Stat.

Netherlands school of economics
ECONOMETRIC INSTITUTE

AN INTRODUCTION TO MULTIPROCESSOR SCHEDULING

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

Erasmus
GIANNINI FOUNDATION OF AGRICULTURAL ECONOMICS LIBRARY
WITHDRAWN
JAN 21 1981

REPORT 8009/O

ERASMUS UNIVERSITY ROTTERDAM P.O. BOX 1738, 3000 DR. ROTTERDAM, THE NETHERLANDS.

AN INTRODUCTION TO MULTIPROCESSOR SCHEDULING

J.K. LENSTRA
Mathematisch Centrum, Amsterdam

A.H.G. RINNOOY KAN
Erasmus University, Rotterdam

ABSTRACT

This is a tutorial survey of recent results in the area of multiprocessor scheduling. Computational complexity theory provides the framework in which these results are presented. They involve on one hand the development of new polynomial optimization algorithms, and on the other hand the application of the concept of NP-hardness as well as the analysis of approximation algorithms.

KEY WORDS & PHRASES: *parallel machines, jobs, precedence constraints, preemption, maximum completion time, total completion time, computational complexity, polynomial algorithm, NP-hardness, optimization, approximation.*
NOTE: This report will appear in the proceedings of a conference.

1. INTRODUCTION

Throughout recent years, the theory of multiprocessor scheduling has been in rapid development. This is partly due to the spectacular success of computational complexity theory. Application of this theory has established a sharp borderline between two classes of scheduling problems: the *well-solved* problems, for which polynomial-time algorithms exist, and the *NP-hard* problems, which are probably intractable in the sense that the existence of polynomial algorithms is very unlikely. The former class has been continually expanded by the development of new polynomial optimization algorithms. At the same time, for problems in the latter class many approximation algorithms have been analyzed.

The outline of the paper is as follows. Section 2 gives a short introduction to the theory of the computational complexity of combinatorial problems; a more detailed treatment can be found in [Karp 1972, 1975; Garey & Johnson 1979; Lenstra & Rinnooy Kan 1979]. The next three sections provide a brief survey of the results available for multiprocessor scheduling problems. Section 3 deals with a number of basic models for scheduling jobs on parallel machines. Section 4 considers the special case of unit processing times and the influence of precedence constraints between the jobs. Section 5 is devoted to the case in which preemption (job splitting) is allowed and varying job release dates may be specified. Section 6 contains some concluding remarks.

2. COMPUTATIONAL COMPLEXITY OF COMBINATORIAL PROBLEMS

The inherent computational complexity of a combinatorial problem obviously has to be related to the computational behavior of algorithms designed for its solution. This behavior is usually measured by the *running time* of the algorithm (*i.e.*, the number of elementary operations such as additions and comparisons) as related to the *size* of the problem (*i.e.*, the number of bits occupied by the data).

If a problem of size n can be solved by an algorithm with running time $O(p(n))^*$ where p is a *polynomial* function, then the algorithm may be called *good* and the problem *well solved*. These notions were introduced by Edmonds [Edmonds 1965] in the context of the matching problem; his algorithm can be implemented to run in $O(n^3)$ time on graphs with n vertices. Polynomial algorithms have been developed for a wide variety of combinatorial optimization problems [Lawler 1976]. On the other hand, many such problems can only be solved by enumerative methods which may require *exponential* time.

When encountering a combinatorial problem, one would naturally like to know if a polynomial algorithm exists or if, on the contrary, any solution method must require exponential time in the worst case. Results of the latter type are still rare, but it is often possible to show that the existence of a polynomial algorithm is at the very least extremely unlikely. One may arrive at such a result by proving that the problem in question is *NP-complete* [Cook 1971; Karp 1972]. According to the formal definition given below, the NP-complete problems are equivalent in the sense that none of them has been well solved and that, if one of them would be well solved, then the same would be true for all of them. Since all the classical problems that are notorious for their computational intractability, such as traveling salesman, job shop scheduling and integer programming problems, are known to be NP-complete, the polynomial-time solution of such a problem would be very surprising indeed. For practical purposes, this implies that in solving those problems one may just as well accept the inevitability of a bad (superpolynomial) *optimization* algorithm or resort to using a good (polynomial) *approximation* algorithm.

