AN INTRODUCTION TO MULTIPROCESSOR SCHEDULING

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

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 the 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.

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 computational complexity of combinatorial problems; a more detailed treatment can be found in [23, 24, 11, and 31]. 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 [8] 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 [27]. 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 The notation $q(n)=O(p(n))$ means that there exists a constant $c>0$ such that $q(n)\leq cp(n)$ for all $n>0$. 
a result by proving that the problem in question is NP-complete /7,23/. 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 problem, 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 infeasibility of a bad (superpolynomial) optimization algorithm or resort to using a good (polynomial) approximation algorithm.

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 $S \subseteq \{1, \ldots, t\}$ 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 /1/, 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. Flexibility 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, it's 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 $S \subseteq \{1, \ldots, t\}$ 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: $P' \leq P$) 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 $P' \leq P$ 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 are the most difficult problems in $NP$.

A polynomial algorithm for an NP-complete problem 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 combinatorial problems, for which in spite of a considerable research effort no polynomial algorithms have been found so far.

(ii) It is very unlikely that $P \subseteq NP$ for any NP-complete $P$, since this would imply that $P = NP$ by the earlier argument.

The first NP-completeness result is due to Cook /7/. He designed a “master reduction” to prove that every problem in $NP$ is reducible to the so-called SATISFIABILITY problem. Starting from this result, Karp /23/ and many others (see, e.g., /24, 11, 31/) identified a large number of NP-complete problems in the following way. One can establish NP-completeness of some $P \subseteq NP$ by specifying a reduction $P' \leq P$ with $P'$ already known to be NP-complete: for every $P' \in NP$, $P' = P$ and $P'$ then imply that $P \subseteq NP$ as well. In this way, PARTITION has been proved to be NP-complete /23/.

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 an algorithm for its solution would imply that $P = NP$.

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 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 $a|b|c/18$.

The first field $a=a_1a_2$ specifies the machine environment. Let $p_{ij}$ denote the time required to process $J_j$ on $M_i$. Three possible values of $a_1$ 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 $M_i$;

- $C$ (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 $a_2$ is a positive integer, then $m$ is constant and equal to $a_2$ if $a_2$ is empty, then $m$ is variable.

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

- all $p_{ij}$ (or $p_i$) 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 $c$ 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 $C_{\max}=\max\{C_1,...,C_n\}$;

- total completion time $\sum C_j=C_1+...+C_n$.

The optimal value of $\gamma$ will be denoted by $\gamma^*$, 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,...,6).

optimal schedule:

$$
\begin{array}{|c|c|c|c|c|c|}
\hline
M_1 & J_1 & J_2 & J_3 & J_4 & J_5 \\
\hline
M_2 & J_1 & J_2 & J_3 & J_4 & J_5 \\
\hline
0 & 1 & 2 & 4 & 6 & 9
\end{array}
$$

$\sum C_j = 34$

Example 2. $C3||C_{\max}$

problem: minimize maximum completion time on three uniform machines.

instance: $s_1=4$, $s_2=2$, $s_3=1$; $n=7$; $p_j=3$ (j=1,...,7).

optimal schedule:

$$
\begin{array}{|c|c|c|c|c|c|c|}
\hline
M_1 & J_1 & J_2 & J_3 & J_4 & J_5 & J_6 \\
\hline
M_2 & J_1 & J_2 & J_3 & J_4 & J_5 & J_6 \\
M_3 & J_1 & J_2 & J_3 & J_4 & J_5 & J_6 \\
\hline
0 & 1 & 2 & 3 & 4 & 6 & 9
\end{array}
$$

$c_{\max} = 4$

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

problem: minimize maximum completion time on $m$ unrelated machines.

instance: $m=3$; $n=8$

$$
\begin{array}{l}
P_{11} = 1, P_{1j} = 1 (j = 2,...,7), P_{18} = 8, \\
P_{21} = 1, P_{2j} = 2 (j = 2,...,7), P_{28} = 9, \\
P_{31} = 1, P_{3j} = 3 (j = 2,...,7), P_{38} = 9.
\end{array}
$$
Thus, the minimization of $C_j^*$ requires polynomial time, even on $m$ unrelated machines. In constrast, the minimization of $C_{\text{max}}$ is NP-hard, even on two identical machines.

