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 sheduling 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 /23,24,11,and 31/. The next three sections provide a brief survey of the results available for multiprocessor sheduling 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 exis 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

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

Lenestra, J.K. Mathematisch Centrum, Amsterdam
Rinnooy Kan, A.H.G. Erasmus University, Rotterdam

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 notorius 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 inevitability 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 $S\subset\{1,\ldots,t\}$ 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 /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 asnwered 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, 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' NP. In that case, P is at least as hard as any problem in NP. P is called *NP-complete* if P es 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 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 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 $P \in P$ 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 reductible 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 $P \in NP$ by specifying a - reduction $P' \propto P$ with P' already known to be NP-complete: for every $P'' \in NP$, $P'' \propto P'$ and P' P then imply that $P'' \propto P$ 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 ($j=1,\ldots, n$) have to be processed on m *parallel machines* or processors M_i ($i=1,\ldots,m$). 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 $\alpha|\beta|\gamma$/18/.

The first field $\alpha=\alpha_1\alpha_2$ specifies the *machine enviroment*. 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, them 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 ($j=1,\ldots,n$). 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 $\gamma(A)$.

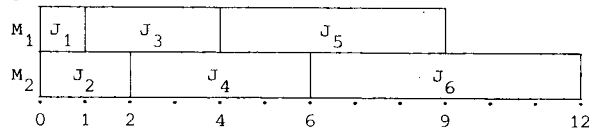
Examples 1, 2 and 3 illustrate this problem classification. *Gantt charts* are used to represent schedules in an obvious way.

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:



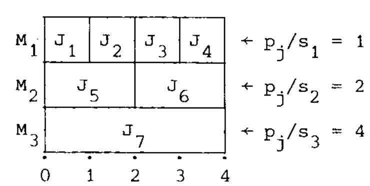
$\sum C_j^* = 34$

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:



$C_{max}^* = 4$

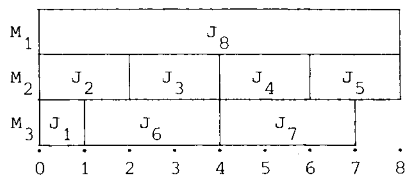
Example 3. $R||C_{max}$

problem: minimize maximium 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 /6/. 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 /6/; /21/.

The most general case $R||\sum C_j$ can be formulated and solved as an mxn *linear transportation* problem in $O(n3)$ time /19/; /3/. 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

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

subjet to

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

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

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

Thus, the minimization of $\sum C_j$ requires polynomial time, even on m unrelated machines. In constrast, 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 $P||C_{max}$ by defining $n=t$ and $P_j = a_j$ ($j=1,\ldots,n$). Clearly, there exists a subset $S \subset \{1,\ldots,t\}$ with $\sum_{j\varepsilon S} a_j = b$ if and only if there exists a schedule with $C_{max} \leq b$. It follows that PARTITION is reductible to $P2||C_{max}$, and since PARTITION is NP-complete /23/, $P2||C_{max}$ 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 /34/; /29/ 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

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

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 subjet such an algorithm to a *worst-case analysis* in order to derive a guarantee on its relative perfomance. 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 /16/:

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

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

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

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

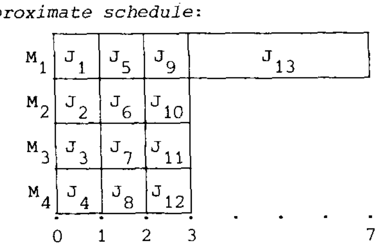
Example 4. $P||C_{max}$(LS)

worst problem instance:

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

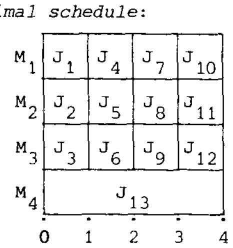
$(p_1,\ldots,p_n) = (1,\ldots,1,m).$

approximate schedule:



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

optimal schedule:



$C^*_{max} = m$

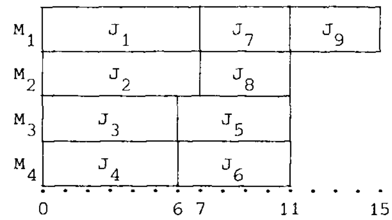
Example 5. $P||C_{max}$(LPT)

worst problem instance:

$n = 2m+1;$

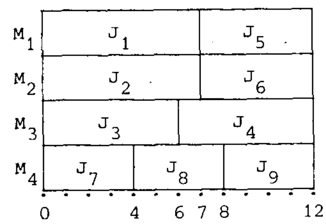
$(p_1,\ldots,p_n) = (2m-1,2m-1,2m-2,2m-2,\ldots,m+1,m+1,m,m,m).$

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.