* The notation "$q(n) = O(p(n))$" means that there exists a constant $c \geq 0$ such that $|q(n)| \leq c \cdot p(n)$ for all $n > 0$.

The theory of NP-completeness deals primarily with *recognition problems*, which require a yes/no answer. An example of a recognition problem is the following:

PARTITION:
instance: positive integers $a_1,\ldots,a_t,b$ with $\sum_{j=1}^{t} a_j = 2b$;
question: does there exist a subset $S \subset \{1,\ldots,t\}$ such that $\sum_{j\in S} a_j = b$?

PARTITION can be solved by complete enumeration in $O(2^{t-1})$ time or by dynamic programming in $O(tb)$ time [Bellman & Dreyfus 1962], but both running times are exponential in the problem size, which is $O(t \log b)$.

An instance of a recognition problem is *feasible* if the question can be answered affirmatively. Feasibility is usually equivalent to the existence of an associated *structure* which satisfies a certain property.

A recognition problem belongs to the class P if, for any instance of the problem, its feasibility or infeasibility can be determined by a polynomial algorithm. It belongs to the class NP if, for any instance, one can determine in polynomial time whether a given structure affirms its feasibility. For example, PARTITION is a member of NP, since for any $S \subset \{1,\ldots,t\}$ one can test whether $\sum_{j\in S} a_j = b$ in $O(t)$ time. It is obvious that $P \subseteq NP$.

Problem P' is said to be *reducible* to problem P (notation: $P' \propto P$) if for any instance of P' an instance of P can be constructed in polynomial time such that solving the instance of P will solve the instance of P' as well. Informally, the reducibility of P' to P implies that P' can be considered as a special case of P, so that P is at least as hard as P'.

P is called *NP-hard* if $P' \propto P$ for every $P' \in NP$. In that case, P is at least as hard as any problem in NP. P is called *NP-complete* if P is NP-hard and $P \in NP$. Thus, the NP-complete problems are the most difficult problems in NP.

A polynomial algorithm for an NP-complete problem P could be used to solve all problems in NP in polynomial time, since for any instance of such a problem the construction of the corresponding instance of P and its solution can be both effected in polynomial time. We note the following two important consequences.

(i) It is very unlikely that $P = NP$, since NP contains many notorious com-

binatorial problems, for which in spite of a considerable research effort no polynomial algorithms have been found so far.
(ii) It is very unlikely that P $\in \mathcal{P}$ for any NP-complete P, since this would imply that $\mathcal{P} = \mathcal{NP}$ by the earlier argument.

The first NP-completeness result is due to Cook [Cook 1971]. He designed a "master reduction" to prove that every problem in $\mathcal{NP}$ is reducible to the so-called SATISFIABILITY problem. Starting from this result, Karp [Karp 1972] and many others (see, *e.g.*, [Karp 1975; Garey & Johnson 1979; Lenstra & Rinnooy Kan 1979]) identified a large number of NP-complete problems in the following way. One can establish NP-completeness of some P $\in \mathcal{NP}$ by specifying a reduction P' $\propto$ P with P' already known to be NP-complete: for every P" $\in \mathcal{NP}$, P" $\propto$ P' and P' $\propto$ P then imply that P" $\propto$ P as well. In this way, PARTITION has been proved to be NP-complete [Karp 1972].

As far as *optimization problems* are concerned, one usually reformulates, say, a minimization problem as a recognition problem by asking for the existence of a feasible solution with value at most equal to a given threshold. When this recognition problem can be proved to be *NP-complete*, the corresponding optimization problem might be called *NP-hard* in the sense that the existence of a polynomial algorithm for its solution would imply that $\mathcal{P} = \mathcal{NP}$.

