Recent Results in the Theory of Machine Scheduling

E. L. Lawler
University of California, College of Engineering, Computer Science Division, 591 Evans Hall, Berkeley, CA 94720, USA

Abstract. The state of the art of deterministic machine scheduling is reviewed. Emphasis is placed on efficient, i.e. polynomial-bounded, optimization algorithms. A few of the more significant NP-hardness results are highlighted, and some open problems are mentioned.

I. Introduction

The theory of sequencing and scheduling encompasses a bewilderingly large variety of problem types. In the present paper, I shall not attempt to survey more than a small part of this active and rapidly growing area of research. In particular, I shall confine my attention to *machine* scheduling problems, excluding from consideration such worthy topics as project scheduling, timetabling, and cyclic scheduling of manpower. I shall concentrate on strictly *deterministic* models, with emphasis on *efficient*, i.e. polynomial-bounded, algorithms for *optimization*, as opposed to approximation. I shall mention certain significant NP-completeness results and point out several open problems.

This selection of topics reflects my own interests in the field, which I have pursued since the early 1960's but most vigorously since 1974 when I began working with Jan Karel Lenstra and Alexander Rinnooy Kan. One of the objectives of our collaboration has been to delineate, as closely as possible, the boundary between those machine scheduling problems which are *easy* (solvable in polynomial time), and those which are *NP-hard*. We have produced two surveys (Graham, et al, 1977), (Lawler, Lenstra & Rinnooy Kan, 1982 A), and a detailed tabulation of the status of problem types (Lageweg, et al, 1981). The reader is referred to (Lawler, Lenstra & Rinnooy Kan, 1982 B), for an anecdotal account of our collaboration, which will eventually result in a book.

The present paper differs from the surveys mentioned above in that I have chosen to emphasize only those algorithms which I believe are most interesting, elegant or important. Moreover, I have tried to provide something of a historical perspective, in that for each type of problem considered, I first try to state a "classical" algorithm (usually meaning one obtained prior to 1960) and then show how this algorithm has been generalized or improved (or how generalization has been blocked by NP-hardness). There are four principal parts to the paper dealing respectively with single machines, identical and uniform parallel machines, open shops and unrelated parallel machines, and flow shops and job shops.

It greatly facilitates any discussion of machine scheduling to have an appropriate notation for problem types. Such a notation is detailed in the references mentioned above, and is of the form $\alpha|\beta|\gamma$, where α indicates *machine environment* (single machine, parallel machine, open shop, flow shop, job shop), β indicates *job characteristics* (independent vs.precedence constrained, etc.), and γ indicates the *optimality criterion* (makespan, flow time, maximum lateness, total tardiness, etc.). Instead of defining this notation at the outset, I shall introduce it a bit at a time, and hope that the reader will find this natural and not distracting.

II. Single-Machine Scheduling

1. Minimizing Maximum Cost

Consider the following simple sequencing problem. There are n jobs to be processed by a *single machine* which can execute at most one job at a time. Each job j requires a *processing time* p_j and has a specified *due date* d_j. If the jobs are executed without interruption and without idle time between them, with the first job beginning at time $t=0$, then any given sequence induces a well defined *completion time* C_j and *lateness* $L_j = C_j - d_j$ for each job j. What sequence will minimize maximum lateness? That is, minimize

$$L_{\max} = \max_j \{L_j\}.$$

The "earliest due date" or EDD rule of (Jackson, 1955) provides a simple and elegant solution to this problem: Any sequence is optimal that puts the jobs in order of nondecreasing due dates. This result can be proved by a simple interchange argument. Let π be any sequence and π^* be an EDD sequence. If $\pi \neq \pi^*$ then there exist two jobs j and k such that j immediately precedes k in π, but k precedes j in π^*. Since $d_k \leqslant d_j$, interchanging the positions of j and k in π cannot increase the value of $L_{\max}$. A finite number of such transpositions transforms π to π^*, showing that π^* is optimal.

One generalization of this problem is to allow a monotone nondecreasing *cost function* f_j to be specified for each job j, and to attempt to minimize $f_{\max}$, where

$$f_{\max} = \max_j \{f_j(C_j)\}.$$

Another generalization is to allow *precedence constraints* to be specified in the form of a partial order $\rightarrow$; if $j \rightarrow k$ then job j has to be completed before job k can start. (In the original problem, jobs were *independent*, i.e. the relation $\rightarrow$ was assumed to be empty.)