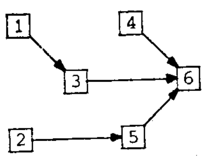
Examples 6 and 7 below will illustrate these concepts.

One of the oldest results in this problem category is the solution of $P|tree,P_j=1|C_{max}$ in

O(n) time /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.

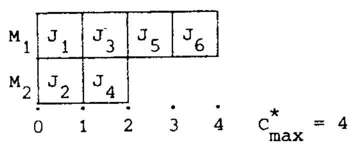
Example 6. $P|tree,p_j=1|C_{max}$

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



$\Rightarrow$ j	1	2	3	4	5	6
ℓ_j	3	3	2	2	2	1

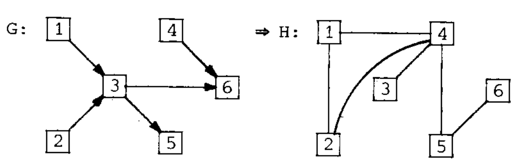
optimal schedule:



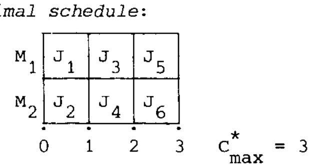
The second basic result is the solution of $P2|prec,P_j=1|C_{max}$ in polynomial time. An $O(n^3)$ algorithm /9/ is as follows: construct an undirected graph H with vertices 1,...,n and edges j,k whenever neither $J_j \to J_k$ nor $J_k \to J_j$, and derive an optimal schedule from a maximum cardinality *matching (i.e.,* a set of vertex-disjoint edges) in H. Example 7 illustrates this algorithm. We note that 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. $P2|prec,p_j=1|C_{max}$

instance: n = 6;



optimal schedule:



For any constant $m \geq 3$, the complexity of $Pm|prec,P_j=1|C_{max}$ is an open question. However, $P|prec,P_j=1|C_{max}$ is know to be NP-hard /37/;/30/. 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 /4/;/5/, 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 /33/. 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 /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.

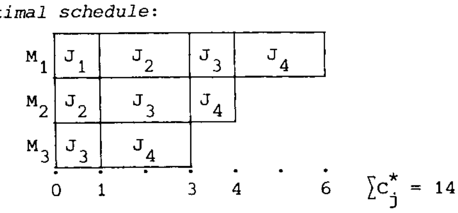
Very little is know about $R|pmtn|\sum C_j$. This is one of the more intriguing open problems in the area of multiprocessor scheduling.

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:



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

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

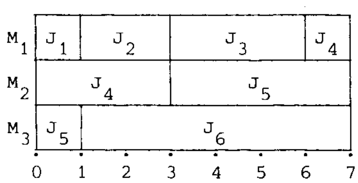
The *wrap-around* rules solve the problem in O(n) time /33/: fill the machines successively, scheduling the jobs in any order and splitting a job whenever the above time bound is met. There will be at most m-1 preemptions. Example 9 illustrates this rule.

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 $O|pmtn|C_{max}$, 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 /15/. It generates at most 2(m-1) preemptions.

$R|pmtn|C_{max}$ can be formulated as a *linear programming* problem /28/. Let

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

Then the problem is to minimize

C_{max}

subjet to

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

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

$P|pmtn,r_j|\ C_j$ and $O|pmtn,r_j|\ C_j$ 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 /26/.

$P|pmtn,r_j|C_{max}$ can be solved by an O(mn) on-line algorithm /20/;/13/. and $O|pmtn,r_j|C_{max}$ by an $O(n^2)$ nearly on-line algorithm /36;/26/.