3. SOME BASIC MODELS

Suppose that n *jobs* or *tasks* J_j (j = 1,...,n) have to be processed on m *parallel machines* or *processors* M_i (i = 1,...,m). Each machine can handle at most one job at a time; each job can be executed on any one of the machines. The problem types that will be dealt with in this survey are characterized by a three-field classification $\alpha|\beta|\gamma$ [Graham *et al.* 1979].

The first field $\alpha = \alpha_1\alpha_2$ specifies the *machine environment*. Let p_{ij} denote the time required to process J_j on M_i. Three possible values of α_1 will be considered:

- P (*identical machines*): $p_{ij} = p_j$, *i.e.*, the processing time of J_j on M_i is equal to the execution requirement p_j of J_j, for all M_i;
- Q (*uniform machines*): $p_{ij} = p_j/s_i$, *i.e.*, the processing time of J_j on M_i is equal to the execution requirement p_j of J_j divided by the speed s_i of M_i;
- R (*unrelated machines*): p_{ij} is arbitrary.

If α_2 is a positive integer, then m is *constant* and equal to α_2; if α_2 is empty, then m is *variable*.

The second field β indicates certain *job characteristics*. In this section, β will be empty, which implies the following:

- all p_{ij} (or p_j) are arbitrary nonnegative integers;
- no precedence constraints between the jobs are specified;
- no preemption (job splitting) is allowed;
- all jobs become available for processing at time 0.

The notation to indicate which of these assumptions are not met will be defined in later sections.

The third field γ corresponds to the *optimality criterion* chosen. Any feasible schedule defines a *completion time* C_j of J_j (j = 1,...,n). We will consider the minimization of two criteria:

- *maximum completion time* $C_{max} = \max\{C_1,\ldots,C_n\}$;
- *total completion time* $\sum C_j = C_1+\ldots+C_n$.

The optimal value of γ will be denoted by γ^*, the value produced by an (approximation) algorithm A by $\gamma(A)$.

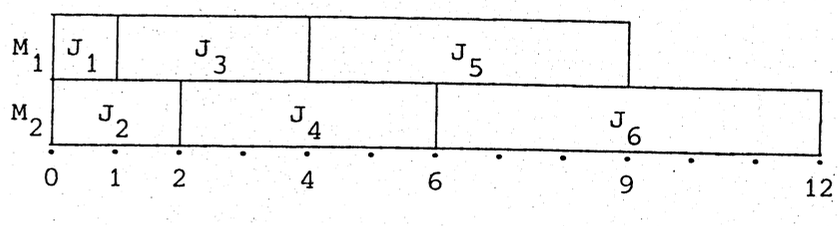
Examples 1, 2 and 3 illustrate this problem classification. *Gantt charts* are used to represent schedules in an obvious way.

Example 1. $P2||\sum C_j$

problem: minimize total completion time on two identical machines.

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

optimal schedule:



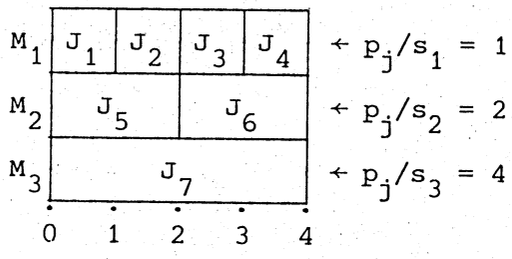
$\sum C_j^* = 34$

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

problem: minimize maximum completion time on three uniform machines.

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

optimal schedule:



$\leftarrow p_j/s_1 = 1$

$\leftarrow p_j/s_2 = 2$

$\leftarrow p_j/s_3 = 4$

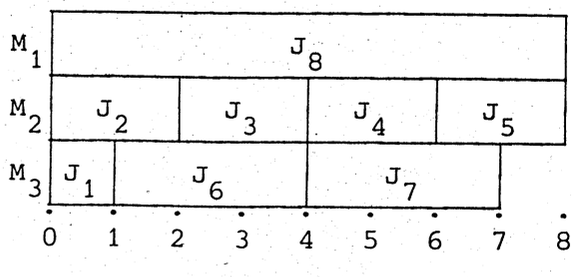
$C_{max}^* = 4$

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

problem: minimize maximum completion time on m unrelated machines.