We denote the problem solved by Jackson by $1||L_{\max}$ and the generalized problem by $1|prec|f_{\max}$ ("1" for single machine; "*prec*" for arbitrary precendence constraints; $f_{\max}$ for the objective function). The problem $1|prec|f_{\max}$ is solved by the following simpe rule (Lawler, 1973): From among all jobs that are eligible to be sequenced last, i.e. that have no successors under $\rightarrow$, put that job last which will incur the smallest cost in that position. Then repeat on the set of $n-1$ jobs remaining, and so on. (For $1||L_{\max}$, this rule says, "Put that job last which has largest due date. Then repeat.")

The correctness of this rule is proved as follows. Let $N=\{1,\dots,n\}$ be the set of all jobs, let $L\subseteq N$ be the set of jobs without successors, and for any $S\subseteq N$ let $f^*(S)$ be the maximum job completion cost in an optimal schedule for S. We know that the last job of any sequence is completed at time $P=p_1+p_2+\dots+p_n$. If job $l\in L$ is chosen such that

$$f_l(P)=\min_{j\in L}\{f_j(P)\},$$

then the optimal value of a sequence subject to the condition that job l is processed last is given by

$$\max\{f^*(N-\{l\}),\ f_l(P)\}.$$

Since both $f^*(N-\{l\})\leqslant f^*(N)$ and $f_l(P)\leqslant f^*(N)$, the rule is proved.

The algorithm can be implemented to run in $O(n^2)$ time, under the assumption that each f_j can be evaluated in unit time for any value of the argument. This contrasts with $O(n\log n)$ time for sorting due dates under Jackson's rule.

A further natural generalization is obtained by specifying a *release time* r_j for each job j, prior to which the job cannot be performed. (Prior to this we have assumed each job is available at time zero, i.e. $r_j=0$ for all j.) If it is required that jobs be executed without interruption, then the introduction of release dates makes things quite difficult. It is an NP-complete problem even to determine if a set of independent jobs can be completed by specified due dates. However, if *preemption* is permitted, i.e. the processing of any job may arbitrarily often be interrupted and resumed at a later time without penalty, then the problem is much easier, and the procedure of (Lawler, 1973) has been further generalized in (Baker, et al, 1982) to apply to the problem $1|pmtn, prec, r_j|f_{\max}$ ("pmtn" for preemption, "r_j" for release dates). For brevity we do not describe this algorithm here, but mention that it can be implemented to run in $O(n^2)$ time.

We should mention that although the nonpreemptive problem $1|r_j|L_{\max}$ is NP-hard, the special case of unit-time jobs, $1|r_j, p_j=1|L_{\max}$ is easy. Moreover, it remains easy even if release dates and due dates are not integers (Simons, 1978).

2. Smith's Rule and Interchange Arguments

Now suppose that we have a single-machine scheduling problem in which each job has a specified *processing time* p_j and *weight* w_j, and the objective is to find a sequence minimizing the *weighted sum of job completion times*, $\Sigma w_j C_j$. This problem, $1||\Sigma w_j C_j$ in our notation, is solved by another classical result of scheduling theory, the "ratio rule" of (Smith, 1956): Any sequence is optimal that puts the jobs in order of nondecreasing ratios $\rho_j=p_j/w_j$. As a corollary, if all jobs have equal weight, any sequence is optimal which places the jobs in nondecreasing order of processing times. (This is known as the "shortest processing time"or SPT rule.)

Let us pose a very general type of sequencing problem that includes $1||L_{\max}$ and $1||\Sigma w_j C_j$ as special cases. Given a set of n jobs and a real-valued function f which assigns a value $f(\pi)$ to each permutation π of the jobs, find a permutation π^* such that

$$f(\pi^*) = \min_{\pi} \{f(\pi)\}.$$

If we know nothing of the structure of the function f, there is clearly nothing to be done except to evaluate $f(\pi)$ for each of the $n!$ permutations π. However, we may be able to find a transitive and complete relation $\leqslant$ (i.e. a quasi-total order) on the jobs with the property that for any two jobs b, c and any permutation of the form $\alpha b c \delta$ we have

$$b \leqslant c \Rightarrow f(\alpha b c \delta) \leqslant f(\alpha b c \delta).$$

Such a relation is called a *job interchange relation*. It says that whenever b and c occur as adjacent jobs with c before b, we are at least as well off to interchange their order. Hence, this relation is sometimes referred to as the "adjacent pairwise interchange property." It is a simple matter to verify the following.