The NP-hardness proof for $P_2||C_{\text{max}}$ is trivial. Given any instance of PARTITION, defined by positive integers $a_1, \ldots, a_n$ (see Section 2), we construct an instance of $P_2||C_{\text{max}}$ by defining $n$ and $P_j = a_j + \sum_{i < j} a_i$ if and only if there exists a schedule with $C_{\text{max}} \leq b$. It follows that PARTITION is reducible to $P_2||C_{\text{max}}$, and since PARTITION is NP-complete, $P_2||C_{\text{max}}$ is NP-hard. This implies that all generalizations of $P_2||C_{\text{max}}$ such as $P_3||C_{\text{max}}, \ldots, P_{\lambda}||C_{\text{max}}$, $\Omega^2||C_{\text{max}}$, $\ldots$, $\Omega^2||C_{\text{max}}$, 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 /34/ /49/ has wide applicability. For $P_2||C_{\text{max}}$, the scheme is as follows. Let

$$B_j(t_1, \ldots, t_m) = \begin{cases} \text{true} & \text{if } J_1, \ldots, J_j \text{ can be scheduled on } M_1, \ldots, M_m \\ \text{false} & \text{otherwise} \end{cases}$$

such that $M_1$ is busy from $0$ to $t_1(i=1, \ldots, m)$, with

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

Then the recursive equation is

$$B_j(t_1, \ldots, t_m) = \sum_{i=1}^{n} B_j-1(t_1, \ldots, t_{i-1}, t_i-t_j, t_j, t_{j+1}, \ldots, t_m).$$

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

$$C_{\text{max}}^* = \min(\max(t_1, \ldots, t_m) | B_n(t_1, \ldots, t_m) = \text{true}).$$

This procedure solves $P_2||C_{\text{max}}$ in $O(nC^m)$ time.

Questa 6 - V. 5, n° 1 (Març 1981)
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 $P|C_{\text{max}}$ 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 $P|C_{\text{max}}$ 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 /16/:

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

For the longest processing time (LPT) rule, whereby the list contains the jobs in order of nonincreasing $P_j$, the bound improves considerably /17/:

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

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

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

worst problem instance:

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

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

approximate schedule:

\begin{align*}
M_1 & : J_1 J_5 J_9 J_{13} \\
M_2 & : J_2 J_6 J_{10} \\
M_3 & : J_3 J_7 J_{11} \\
M_4 & : J_4 J_8 J_{12} \\
\end{align*}

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

optimal schedule:

\begin{align*}
M_1 & : J_1 J_4 J_7 J_{10} \\
M_2 & : J_2 J_5 J_8 J_{11} \\
M_3 & : J_3 J_6 J_9 J_{12} \\
M_4 & : J_{13} \ldots \ldots \\
\end{align*}

$C^*_{\text{max}} = m$

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

worst problem instance:

$n = 2m+1$,

$(p_1, \ldots, p_n) = (2m+1, 2m-1, 2m-2, 2m-3, \ldots, m+1, m+1, m, m, m)$.

approximate schedule:

\begin{align*}
M_1 & : J_1 J_7 J_9 \ldots \ldots \\
M_2 & : J_2 J_8 \ldots \ldots \\
M_3 & : J_3 J_5 \ldots \ldots \\
M_4 & : J_4 J_6 \ldots \ldots \\
\end{align*}

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

optimal schedule:

\begin{align*}
M_1 & : J_1 J_5 J_9 \\
M_2 & : J_2 J_6 J_{10} \\
M_3 & : J_3 J_7 J_{11} \\
M_4 & : J_4 J_8 J_{12} \\
\end{align*}

