrence de P résoudra l'occurrence de P' également. Informellement, la réductibilité de P' à P implique que P' peut être considéré comme un cas particulier de P, de sorte que P est au moins aussi difficile que P'.

P est appelé *NP-dur* si $P' \propto P$ pour tout $P' \in NP$. Dans ce cas, P est au moins aussi difficile que tout problème de *NP*. P est appelé *NP-complet* si P est NP-dur et $P \in NP$. Donc les problèmes NP-complets sont les plus difficiles des problèmes de *NP*.

Un algorithme polynomial pour un problème NP-complet P pourrait être utilisé pour résoudre tous les problèmes de *NP* en temps polynomial, puisque pour toute occurrence d'un tel problème la construction de l'occurrence correspondante de P et sa résolution peuvent être effectuées toutes les deux en temps polynomial. Nous notons les deux conséquences importantes suivantes.

(i) Il est très improbable que $P = NP$, car NP contient de nombreux problèmes combinatoires bien connus, pour lesquels en dépit d'un considérable effort de recherche aucun algorithme polynomial n'a été découvert jusqu'ici.
(ii) Il est très improbable que $P \in P$ pour n'importe quel problème NP-complet P, car ceci impliquerait que $P = NP$ d'après l'argument précédent.

La premier résultat sur la NP-complétude est dû à Cook [Cook 1971]. Il a élaboré une "réduction maîtresse" pour prouver que tout problème de NP est réductible au problème appelé SATISFIABILITE. Sur base de ce résultat, Karp [Karp 1972] et beaucoup d'autres (voir, par exemple, [Karp 1975; Garey & Johnson 1979; Lenstra & Rinnooy Kan 1979]) ont identifié un grand nombre de problèmes NP-complets de la manière suivante. On peut établir la NP-complétude d'un problème $P \in NP$ en spécifiant une réduction $P' \propto P$ où P' est déjà connu pour être NP-complet: pour tout $P'' \in NP$, $P'' \propto P'$ et $P' \propto P$ impliquent alors que $P'' \propto P$ également. De cette façon, il a été prouvé que PARTITION est NP-complet [Karp 1972].

En ce qui concerne les *problemes d'optimisation,* on reformule usuellement, disons, un problème de minimisation en un problème de reconnaissance en demandant l'existence d'une solution admissible dont la valeur est au plus égale à un *seuil* donné. Lorsqu'il peut être prouvé que ce problème de reconnaissance est *NP-complet,* le problème d'optimisation correspondant est *NP-dur* en ce sens que l'existence d'un algorithme polynomial pour le résoudre impliquerait que $P = NP$.

3. QUELQUES MODELES DE BASE

Supposons que n *tâches* J_j $(j = 1,2,\ldots,n)$ doivent être exécutées sur m *machines* ou *processeurs parallèles* M_i $(i = 1,2,\ldots,m)$. Chaque machine peut exécuter à la fois une tâche au plus; chaque tâche peut être exécutée sur n'importe laquelle des machines. Les types de problèmes étudiés dans cet article seront caractérisés par une classification à trois champs $\alpha|\beta|\gamma$ [Graham *et al.* 1979].

Le premier champ $\alpha = \alpha_1\alpha_2$ spécifie l'*environnement en machines.* Notons p_{ij} le temps requis pour exécuter J_j sur M_i. On considérera trois valeurs possibles de α_1:

- P (*machines identiques*): $p_{ij} = p_j$, c'est-à-dire que le temps d'exécution de J_j sur M_i est égal au temps d'exécution requis p_j de J_j, pour tout M_i;
- Q (*machines uniformes*): $p_{ij} = p_j/s_i$, c'est-à-dire que le temps d'exécution de J_j sur M_i est égal au temps d'exécution requis p_j de J_j divisé par la vitesse s_i de M_i;
- R (*machines sans relations*): p_{ij} est arbitraire.

Si α_2 est un entier positif, m est *constant* et égal à α_2; si α_2 est vide, m est *variable*.

Le second champ β indique certaines *caractéristiques des tâches*. Dans cette section, β sera vide, ce qui implique ce qui suit:

- tous les p_{ij} (ou p_j) sont des entiers non-négatifs arbitraires;
- on ne spécifie pas de contraintes de précédence entre les tâches;
- la préemption (interruption de tâches) n'est pas admise;
- toutes les tâches sont disponibles pour l'exécution au temps 0.

La notation nécessaire à indiquer lesquelles de ces hypothèses ne sont pas vérifiées sera définie dans les sections suivantes.

Le troisième champ γ correspond au *critère d'optimalité* choisi. Tout ordonnancement admissible définit un *temps d'achèvement* C_j de J_j $(j = 1,\ldots,n)$. Nous considérerons la minimisation de deux critères:

- le *temps d'achèvement maximum* $C_{max} = \max\{C_1,\ldots,C_n\}$;
- le *temps d'achèvement total* $\sum C_j = C_1+\ldots+C_n$.

La valeur optimale de γ sera notée γ^*, la valeur obtenue par un algorithme (d'approximation) A par $\gamma(A)$.