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

optimal schedule:



$C_{max}^* = 8$

Let us survey the results available for these basic models. It will turn out that the $\sum C_j$ problems are quite easy, while the C_{max} problems are very difficult.

The *shortest processing time* (SPT) rule solves $P||\sum C_j$ in $O(n \log n)$ time in the following way [Conway *et al.* 1967]. Assume that $n = \ell m$ (dummy jobs with zero processing times are added if not), renumber the jobs such that $p_1 \leq \ldots \leq p_n$, and schedule the m jobs $J_{(k-1)m+1}, J_{(k-1)m+2}, \ldots, J_{km}$ in the k-th position on the m machines ($k = 1,\ldots,\ell$). Example 1 illustrates this rule. An optimality proof is straightforward: in the criterion value $\sum C_j$, the processing time of a job in the k-th position on a machine is counted $\ell+1-k$ times, and hence $\sum C_j$ is equal to the inner product of two n-vectors $(\ell,\ldots,\ell, \ell-1,\ldots,\ell-1,\ldots,1,\ldots,1)$ and $(p_1,\ldots,p_n)$; since the multipliers in the former vector are nonincreasing, $\sum C_j$ is minimal if the processing times in the latter one are nondecreasing.

This algorithm has been generalized to solve $Q||\sum C_j$ in $O(n \log n)$ time as well [Conway *et al.* 1967; Horowitz & Sahni 1976].

The most general case $R||\sum C_j$ can be formulated and solved as an $m \times n$ *linear transportation* problem in $O(n^3)$ time [Horn 1973; Bruno *et al.* 1974]. Let

$$x_{ijk} = \begin{cases} 1 & \text{if } J_j \text{ is in the k-th last position on } M_i, \\ 0 & \text{otherwise.} \end{cases}$$

Then the problem is to minimize

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

subject to

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

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

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

Thus, the minimization of $\sum C_j$ requires polynomial time, even on m unrelated machines. In contrast, the minimization of C_{max} is NP-hard, even on two identical machines.

The NP-hardness proof for $P2||C_{max}$ is trivial. Given any instance of PARTITION, defined by positive integers $a_1,\ldots,a_t,b$ (see Section 2), we construct an instance of $P2||C_{max}$ by defining $n = t$ and $p_j = a_j$ $(j = 1,\ldots,n)$.

Clearly, there exists a subset $S \subset \{1,\ldots,t\}$ with $\sum_{j\in S} a_j = b$ if and only if there exists a schedule with $C_{max} \leq b$. It follows that PARTITION is reducible to $P2||C_{max}$, and since PARTITION is NP-complete [Karp 1972], $P2||C_{max}$ is NP-hard. This implies that all generalizations of $P2||C_{max}$, such as $P3||C_{max}$, $\ldots,P||C_{max}$, $Q2||C_{max},\ldots,R||C_{max}$, are NP-hard as well.

As a consequence, it seems unavoidable that optimization algorithms for these problems will be of an enumerative nature. A general *dynamic programming* scheme [Rothkopf 1966; Lawler & Moore 1969] has wide applicability. For $P||C_{max}$, the scheme is as follows. Let

$$B_j(t_1,\ldots,t_m) = \begin{cases} \textit{true} & \text{if } J_1,\ldots,J_j \text{ can be scheduled on } M_1,\ldots,M_m \\ & \text{such that } M_i \text{ is busy from 0 to } t_i \ (i = 1,\ldots,m), \\ \textit{false} & \text{otherwise,} \end{cases}$$

with

$$B_0(t_1,\ldots,t_m) = \begin{cases} \textit{true} & \text{if } t_i = 0 \ (i = 1,\ldots,m), \\ \textit{false} & \text{otherwise.} \end{cases}$$

Then the recursive equation is

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

Let C be an upper bound on the optimal value C^*_{max}. For $j = 0,1,\ldots,n$, compute $B_j(t_1,\ldots,t_m)$ for $t_i = 0,1,\ldots,C$ $(i = 1,\ldots,m)$, and determine

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

This procedure solves $P||C_{max}$ in $O(nC^m)$ time. For large values of C, a *branch-and-bound* method may be preferable. All these optimization methods, however, require prohibitive running times in the worst case.