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

4. Unit Processing Times and the Influence of Precedence Constraints

The results of Section 3 suggest that additional simplifying assumptions are necessary to solve $P|C_{\text{max}}$ 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 $1, \ldots, n$ 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 $P|\text{tree}, P_j = 1|C_{\text{max}}$ in
O(n) time /22/. 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. \( P[\text{tree}, p_1=1]/C_{\text{max}} \)

instance: \( m = 2; n = 6; G:\)

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
J_j & 1 & 3 & 3 & 2 & 2 & 2 & 1 \\
\end{array}
\]

optimal schedule:

\[
\begin{array}{cccccc}
M_1 & J_1 & J_3 & J_5 & J_6 \\
M_2 & J_2 & J_4 \\
0 & 1 & 2 & 3 & 4 \quad C_{\text{max}} = 4 \\
\end{array}
\]

The second basic result is the solution of \( P_{2}\{\text{prec}, p_1=1\}/C_{\text{max}} \) in polynomial time. An \( O(n^3) \) algorithm /9/ is as follows: construct an undirected graph \( H \) with vertices \( 1, \ldots, n \) 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 /10/.

Example 7. \( P_2|\text{prec}, p_1=1|C_{\text{max}} \)

instance: \( n = 6; G:\)

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 \quad 6 \\
G & 1 & 3 & 5 & 6 \\
\end{array}
\]

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 \quad 6 \\
H & 1 & 3 & 5 & 6 \\
\end{array}
\]

optimal schedule:

\[
\begin{array}{cccccc}
M_1 & J_1 & J_3 & J_5 & J_6 \\
M_2 & J_2 & J_4 \\
0 & 1 & 2 & 3 & 4 \quad C_{\text{max}} = 3 \\
\end{array}
\]

For any constant \( m \geq 3 \), the complexity of \( P_m|\text{prec}, p_1=1|C_{\text{max}} \) is an open question. However, \( P|\text{prec}, p_1=1|C_{\text{max}} \) is known to be \( \text{NP-hard} /37/; /13/ \). The latter proof implies that no polynomial approximation algorithm for \( P|\text{prec}, p_1=1|C_{\text{max}} \) could ever achieve a worst-case bound better than \( \frac{4}{3} \), unless \( \text{P=NP} \). For critical path scheduling (CP), it has been shown /4/1/5/, that

\[
C_{\text{max}}^{(\text{CP})}/C_{\text{max}}^{*} \leq \begin{cases} 
\frac{3}{2} - \frac{1}{n-1} & \text{for } m = 2, \\
\frac{3}{m} & \text{for } m \geq 3,
\end{cases}
\]

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 \( \text{preempt} \).

It has been shown that for \( P|m\text{mt}|\bar{C}_j \) there is no advantage to preemption at all /13/. 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 \( 0|\text{pmtn}|\bar{C}_j \) in \( O(n \log n + m n) \) time /12/; 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 \( P|m\text{mt}|\bar{C}_j \). This is one of the more intriguing open problems in the area of multiprocessor scheduling.

Example 8. \( Q|m\text{tn}|\bar{C}_j \)

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:

\[
\begin{array}{cccccc}
M_1 & J_1 & J_2 & J_3 & J_4 \\
M_2 & J_2 & J_3 & J_4 \\
M_3 & J_3 & J_4 \\
0 & 1 & 3 & 4 & 6 \quad \bar{C}_j^{*} = 14 \\
\end{array}
\]
Then the problem is to minimize

\[ C_{\text{max}} \]

subject to

\[
\begin{align*}
\sum_{i=1}^{m} x_{ij} / p_{ij} - 1 & \quad (j = 1, \ldots, n), \\
\sum_{i=1}^{m} x_{ij} & \leq C_{\text{max}} \quad (j = 1, \ldots, n), \\
\sum_{j=1}^{n} x_{ij} & \leq C_{\text{max}} \quad (i = 1, \ldots, n), \\
x_{ij} & \geq 0 \quad (i = 1, \ldots, m; j = 1, \ldots, n).
\end{align*}
\]