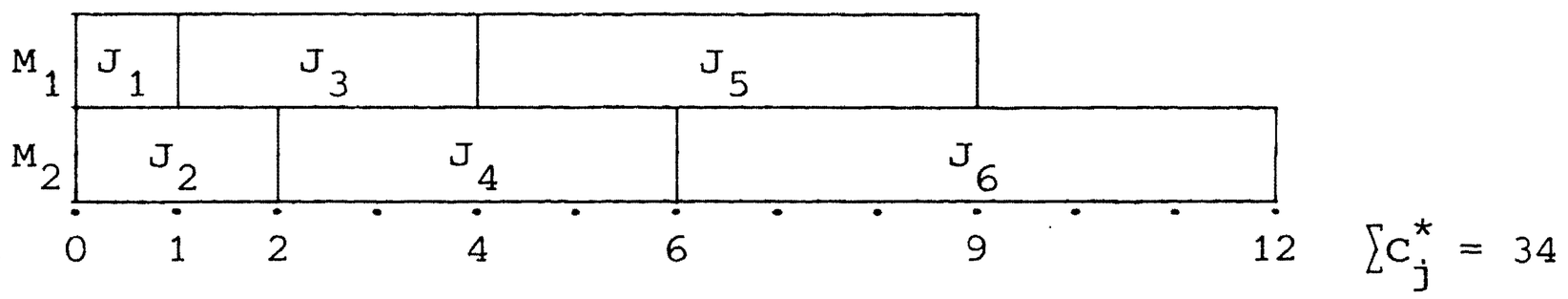
Les exemples 1, 2 et 3 illustrent cette classification de problèmes. On utilise des *diagrammes de Gantt* pour représenter les ordonnancements de manière évidente.

Exemple 1. $P2||\sum C_j$

problème: minimiser le temps d'achèvement total sur deux machines identiques.

occurrence: $n = 6$; $p_j = j$ $(j = 1,\ldots,6)$.

ordonnancement optimal:



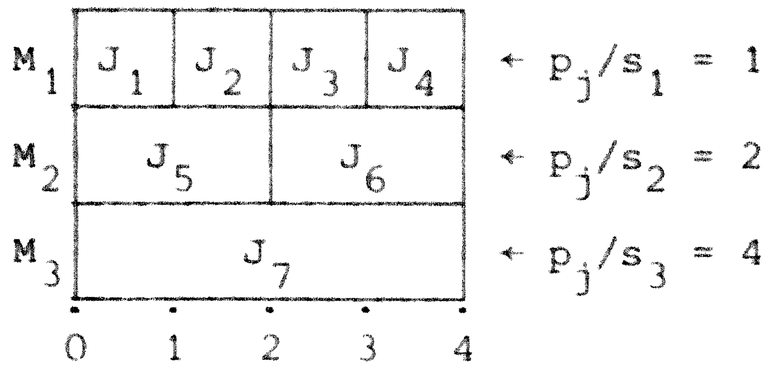
$\sum C_j^* = 34$

Exemple 2. $Q3||C_{max}$

problème: minimiser le temps d'achèvement maximum sur trois machines uniformes.

occurrence: $s_1 = 4$, $s_2 = 2$, $s_3 = 1$; $n = 7$; $p_j = 4$ $(j = 1,\ldots,7)$.

ordonnancement optimal:



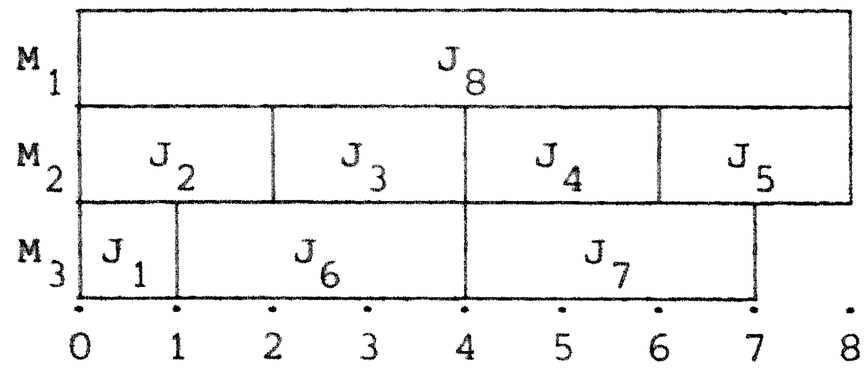
$C^*_{max} = 4$

Exemple 3. $R||C_{max}$

problème: minimiser le temps d'achèvement total sur m machines sans relations.

occurrence: $m = 3$; $n = 8$; $p_{11} = 1$, $p_{1j} = 1$ $(j = 2,\ldots,7)$, $p_{18} = 8$,
$p_{21} = 1$, $p_{2j} = 2$ $(j = 2,\ldots,7)$, $p_{28} = 9$,
$p_{31} = 1$, $p_{3j} = 3$ $(j = 2,\ldots,7)$, $p_{38} = 9$.

ordonnancement optimal:



$C^*_{max} = 8$

Présentons une synthèse des résultats disponibles pour ces modèles de base. Il se trouve que les problèmes $\sum C_j$ sont relativement aisés, tandis que les problèmes C_{max} sont très difficiles.

La règle du *plus court temps d'exécution* ("shortest processing time": SPT) résoud $P||\sum C_j$ en temps O(n log n) de la manière suivante [Conway *et al.* 1967]. Supposons que $n = \ell m$ (des tâches fictives avec temps d'exécution nul sont ajoutées si ce n'est pas le cas), réindiçons les tâches de sorte que $p_1 \leq \ldots \leq p_n$, et ordonnançons les m tâches $J_{(k-1)m+1}, J_{(k-1)m+2}, \ldots, J_{km}$ en $k^{ème}$ position sur les m machines $(k = 1,2,\ldots,\ell)$. L'exemple 1 illustre cette règle. La preuve d'optimalité est directe: dans la valeur $\sum C_j$ du critère, le temps d'exécution d'une tâche en $k^{ème}$ position sur une machine est compté $\ell+1-k$ fois, et, par conséquent, $\sum C_j$ est égal au produit scalaire

de deux n-vecteurs $(\ell,\ldots,\ell,\ell-1,\ldots,\ell-1,\ldots,1,\ldots,1)$ et $(p_1,\ldots,p_n)$; comme les multiplicateurs du premier de ces vecteurs sont non-croissants, $\sum C_j$ est minimum si les temps d'exécution dans le second vecteur sont non-décroissants.