Finally, we assume that in addition J_j has to be completed not later than a given *due date* d_j (j=1,...,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 /35/. However, off-line algorithms are still available: $P|pmtn,r_j, d_j|$ is solvable by an $O(n^3)$ network flow -- computation /20/, and $Q|pmtn,r_j, d_j|$- by --- means of an $O(n^6)$ "generalized" network flow model /32/.

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 /18/; / 2/. The - development of increasingly sophisticated -- algorithmic techniques combined with a further application of the tools from computational complexitiy theory should continue to render the area of multiprocessor scheduling an interesting one to theoreticians and practitioners alike.

7. ACKNOWLEDGMENT

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

8. REFERENCES

/1/ BELLMAN, R.E. and DREYFUS, S.E.: "Applied Dynamic Programming", Princeton University Press, Princeton, N.J., 1962.

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

/3/ BRUNO, J., 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 - algorithms in multiprocessing computing systems",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.) Parallell 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-- Wesley, Reading, Mass. 1967.

/7/ COOK, S.A.: "The complexity of theorem-proving procedures", Proc. 3rd Annual - ACM Symp. Theory of Computing, 1971, pp. 151-158.

/8/ EDMONDS, J.:"Paths, trees, and flowers. Cand. J. Math., v. 17, 1965, pp. 449-467.

/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 processor scheduling with start-times and - deadlines", SIAM J. Comput. v. 6, 1977, pp. 416-426.

/11/ GAREY, M.R. and JOHNSON, D.S.: "Computers and Intractability : a Guide to the ---- Theory of NP-Completeness", Freeman, San Francisco, 1979.

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

/13/ GONZALEZ, T. and JOHNSON, D.B.: "A new algorithm for preemptive scheduling of - trees", Technical Report 222, Computer Science Department, Pennsylvania State - Unviersity, 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", J. Assoc. Comput. Mach. v. 25,1978, pp. 92-101.

/16/ GRAHAM, R.L.: "Bounds for certain multiprocessing anomalies", Bell System Tech. J. v. 45, 1966, pp. 1536-1581.

/17/ GRAHAM, R.L.: "Bounds on multiprocessing timing anomalies", SIAM J. Appl. Math. v. 17, 1969, pp. 263-269.

/18/ GRAHAM, R.L., LAWLER, E.L., LENSTRA, J. K. and RINNOOY KAN, A.H.G.: "Optimization and approximation in deterministic sequencing and scheduling: a survey", Ann. Discrete Math. v. 5, 1979, pp. 287-326.

/19/ HORN, W.A.: "Minimizing average flow time with parallell machines", Operations Res. v. 21, 1973, pp. 846-847.

/20/ HORN, W.A.: "Some simple scheduling algorithms", 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. Comput. Mach. v. 23, 1976, pp. 317-327.

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

/23/ KARP, R.M.: "Reducibility among combinatorial 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 complexity of combinatorial problems", Networks 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/ LABETOULLE, J., LAWLER, E.L., LENSTRA, J.K. and RINNOOY KAN, A.H.G.: "Prremptive scheduling of uniform machines subject to release dates", Report BW 99, Mathematisch Centrum, Amsterdam, 1979,

/27/ LAWLER, E.L.: "Combinatorial Optimization: Network and Matroids", Holt, Rinehart and Winston, New York, 1976.

/28/ LAWLER, E.L. and LABETOULLE, J.: "On preemptive scheduling of unrelated parallel processor by linear programming", J. Assoc. Comput. Mach. v. 25, 1978, pp. 612-619.

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

/30/ LENSTRA, J.K. and RINNOOY KAN, A.H.G.: "Complexity of scheduling under precedence constraints", Operations Res. v. 26, 1978, pp. 22-35.

/31/ LENSTRA, J.K.and RINNOOY 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/ McNAUGHTON, R.: "Scheduling with deadlines and loss functions", Management Sci. v. 6, 1959, pp. 1-12.

/34/ ROTHKOPF, M.H.: "Scheduling independent tasks on parallel processors", Management Sci. v. 12, 1966, pp. 437-447.

/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.