Khachian has shown that linear programs can be solved in polynomial time /25/. Given a solution \( x_{ij}^* \), a feasible schedule can be constructed in polynomial time as well /14/. There will be no more than about \( \frac{7}{2} m \) 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 subjects 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.

\[ P|\text{pmtn}, r_j| C_{\text{max}} \text{ and } O|\text{pmtn}, r_j| C_{\text{max}} \text{ are very much open. All we know about these problems is that no on-line algorithm exists, even -- for the case of two identical machines /26/}. \]

\[ P|\text{pmtn}, r_j| C_{\text{max}} \text{ can be solved by an } O(mn) \text{ on-line algorithm /20/13/}. \text{ and } O|\text{pmtn}, r_j| C_{\text{max}} \text{ by an } O(n^2) \text{ nearly on-line algorithm /36/}. \]

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 \( C_{\text{max}} \) 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 /35/. However, off-line algorithms are still available: \( P|p_{mn}, r_j, d_j| \) is solvable by an \( O(n^3) \) network flow -- computation /20/, and \( 0|p_{mn}, r_j, d_j| \) by --- means of an \( O(n^5) \) "generalized" network flow model /32/.

6. CONCLUDING REMARKS

We have surveyed a few of the many recent re-
sults in the area of multiprocessor schedul-
ing. There are several topics that we have
not dealt with; in particular, we mention --
the extension of the model to include addi-
tional resource constraints, for which many
results are now available /18/; /2/. The
development of increasingly sophisticated --
algorithmic techniques combined with a fur-
ther application of the tools from computa-
tional complexity theory should continue to
render the area of multiprocessor scheduling
an interesting one to theoreticians and prac-
titioners alike.

7. ACKNOWLEDGMENT

This research was partially supported by NA-
TO Special Research Grant 9.2.02 (SRG.7).

8. REFERENCES

/1/ BELLMAN, R.E. and DREYFUS, S.E.: "Applied
Dynamic Programming", Princeton University

/2/ BLAZEWICZ, J.K.LENSTRA, A.H.G. KNAAY KAN; --
"Scheduling subject to resource cons-
traints: classification and complexity",
Report, Mathematisch Centrum, Amsterdam,
1980.

/3/ BRUNO, J.E., COFFMAN, E.G. and SETHI, R.: 
"Scheduling independent task to reduce 
mean finishing time", Comm. ACM, v. 17, 
1974, pp.382-387.

/4/ CHEN, N.F.: "An analysis of scheduling --
algorithm in multiprocessor computing sys-
tems",Technical Report UIUCDCS-R-75-724, Department of Computer Science, ---
University of Illinois at Urbana-Champaign,
1975.

/5/ CHEN, N.F., and LIU, C.L.: "On a class of 
scheduling algorithms for multipro-
cessors computing systems". In: T.Y.--
FENG (ed.) Parallel Processing, Lecture 
Notes in Computer Science 24, Springer,
Berlin, 1975, pp. 1-16.

/6/ CONWAY, R.W., MAXWELL, W.L. and MILLER,
L.W.: "Theory of Scheduling", Addison--

/7/ COOK, S.A.: "The complexity of theorem-
proving procedures", Proc. 3rd Annual --
151-158.

/8/ EDMONDS, J.:"Paths, trees, and flowers.

/9/ FUJII, M., KASAMI, T. and NINOMIYA, K.: 
"Optimal sequencing of two equivalent --
processors", SIAM J. Appl. Math. v.17, 
1969,1971, pp. 784-789; Erratum, v.20,
pp.141.

/10/ GAREY, M.R. and JOHNSON, D.S.: "Two pro-
cessor scheduling with start-times and --
deadlines", SIAM J. Comput. v. 6, 1977,
pp. 416-426.

and Intractability : a Guide to the ----
Theory of NP-Completeness", Freeman, San

/12/ GONZALEZ, T.: "Optimal mean finish time 
preemptive schedules", Technical Report 
220, Computer Science Department, Pennsyl-
vania State University, 1977.

algorithm for preemptive scheduling of --
trees", Technical Report 222, Computer 
Science Department, Pennsylvania State --
University, 1977.