Cet algorithme a été généralisé pour résoudre $Q||\sum C_j$ en temps $O(n \log n)$ également [Conway *et al.* 1967; Horowitz & Sahni 1976].

Le cas le plus général $R||\sum C_j$ peut être formulé et résolu comme un *problème linéaire de transport* m×n, en temps $O(n^3)$ [Horn 1973; Bruno *et al.* 1974]. Soit

$$x_{ijk} = \begin{cases} 1 & \text{si } J_j \text{ est dans la } k^{\text{ème}} \text{ dernière position sur } M_i; \\ 0 & \text{sinon.} \end{cases}$$

Le problème est alors de minimiser

$$\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{n} kp_{ij}x_{ijk}$$

sous les contraintes

$$\sum_{i=1}^{m}\sum_{k=1}^{n} x_{ijk} = 1 \quad (j = 1,\ldots,n),$$
$$\sum_{j=1}^{n} x_{ijk} \leq 1 \quad (i = 1,\ldots,m;\ k = 1,\ldots,n),$$
$$x_{ijk} \geq 0 \quad (i = 1,\ldots,m;\ j = 1,\ldots,n;\ k = 1,\ldots,n).$$

Donc la minimisation de $\sum C_j$ requiert un temps polynomial, même sur m machines sans relations. Par contre, la minimisation de C_{max} est NP-dur, même sur deux machines identiques.

La preuve de NP-dureté pour $P2||C_{max}$ est triviale. Etant donné une occurrence quelconque de PARTITION, définie par des entiers positifs $a_1,\ldots,a_t,b$ (voir section 2), nous construisons une occurrence de $P2||C_{max}$ en définissant $n = t$ et $p_j = a_j$ $(j = 1,\ldots,n)$. Il est clair qu'il existe un sous-ensemble $S \subset \{1,\ldots,t\}$ avec $\sum_{j\in S} a_j = b$ si et seulement s'il existe un ordonnancement avec $C_{max} \leq b$. Il s'ensuit que PARTITION est réductible à $P2||C_{max}$, et comme PARTITION est NP-complet [Karp 1972], $P2||C_{max}$ est NP-dur. Ceci implique que toutes les généralisations de $P2||C_{max}$, telles que $P3||C_{max},\ldots,P||C_{max},Q2||C_{max},\ldots,R||C_{max}$, sont également NP-durs.

En conséquence, il semble inévitable que les algorithmes d'optimisa-

tion pour ces problèmes soient de nature énumérative. Un schéma général de *programmation dynamique* [Rothkopf 1966; Lawler & Moore 1969] peut être largement appliqué. Pour $P||C_{max}$, ce schéma est le suivant. Soit

$$B_j(t_1,\ldots,t_m) = \begin{cases} \textit{vrai} & \text{si } J_1,\ldots,J_j \text{ peuvent être ordonnancées} \\ & \text{sur } M_1,\ldots,M_m \text{ de sorte que } M_i \text{ est} \\ & \text{utilisée de 0 à } t_i \ (i = 1,\ldots,m), \\ \textit{faux} & \text{sinon,} \end{cases}$$

avec

$$B_0(t_1,\ldots,t_m) = \begin{cases} \textit{vrai} & \text{si } t_i = 0 \ (i = 1,\ldots,m), \\ \textit{faux} & \text{sinon.} \end{cases}$$

L'équation de récurrence est alors

$$B_j(t_1,\ldots,t_m) = \vee_{i=1}^{m} B_{j-1}(t_1,\ldots,t_{i-1},t_i-p_j,t_{i+1},\ldots,t_m).$$

Soit C une borne supérieure de la valeur optimale C^*_{max}. Pour $j = 0,1,\ldots,n$, calculer $B_j(t_1,\ldots,t_m)$ pour $t_i = 0,1,\ldots,C$ $(i = 1,\ldots,m)$, et déterminer

$$C^*_{max} = \min\{\max\{t_1,\ldots,t_m\} \mid B_n(t_1,\ldots,t_m) = \textit{vrai}\}.$$

Cette procédure résoud $P||C_{max}$ en temps $O(nC^m)$. Pour de grandes valeurs de C une *procédure de séparation* ("branch-and-bound") pourrait être préférable. Toutes ces algorithmes d'optimisation requièrent toutefois des temps de calcul prohibitifs dans le pire des cas.

Comme on l'a discuté plus haut, la NP-dureté de $P||C_{max}$ justifie également l'emploi d'algorithmes d'approximation rapides. Il est devenu courant de soumettre un tel algorithme à une *analyse du pire des cas* de façon à obtenir une garantie sur sa performance relative. Un des premiers résultats de ce type concerne la solution de $P||C_{max}$ par *ordonnancement par liste* ("list scheduling": LS), dans laquelle une liste de priorités des tâches est donnée et à chaque étape la première machine disponible est sélectionnée pour exécuter la première tâche disponible de la liste [Graham 1966]:

$$C_{max}(LS)/C^*_{max} \leq 2 - \frac{1}{m}.$$

Pour la règle du *temps d'exécution le plus long* ("longest processing time": LPT), où la liste contient les tâches dans l'ordre des p_j non-croissants, la borne s'améliore considérablement [Graham 1969]:

$$C_{max}(LPT)/C^*_{max} \leq \frac{4}{3} - \frac{1}{3m}.$$

Les exemples 4 et 5 démontrent que ces bornes sont les meilleures possibles.