As argued before, the NP-hardness of $P||C_{max}$ also justifies the use of fast approximation algorithms. It has become fashionable to subject such an algorithm to a *worst-case analysis* in order to derive a guarantee on its relative performance. One of the earliest results of this type concerns the solution of $P||C_{max}$ by *list scheduling* (LS), whereby a priority list of the jobs is given and at each step the first available machine is selected to

process the first available job on the list [Graham 1966]:

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

For the *longest processing time* (LPT) rule, whereby the list contains the jobs in order of nonincreasing p_j, the bound improves considerably [Graham 1969]:

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

Examples 4 and 5 demonstrate that these bounds are the best possible ones.

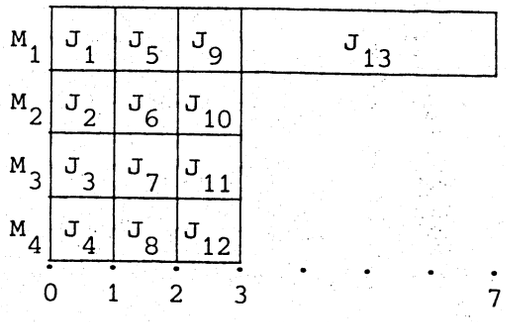
Example 4. $P||C_{max}(LS)$

worst problem instance:

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

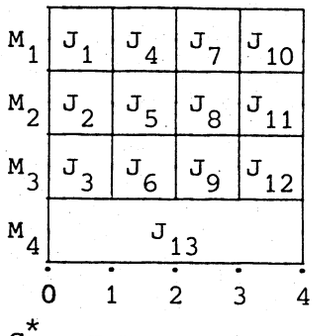
$(p_1,\ldots,p_n) = (1,\ldots,1,m).$

approximate schedule:



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

optimal schedule:



$C^*_{max} = m$

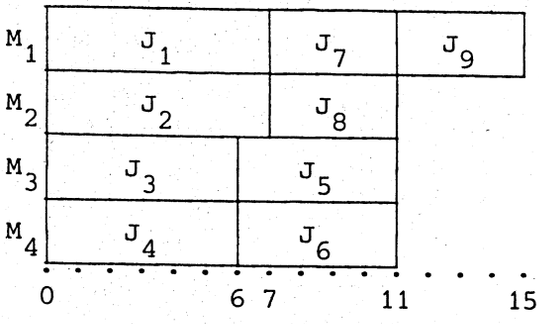
Example 5. $P||C_{max}(LPT)$

worst problem instance:

$n = 2m+1;$

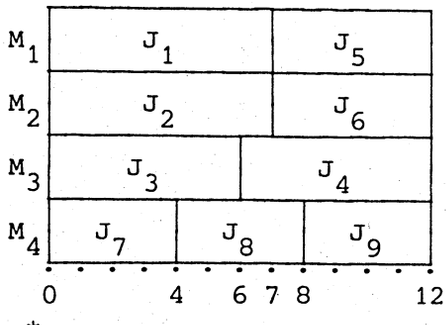
$(p_1,\ldots,p_n) = (2m-1,2m-1,2m-2,2m-2,\ldots,m+1,m+1,m,m,m).$

approximate schedule:



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

optimal schedule:



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

4. UNIT PROCESSING TIMES AND THE INFLUENCE OF PRECEDENCE CONSTRAINTS

The results of Section 3 suggest that additional simplifying assumptions are necessary to solve $P||C_{max}$ optimally in polynomial time. In this section, we assume that all jobs have *unit processing times*, which will be indicated in the second field of our problem classification by $p_j=1$. This assumption also allows us to investigate the influence of *precedence constraints* between the jobs. It turns out to be useful to distinguish between two types of precedence constraints:

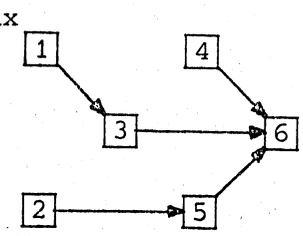
- *prec* (arbitrary precedence constraints): a directed acyclic graph G with vertices 1,...,n is given; if G contains a directed path from j to k, we write $J_j \rightarrow J_k$ and require that J_j is completed before J_k can start;
- *tree* (tree-like precedence constraints): G is a rooted tree with outdegree at most one for each vertex.

Examples 6 and 7 below will illustrate these concepts.

One of the oldest results in this problem category is the solution of $P|tree,p_j=1|C_{max}$ in O(n) time [Hu 1961]. Hu's algorithm involves *critical path scheduling*: define the *level* ℓ_j of J_j as the number of vertices on the unique path from j to the root of the tree, and apply list scheduling to a list which contains the jobs in order of nonincreasing ℓ_j. Example 6 illustrates this algorithm.

Example 6. $P|tree,p_j=1|C_{max}$

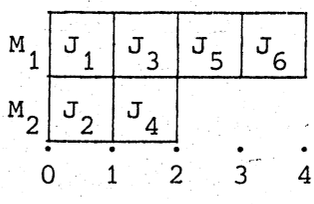
instance: m = 2; n = 6; G:



⇒

j	1	2	3	4	5	6
ℓ_j	3	3	2	2	2	1

optimal schedule:



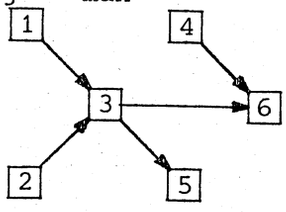
$C^*_{max} = 4$

The second basic result is the solution of $P2|prec,p_j=1|C_{max}$ in polynomial time. An $O(n^3)$ algorithm [Fujii *et al.* 1969,1971] is as follows: construct

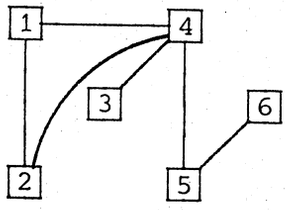
an undirected graph H with vertices 1,...,n and edges {j,k} whenever neither $J_j \rightarrow J_k$ nor $J_k \rightarrow J_j$, and derive an optimal schedule from a maximum cardinality *matching* (*i.e.*, a set of vertex-disjoint edges) in H. Example 7 illustrates this algorithm. We note that the problem can still be solved in $O(n^3)$ time if, in addition, each job is constrained to be processed between its *release date* and its *due date* [Garey & Johnson 1977].

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

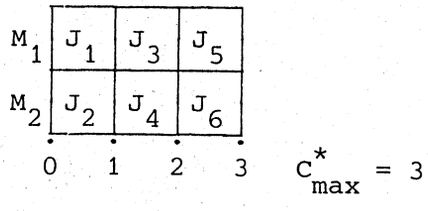
instance: n = 6; G:



⇒ H:



optimal schedule:



$C^*_{max} = 3$

For any constant m ≥ 3, the complexity of $Pm|prec,p_j=1|C_{max}$ is an open question. However, $P|prec,p_j=1|C_{max}$ is known to be NP-hard [Ullman 1975; Lenstra & Rinnooy Kan 1978]. The latter proof implies that no polynomial approximation algorithm for $P|prec,p_j=1|C_{max}$ could ever achieve a worst-case bound better than $\frac{4}{3}$, unless *P = NP*. For *critical path scheduling* (CP), it has been shown [Chen 1975; Chen & Liu 1975] that

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

and these bounds are tight.