/14/ GONZALEZ, T. and SAHNI, S.: "Open shop 
scheduling to minimize finish time", J. 
Assoc. Comp. Mach. v. 23, 1976, pp. 665-
679.

/15/ GONZALEZ, T. and SAHNI, S.: "Preemptive 
scheduling of uniform processor system", 
92-101.

/16/ GRAHAM, R.L.: "Bounds for certain multi-
processing anomalies", Bell System Tech.

/17/ GRAHAM, R.L.: "Bounds on multiprocessing

/18/ GRAHAM, R.L., LAWLER, E.L., LENSTRA, J.
K. and RINNOY KAN, A.H.G.: "Optimiza-
tion and approximation in deterministic
sequencing and scheduling: a survey",--
-326.

/19/ HORN, W.A.: "Minimizing average flow
time with parallell machines", Operations

/20/ HORN, W.A.: "Some simple scheduling algo-
rithms", Naval Res. Logist. Quart. v.
21, 1974, pp. 177-185.

/21/ HOROWITZ, E. and SAHNI, S.: "Exact and
approximate algorithms for scheduling
nonidentical processors", J. Assoc. ---

/22/ HU, T.C.: "Parallel sequencing and ----
assembly line problems", Operations Res.
v. 9, 1961, pp. 841-848.

/23/ KARP, R.M.: "Reducibility among combina-
torial problems", In: R.E. MILLER, J.W.
THATCHER (eds.), Complexity of Computer
Computations, Plenum Press, New York, --
1972, pp. 85-103.

/24/ KARP, R.M.: "On the computational com-
plicity of combinatorial problems", Net-
works v.5, 1975, pp. 45-68.

/25/ KHACHIAN, L.G.: "A polynomial algorithm
in linear programming", Soviet Math. --
Dokl. v. 20, 1979, pp. 191-194.

/26/ Labetouille, J., LAWLER, E.L., LENSTRA,
J.K. and RINNOY KAN, A.H.G.: "Preempt-
tive scheduling of uniform machines ---
subject to release dates", Report BW 99,
Mathematisch Centrum, Amsterdam, 1979,

/27/ LAWLER, E.L.: "Combinatorial Optimiza-
tion: Network and Matroids", Holt, ----

/28/ LAWLER, E.L. and LABETOUILLE, J.: "On --
preemptive scheduling of unrelated para-
lel processor by linear programming",
612-619.

/29/ LAWLER, E.L., and MOORE, J.M.: "A func-
tional equation and its application to
resource allocation and sequencing pro-
blems", Management Sci. v. 16, 1969, --
pp. 77-84.

/30/ LENSTRA, J.K. and RINNOY KAN, A.H.G.:
"Complexity of scheduling under preced-
dence constraints", Operations Res. v.

/31/ LENSTRA, J.K. and RINNOY KAN, A.H.G.:
"Computational complexity of discrete-
optimization problems", Ann. Discrete --
Math. v. 4, 1979, pp. 121-140.

/32/ MARTEL, C.: "Generalized network flows
with an application to multiprocessor
scheduling", Computer Science Division,
University of California, Berkeley,1979.

/33/ MCAUGHTON, R.: "Scheduling with dead-
lines and loss functions", Management --
Sci. v. 6, 1959, pp. 1-12.

/34/ ROTHKOPF, M.H.: "Scheduling independent
tasks on parallel processors", Manage-

/35/ SAHNI, S.: "Preemptive scheduling with
due dates", Operation Res. v. 27, 1979,
pp. 925-934.

/36/ SAHNI, S. and CHO, Y.: "Nearly on line
scheduling of a uniform processor system
with release times", SIAM, J. Comput. --
v. 8, 1979, pp. 275-285.

/37/ ULLMAN, J.D.: "NP-complete scheduling
problems", J. Comput. System Sci. v. 10,
1975, pp. 384-393.