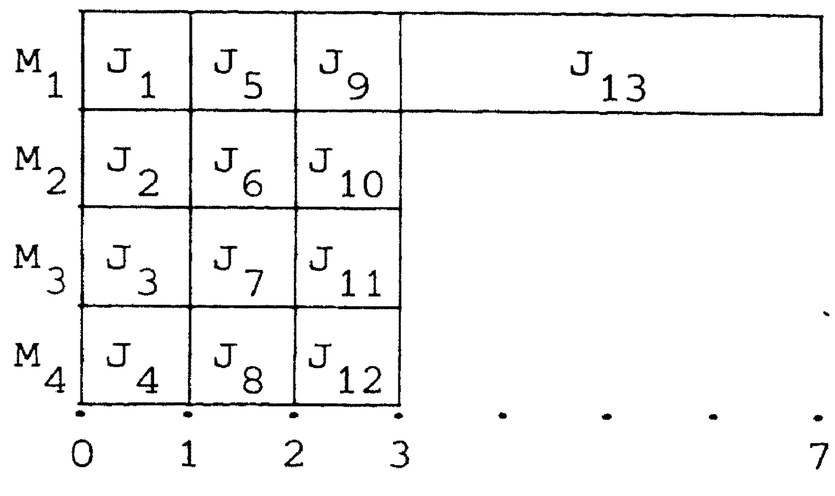
Exemple 4. $P||C_{max}(LS)$

la pire des occurrences:

$n = (m-1)m+1;$

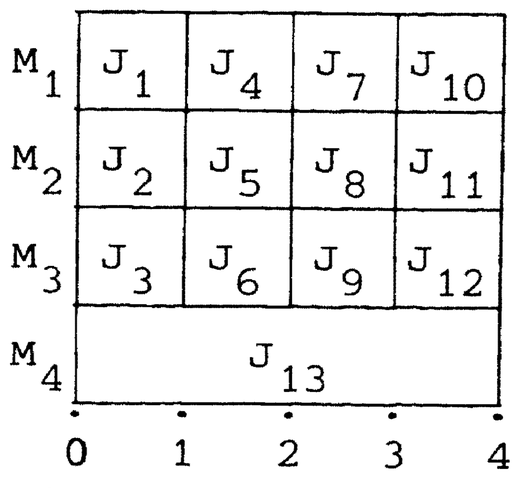
$(p_1,\ldots,p_n) = (1,\ldots,1,m).$

ordonnancement approché:



$C_{max}(LS) = 2m-1$

ordonnancement optimal:



$C^*_{max} = m$

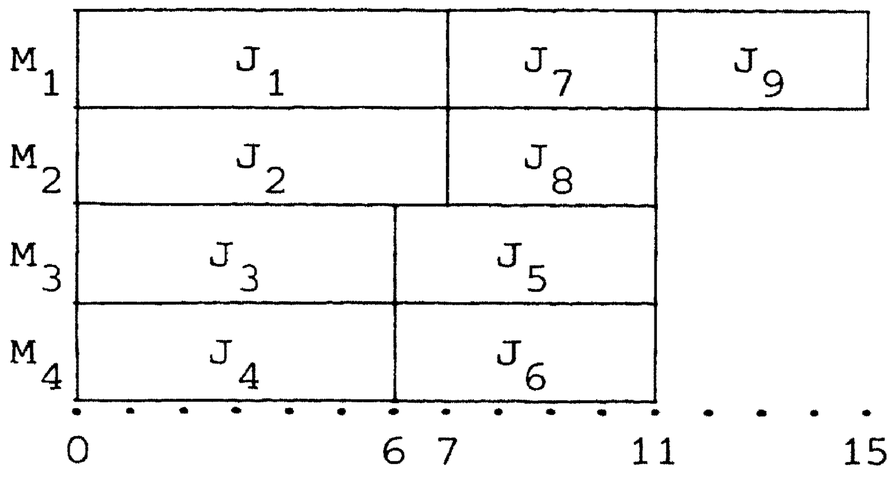
Exemple 5. $P||C_{max}(LPT)$

la pire des occurrences:

$n = 2m+1;$

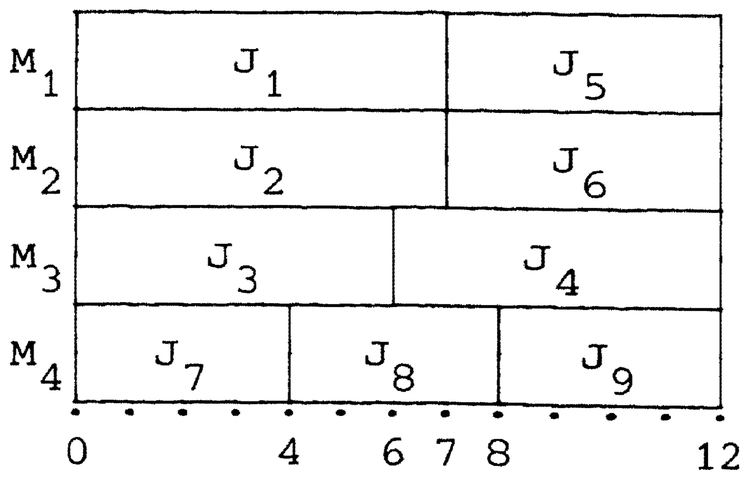
$(p_1,\ldots,p_n) = (2m-1,2m-1,2m-2,2m-2,\ldots,m+1,m+1,m,m,m).$

ordonnancement approché:



$C_{max}(LPT) = 4m-1$

ordonnancement optimal:



$C^*_{max} = 3m$

4. TEMPS D'EXECUTION UNITAIRES ET INFLUENCE DES CONTRAINTES DE PRECEDENCE

Les résultats de la section 3 suggèrent que des hypothèses simplificatrices sont nécessaires pour résoudre optimalement $P||C_{max}$ en temps polynomial. Dans cette section, nous supposons que toutes les tâches ont des *temps d'exécution unitaires*, ce qui sera indiqué dans le second champ de notre classification par $p_j=1$. Cette hypothèse nous permet également d'étudier l'influence des *contraintes de précédence* entre les tâches. Il apparaît qu'il est utile de distinguer entre deux types de contraintes de précédence:

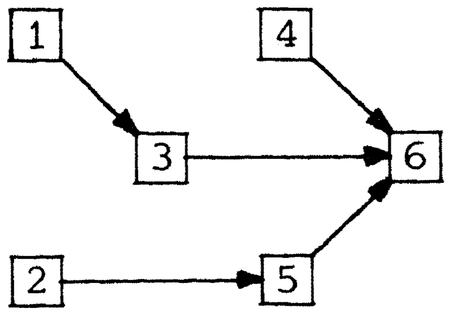
- *prec* (contraintes de précédence arbitraires): un graphe orienté G sans circuits, avec des sommets 1,...,n, est donné; si G contient un chemin de j à k, on écrit $J_j \rightarrow J_k$ et on exige que J_j soit achevée avant que J_k ne débute;
- *arbre* (contraintes de précédence en forme d'arbre): G est un arbre orienté avec une anti-racine et des demi-degrés extérieurs au plus égaux à 1 en tout sommet.

Les exemples 6 et 7 ci-dessous illustrent ces concepts.

Un des plus anciens résultats dans cette catégorie de problèmes est la solution de $P|arbre,p_j=1|C_{max}$ en temps O(n) [Hu 1961]. L'algorithme de Hu utilise l'*ordonnancement par chemin critique*: on définit le *niveau* ℓ_j de J_j comme le nombre de sommets sur le chemin unique de j à l'anti-racine de l'arbre et on applique l'ordonnancement par liste à une liste qui contient les tâches dans l'ordre des ℓ_j non-croissants. L'exemple 6 illustre cet algorithme.

Exemple 6. $P|arbre,p_j=1|C_{max}$

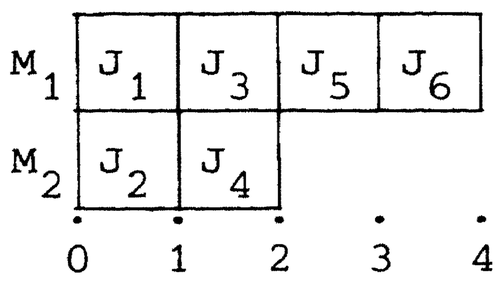
occurrence: m = 2; n = 6; G:



⇒

j	1	2	3	4	5	6
ℓ_j	3	3	2	2	2	1

ordonnancement optimal:

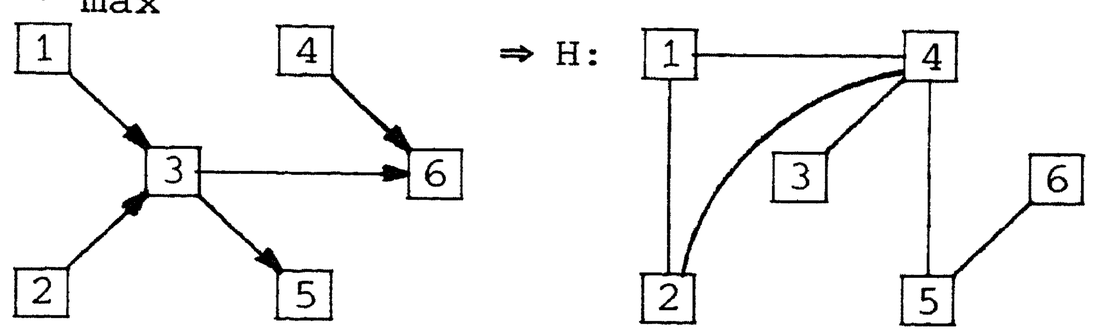


$C^*_{max} = 4$

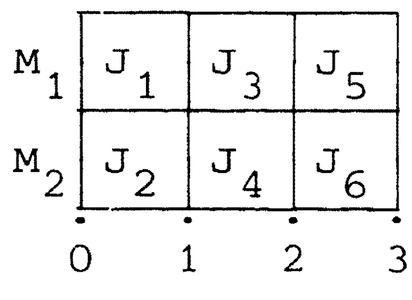
Le second résultat de base est la solution de $P2|prec,p_j=1|C_{max}$ en temps polynomial. Un algorithme $O(n^3)$ [Fujii *et al.* 1969,1971] fonctionne comme suit: on construit un graphe non-orienté H avec des sommets 1,...,n et des arêtes {j,k} si et seulement si ni $J_j \to J_k$ ni $J_k \to J_j$ et on obtient un ordonnancement optimal à partir d'un *couplage* (c'est-à-dire un ensemble d'arêtes sans sommets communs) de cardinalité maximum de H. L'exemple 7 illustre cet algorithme. Notons que le problème peut encore être résolu en temps $O(n^3)$ si, de plus, chaque tâche est contrainte à être exécutée entre sa *date de disponibilité* et sa *date de livraison* [Garey & Johnson 1977].