5. PREEMPTION AND THE INFLUENCE OF RELEASE DATES

We now consider a second modification of the multiprocessor scheduling models that will lead to several polynomial optimization algorithms. More specifically, we assume that unlimited *preemption* is allowed: the processing of any job may arbitrarily often be interrupted and resumed at the same time on a different machine or at a later time on any machine. This will be indicated in the second field of our problem classification by *pmtn*.

It has been shown that for $P|pmtn|\sum C_j$ there is no advantage to preemption at all [McNaughton 1959]. Hence, the nonpreemptive SPT rule of Section 3 can be applied to solve the problem in $O(n \log n)$ time.

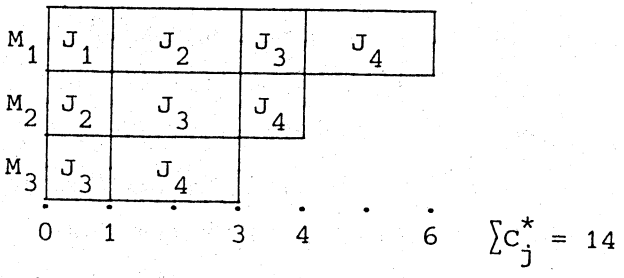
A preemptive version of the SPT rule solves $Q|pmtn|\sum C_j$ in $O(n \log n + mn)$ time [Gonzalez 1977]: place the jobs in SPT order, and schedule each successive job preemptively so as to minimize its completion time. The resulting schedule contains at most $(m-1)(n-\frac{m}{2})$ preemptions. Example 8 illustrates this rule.

Very little is known about $R|pmtn|\sum C_j$. This is one of the more intriguing open problems in the area of multiprocessor scheduling.

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

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

optimal schedule:



$\sum C_j^* = 14$

$P|pmtn|C_{max}$ and $Q|pmtn|C_{max}$ are distinguished because in both cases there is a simple closed form expression which is an obvious lower bound on C_{max}^* whereas a schedule meeting this bound can be constructed in polynomial time. For $P|pmtn|C_{max}$, we have

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

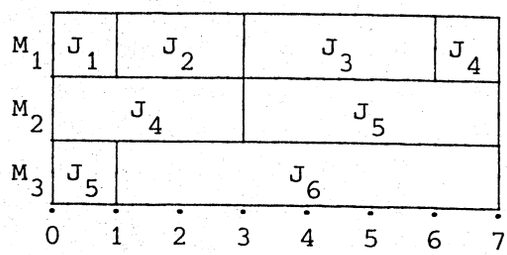
The *wrap-around* rule solves the problem in $O(n)$ time [McNaughton 1959]: fill the machines successively, scheduling the jobs in any order and splitting a

job whenever the above time bound is met. There will be at most m-1 preemptions. Example 9 illustrates this rule.

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

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

optimal schedule:



For $Q|pmtn|C_{max}$, we have

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

where $s_1 \geq \ldots \geq s_m$ and $p_1 \geq \ldots \geq p_n$. If the machines and jobs are ordered in this way, a complicated algorithm solves the problem in $O(n)$ time [Gonzalez & Sahni 1978]. It generates at most $2(m-1)$ preemptions.

$R|pmtn|C_{max}$ can be formulated as a *linear programming* problem [Lawler & Labetoulle 1978]. Let

$$x_{ij} = \text{time spent by } J_j \text{ on } M_i.$$

Then the problem is to minimize

$$C_{max}$$

subject to

$$\sum_{i=1}^{m} x_{ij}/p_{ij} = 1 \quad (j = 1,\ldots,n),$$

$$\sum_{i=1}^{m} x_{ij} \leq C_{max} \quad (j = 1,\ldots,n),$$

$$\sum_{j=1}^{n} x_{ij} \leq C_{max} \quad (i = 1,\ldots,m),$$

$$x_{ij} \geq 0 \quad (i = 1,\ldots,m;\ j = 1,\ldots,n).$$

Khachian has shown that linear programs can be solved in polynomial time [Khachian 1979]. Given a solution (x^*_{ij}), a feasible schedule can be con-

structed in polynomial time as well [Gonzalez & Sahni 1976]. There will be no more than about $\frac{7}{2}m^2$ preemptions.