Exemple 7. $P2|prec,p_j=1|C_{max}$

occurrence: n = 6; G: ⇒ H:



ordonnancement optimal:



$C^*_{max} = 3$

Pour toute constante $m \geq 3$, la complexité de $Pm|prec,p_j=1|C_{max}$ est une question ouverte. Toutefois, il est connu que $P|prec,p_j=1|C_{max}$ est NP-dur [Ullman 1975; Lenstra & Rinnooy Kan 1978]. La dernière de ces preuves implique qu'aucun algorithme d'approximation polynomial pour $P|prec,p_j=1|C_{max}$ ne peut avoir de borne dans le pire des cas meilleure que 4/3, sauf si $P = NP$. Pour l'*ordonnancement par chemin critique* ("critical path scheduling": CP) il a été montré [Chen 1975; Chen & Liu 1975] que

$$C^*_{max}(CP)/C^*_{max} \leq \begin{cases} \frac{4}{3} & \text{pour } m = 2, \\ 2 - \frac{1}{m-1} & \text{pour } m \geq 3, \end{cases}$$

et ces bornes sont atteintes.

5. PREEMPTION ET INFLUENCE DES DATES DE DISPONIBILITE

Nous considérons maintenant une seconde modification des modèles d'ordonnancement de plusieurs machines qui conduira à plusieurs algorithmes d'optimisation polynomiaux. Plus spécifiquement, nous faisons l'hypothèse que la *préemption* est admise sans limites: l'exécution de n'importe quelle tâche peut être interrompue arbitrairement souvent et reprise en même temps sur une machine différente ou plus tard sur n'importe quelle machine. Ceci sera indiqué dans le second champ de notre classification de problèmes par *pmtn*.

On a montré que pour $P|pmtn|\sum C_j$ la préemption ne présente aucun avantage [McNaughton 1959]. Donc, la règle non-préemptive SPT de la section 3 peut être appliquée pour résoudre le problème en temps O(n log n).

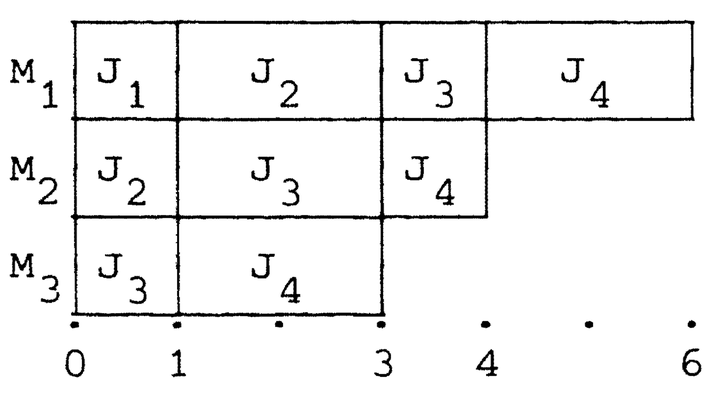
Une version préemptive de la règle SPT résoud $Q|pmtn|\sum C_j$ en temps O(n log n + mn) [Gonzalez 1977]: on place les tâches dans l'ordre SPT, et on ordonnance chaque tâche successive préemptivement de façon à minimiser son temps d'achèvement. L'ordonnancement résultant contient au plus $(m-1)(n-\frac{m}{2})$ préemptions. L'exemple 8 illustre cette règle.

Très peu de choses sont connues sur $R|pmtn|\sum C_j$. Ceci est un des problèmes ouverts les plus intrigants dans le domaine de l'ordonnancement de plusieurs machines.

Exemple 8. $Q|pmtn|\sum C_j$

occurrence: $m = 3$; $s_1 = 3$, $s_2 = 2$, $s_3 = 1$; $n = 4$; $p_1 = 3$, $p_2 = p_3 = 8$, $p_4 = 10$.

ordonnancement optimal:



$\sum C_j^* = 14$

$P|pmtn|C_{max}$ et $Q|pmtn|C_{max}$ se distinguent particulièrement car dans les deux cas il existe une expression analytique simple qui est une borne inférieure évidente sur C_{max}^* tandis qu'un ordonnancement atteignant cette borne peut être obtenu en temps polynomial. Pour $P|pmtn|C_{max}$, nous avons

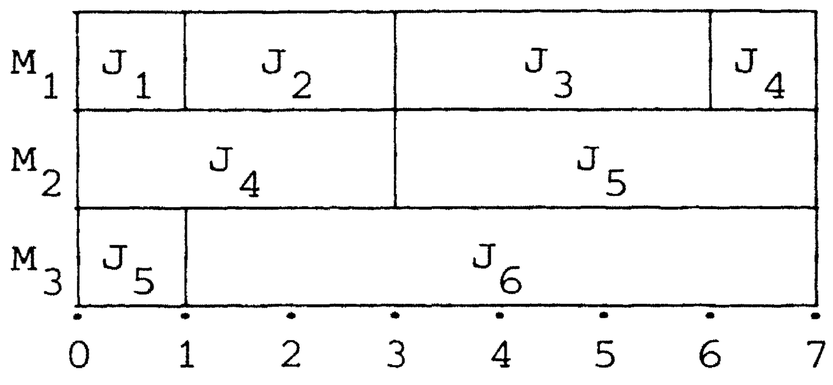
$$C_{max}^* = \max\{p_1,\ldots,p_n,\frac{p_1+\ldots+p_n}{m}\}.$$

La *règle d'enroulement* résoud ce problème en temps O(n) [McNaughton 1959]: remplir les machines successivement en ordonnançant les tâches dans n'importe quel ordre et en interrompant une tâche chaque fois que la borne donnée ci-dessus est atteinte. Il y aura au plus m-1 préemptions. L'exemple 9 illustre cette règle.

Exemple 9. $P|pmtn|C_{max}$

occurrence: $m = 3$; $n = 6$; $p_j = j$ $(j = 1,\ldots,6) \Rightarrow C^*_{max} = \max\{6, \frac{21}{3}\} = 7$.

ordonnancement optimal:



Pour $Q|pmtn|C_{max}$, nous avons

$$C^*_{max} = \max\{\frac{p_1}{s_1}, \frac{p_1+p_2}{s_1+s_2}, \ldots, \frac{p_1+\ldots+p_{m-1}}{s_1+\ldots+s_{m-1}}, \frac{p_1+\ldots+p_n}{s_1+\ldots+s_m}\},$$

où $s_1 \geq \ldots \geq s_m$ et $p_1 \geq \ldots \geq p_n$. Si les machines et les tâches sont rangées de cette façon, un algorithme compliqué résoud le problème en temps O(n) [Gonzalez & Sahni 1978]. Il engendre au plus 2(m-1) préemptions.

$R|pmtn|C_{max}$ peut être formulé comme un problème de *programmation linéaire* [Lawler & Labetoulle 1978]. Soit

x_{ij} = temps passé par J_j sur M_i.

Le problème est alors de minimiser

$$C_{max}$$

sous les contraintes

$$\sum_{i=1}^{m} x_{ij}/p_{ij} = 1 \quad (j = 1,\ldots,n),$$
$$\sum_{i=1}^{m} x_{ij} \leq C_{max} \quad (j = 1,\ldots,n),$$
$$\sum_{j=1}^{n} x_{ij} \leq C_{max} \quad (i = 1,\ldots,m),$$
$$x_{ij} \geq 0 \quad (i = 1,\ldots,m;\ j = 1,\ldots,n).$$

Khachian a montré que les programmes linéaires peuvent être résolus en temps polynomial [Khachian 1979]. Etant donné une solution (x^*_{ij}), un ordonnancement admissible peut également être construit en temps polynomial [Gonzalez & Sahni 1976]. Il n'y aura pas plus d'environ $\frac{7}{2}m^2$ préemptions.

Nous pouvons étendre les modèles d'ordonnancement avec préemption en faisant l'hypothèse que J_j devient disponible pour l'exécution à une *date de disponibilité* r_j (j = 1,...,n) entière et donnée. Ceci sera indiquée dans le second champ de la classification par r_j. Les modèles résultants sont loin d'être triviaux et nous nous limitons à mentionner les résultats les plus importants.

Lorsqu'on ordonnance avec des dates de disponibilité, on peut distinguer trois types d'algorithmes. Un algorithme est *connecté* ("on-line") si à n'importe quel moment seule l'information sur les tâches disponibles est requise. Il est *presque connecté* ("nearly on-line") si de plus la date de disponibilité suivante doit être connue. Il est *déconnecté* ("off-line") si toute l'information est disponible à l'avance.

$P|pmtn,r_j|\sum C_j$ et $Q|pmtn,r_j|\sum C_j$ sont très largement ouverts. Tout ce que nous connaissons à propos de ces deux problèmes c'est qu'il n'existe pas d'algorithme connecté, même pour le cas de deux machines identiques [Labetoulle *et al.* 1979].

$P|pmtn,r_j|C_{max}$ peut être résolu par un algorithme connecté O(mn) [Horn 1974; Gonzalez & Johnson 1980], et $Q|pmtn,r_j|C_{max}$ par un algorithme presque connecté $O(n^2)$ [Sahni & Cho 1979; Labetoulle *et al.* 1979].

Finalement, nous faisons l'hypothèse que, de plus, J_j doit être achevée pas plus tard qu'une *date de livraison* d_j (j = 1,...,n), et nous remplaçons l'objectif de minimisation de C_{max} par un test d'existence d'un ordonnancement avec préemption admissible tant pour les dates de disponibilité que pour les dates de livraison. On a montré qu'il n'existe pas d'algorithme presque connecté, même pour le cas de deux machines identiques [Sahni 1979]. Toutefois des algorithmes déconnectés sont disponibles: $P|pmtn,r_j,d_j|-$ peut être résolu par un calcul de flot dans les graphes $O(n^3)$ [Horn 1974], et $Q|pmtn,r_j,d_j|-$ au moyen d'un modèle de flot dans les graphes "généralisé" $O(n^6)$ [Martel 1979].

6. REMARQUES EN GUISE DE CONCLUSION

Nous avons passé en revue quelques-uns parmi les nombreux résultats récents dans le domaine de l'ordonnancement de plusieurs machines. Plusieurs sujets n'ont pas été examinés; en particulier nous mentionnons l'extension du modèle pour inclure des *contraintes sur les ressources* additionnelles, pour lequel de nombreux résultats sont maintenant disponibles [Graham *et al.* 1979; Blazewicz *et al.* 1980]. Le développement d'algorithmes de sophistication croissante combiné avec une application approfondie des outils de la théorie de la complexité de calcul devrait continuer à rendre intéressant le domaine de l'ordonnancement de plusieurs machines tant pour les théoriciens que pour les praticiens.