We may extend the preemptive scheduling models by assuming that J_j becomes available for processing at a given integer *release date* r_j ($j = 1,\ldots,n$). This will be indicated in the second field of the classification by r_j. The resulting models are far from trivial, and we restrict ourselves to mentioning the most important results.

When scheduling subject to release dates, one can distinguish between three types of algorithms. An algorithm is *on-line* if at any time only information about the available jobs is required. It is *nearly on-line* if in addition the next release date has to be known. It is *off-line* if all information is available in advance.

$P|pmtn,r_j|\sum C_j$ and $Q|pmtn,r_j|\sum C_j$ are very much open. All we know about these problems is that no on-line algorithm exists, even for the case of two identical machines [Labetoulle *et al.* 1979].

$P|pmtn,r_j|C_{max}$ can be solved by an $O(mn)$ on-line algorithm [Horn 1974; Gonzalez & Johnson 1977], and $Q|pmtn,r_j|C_{max}$ by an $O(n^2)$ nearly on-line algorithm [Sahni & Cho 1979; Labetoulle *et al.* 1979].

Finally, we assume that in addition J_j has to be completed not later than a given *due date* d_j ($j = 1,\ldots,n$), and we replace the objective of minimizing C_{max} by testing for the existence of a feasible preemptive schedule with respect to release dates and due dates. It has been shown that no nearly on-line algorithm exists, even for the case of two identical machines [Sahni 1979]. However, off-line algorithms are still available: $P|pmtn,r_j,d_j|-$ is solvable by an $O(n^3)$ network flow computation [Horn 1974], and $Q|pmtn,r_j,d_j|-$ by means of an $O(n^6)$ "generalized" network flow model [Martel 1979].

6. CONCLUDING REMARKS

We have surveyed a few of the many recent results in the area of multiprocessor scheduling. There are several topics that we have not dealt with; in particular, we mention the extension of the model to include additional *resource constraints*, for which many results are now available [Graham *et al.* 1979; Blazewicz *et al.* 1980]. The development of increasingly sophisticated algorithmic techniques combined with a further application of the tools from computational complexity theory should continue to render the area of multiprocessor scheduling an interesting one to theoreticians and practitioners alike.

ACKNOWLEDGMENT

This research was partially supported by NATO Special Research Grant 9.2.02 (SRG.7).

REFERENCES

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, 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 (ed.) (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 (1977) A new algorithm for preemptive scheduling of trees. Technical Report 222, Computer Science Department, Pennsylvania State University.

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 (eds.) (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 multiprocessor 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.

LIST OF REPORTS 1980

8000 "List of Reprints, nos 241-260, Abstracts of Reports Second Half 1979".

8001/O "A Stochastic Method for Global Optimization", by C.G.E. Boender, A.H.G. Rinnooy Kan, L. Stougie and G.T. Timmer.

8002/M "The General Linear Group of Polynomial Rings over Regular Rings", by A.C.F. Vorst.

8003/O "A Recursive Approach to the Implementation of Enumerative Methods", by J.K. Lenstra and A.H.G. Rinnooy Kan.

8004/E "Linearization and Estimation of the Addi-Log Budget Allocation Model", by P.M.C. de Boer and J. van Daal.

8005/O "The Complexity of the Constrained Gradient Method for Linear Programming", by J. Telgen.

8006/E "On Functions with Small Differences", by J.L. Geluk and L. de Haan.

8007/O "Analytical Evaluation of Hierarchical Planning Systems", by M.A.H. Dempster, M.L. Fisher, L. Jansen, B.J. Lageweg, J.K. Lenstra and A.H.G. Rinnooy Kan.

8008/S "Looking at Multiway Tables (continued), by A.P.J. Abrahamse and W.M. Lammerts van Bueren.

8009/O "An Introduction to Multiprocessor Scheduling", by J.K. Lenstra and A.H.G. Rinnooy Kan.