REMERCIEMENTS

Les auteurs expriment leur reconnaissance à P. Hansen pour la traduction excellente de l'anglais et à Mme K.C.E. Lenstra pour son assistance dans la préparation de la version définitive. Cette recherche a été supportée partiellement par un crédit NATO Special Research Grant 9.2.02 (SRG.7).

LITTERATURE

R.E. BELLMAN, S.E. DREYFUS (1962) *Applied Dynamic Programming*, Princeton University Press, Princeton, N.J.

J. BLAZEWICZ, J.K. LENSTRA, A.H.G. RINNOOY KAN (1980) Scheduling subject to resource constraints: classification and complexity. Report BW 127, Mathematisch Centrum, Amsterdam.

J. BRUNO, E.G. COFFMAN, JR., R. SETHI (1974) Scheduling independent tasks to reduce mean finishing time. *Comm. ACM* 17,382-387.

N.-F. CHEN (1975) An analysis of scheduling algorithms in multiprocessing computing systems. Technical Report UIUCDCS-R-75-724, Department of Computer Science, University of Illinois at Urbana-Champaign.

N.-F. CHEN, C.L. LIU (1975) On a class of scheduling algorithms for multiprocessors computing systems. In: T.-Y. FENG (réd.) (1975)

Parallel Processing, Lecture Notes in Computer Science 24, Springer, Berlin, 1-16.

R.W. CONWAY, W.L. MAXWELL, L.W. MILLER (1967) *Theory of Scheduling*, Addison-Wesley, Reading, Mass.

S.A. COOK (1971) The complexity of theorem-proving procedures. *Proc. 3rd Annual ACM Symp. Theory of Computing*, 151-158.

J. EDMONDS (1965) Paths, trees, and flowers. *Canad. J. Math.* 17,449-467.

M. FUJII, T. KASAMI, K. NINOMIYA (1969,1971) Optimal sequencing of two equivalent processors. *SIAM J. Appl. Math.* 17,784-789; Erratum. 20,141.

M.R. GAREY, D.S. JOHNSON (1977) Two-processor scheduling with start-times and deadlines. *SIAM J. Comput.* 6,416-426.

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

T. GONZALEZ (1977) Optimal mean finish time preemptive schedules. Technical Report 220, Computer Science Department, Pennsylvania State University.

T. GONZALEZ, D.B. JOHNSON (1980) A new algorithm for preemptive scheduling of trees. *J. Assoc. Comput. Mach.* 27,287-312.

T. GONZALEZ, S. SAHNI (1976) Open shop scheduling to minimize finish time. *J. Assoc. Comput. Mach.* 23,665-679.

T. GONZALEZ, S. SAHNI (1978) Preemptive scheduling of uniform processor systems. *J. Assoc. Comput. Mach.* 25,92-101.

R.L. GRAHAM (1966) Bounds for certain multiprocessing anomalies. *Bell System Tech. J.* 45,1536-1581.

R.L. GRAHAM (1969) Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.* 17,263-269.

R.L. GRAHAM, E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.* 5,287-326.

W.A. HORN (1973) Minimizing average flow time with parallel machines. *Operations Res.* 21,846-847.

W.A. HORN (1974) Some simple scheduling algorithms. *Naval Res. Logist. Quart.* 21,177-185.

E. HOROWITZ, S. SAHNI (1976) Exact and approximate algorithms for scheduling nonidentical processors. *J. Assoc. Comput. Mach.* 23,317-327.

T.C. HU (1961) Parallel sequencing and assembly line problems. *Operations*

Res. 9,841-848.

R.M. KARP (1972) Reducibility among combinatorial problems. In: R.E. MILLER, J.W. THATCHER (réd.) (1972) *Complexity of Computer Computations*, Plenum Press, New York, 85-103.

R.M. KARP (1975) On the computational complexity of combinatorial problems. *Networks* 5,45-68.

L.G. KHACHIAN (1979) A polynomial algorithm in linear programming. *Soviet Math. Dokl.* 20,191-194.

J. LABETOULLE, E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN (1979) Preemptive scheduling of uniform machines subject to release dates. Report BW 99, Mathematisch Centrum, Amsterdam.

E.L. LAWLER (1976) *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York.

E.L. LAWLER, J. LABETOULLE (1978) On preemptive scheduling of unrelated parallel processors by linear programming. *J. Assoc. Comput. Mach.* 25,612-619.

E.L. LAWLER, J.M. MOORE (1969) A functional equation and its application to resource allocation and sequencing problems. *Management Sci.* 16,77-84.

J.K. LENSTRA, A.H.G. RINNOOY KAN (1978) Complexity of scheduling under precedence constraints. *Operations Res.* 26,22-35.

J.K. LENSTRA, A.H.G. RINNOOY KAN (1979) Computational complexity of discrete optimization problems. *Ann. Discrete Math.* 4,121-140.

C. MARTEL (1979) Generalized network flows with an application to multi-processor scheduling. Computer Science Division, University of California, Berkeley.

R. McNAUGHTON (1959) Scheduling with deadlines and loss functions. *Management Sci.* 6,1-12.

M.H. ROTHKOPF (1966) Scheduling independent tasks on parallel processors. *Management Sci.* 12,437-447.

S. SAHNI (1979) Preemptive scheduling with due dates. *Operations Res.* 27,925-934.

S. SAHNI, Y. CHO (1979) Nearly on line scheduling of a uniform processor system with release times. *SIAM J. Comput.* 8,275-285.

J.D. ULLMAN (1975) *NP*-complete scheduling problems. *J. Comput. System Sci.* 10,384-393.