Theorem 1. If f admits of a job interchange relation $\leqslant$, then an optimal permutation π^* can be found by ordering the jobs according to $\leqslant$, with $O(n \log n)$ comparisons of jobs with respect to $\leqslant$.

Note that both Jackson's rule and Smith's rule are based on job interchange relations. Job interchange realtions have been found for a number of other sequencing problems. There is such a relation for the weighted sum of *discounted* job completion times (Rothkopf, 1966), for the "least cost testing sequence" problem (Garey, 1973), and for the two-machine flow shop problem (Johnson, 1954; Mitten, 1958). More will be said about such problems in the next section in the context of precedence constraints.

Let us now consider the generalization of $1||\Sigma w_j C_j$ to allow release dates and deadlines. It turns out that $1|r_j|\Sigma C_j$ (and *a fortiori*, $1|r_j|\Sigma w_j C_j$) is NP-hard (Lenstra, 1977), as is $1|pmtn, r_j|\Sigma w_j C_j$ (Labetoulle, et al, 1979). However, $1|pmtn, r_j|\Sigma C_j$ admits of a very simple solution (Baker, 1974): Schedule over time, starting at the first release date. At each decision point (whenever a job is released or a job is completed) choose to process next from among the available jobs (those whose release dates have been met and for which processing is not yet complete) a job whose remaining processing time is minimal. Only $O(n \log n)$ time is required to construct an optimal schedule in this way.

A *deadline* $\bar{d}_j$ (as opposed to a due date, which is used for computing the cost of a schedule) imposes a constraint that $C_j \leqslant \bar{d}_j$. In (Smith, 1956), a solution is offered to the problem $1|\bar{d}_j|\Sigma C_j$: From among all jobs that are eligible to be sequenced last, i.e. are such that $\bar{d}_j \geqslant p_1 + \ldots + p_n$, put that job last which has the largest possible processing time. Then repeat on the set of $n-1$ jobs remaining, and so on. This rule yields an $O(n \log n)$ algorithm. (It is left to the reader to verify that if there exists any sequence in which all jobs meet their deadlines, then Smith's algorithm produces such a sequence.)

Although $1|\bar{d}_j|\Sigma C_j$ is easy, $1|\bar{d}_j|\Sigma w_j C_j$ is NP-hard (Lenstra, 1977). (An incorrect algorithm is proposed in (Smith, 1956).) It is easily shown that there is no advantage to preemption, in that any solution that is optimal for $1|\bar{d}_j|\Sigma w_j C_j$ is optimal for $1|pmtn, \bar{d}_j|\Sigma w_j C_j$. It follows that $1|pmtn, \bar{d}_j|\Sigma C_j$ is easy and $1|pmtn, \bar{d}_j|\Sigma w_j C_j$ is NP-hard.

The cited results leave us with one unresolved issue in this area.

Open Problem: What is the status of $1|pmtn, r_j, \bar{d}_j|\Sigma C_j$? Easy or NP-hard?

3. Series Parallel Scheduling

Over a period of years, investigators considered the effect of precedence constraints on the $1||\Sigma w_j C_j$ problem. In (Conway, et al, 1967) precedence constraints in the form of parallel chains were dealt with, but only subject to the condition that all $w_j = 1$. In (Horn, 1972), an $O(n^2)$ algorithm was proposed for precedence constraints in the form of rooted trees. In (Adolphson & Hu, 1973) an $O(n \log n)$ algorithm was proposed for the same case. In (Lawler, 1978 A), an $O(n \log n)$ algorithm was given for "series parallel" precedence constraints (with rooted trees as a special case), and the problem was shown to be NP-hard for arbitrary precedence constraints, even if all $w_j = 1$ or if all $p_j = 1$. Other contributions to the problem were made by (Sidney, 1975) and (Adolphson, 1977).

The concept of series parallelism, as introduced in (Lawler, 1978 A), may be unfamiliar to the reader. A digraph is said to be *series-parallel* if its transitive closure is *transitive series-parallel*, as given by the recursive definition below:

(1) A digraph $G = (\{j\}, \phi)$ with a single vertex j and no arcs is transitive series-parallel.

(2) Let $G_1 = (V_1, A_1)$, $G_2 = (V_2, A_2)$ be transitive series-parallel digraphs with disjoint vertex sets. Both the *series composition* $G_1 \rightarrow G_2 = (V_1 \cup V_2, A_1 \cup A_2 \cup (V_1 \times V_2))$ and the *parallel composition* $G_1 \| G_2 = (V_1 \cup V_2, A_1 \cup A_2)$ are transitive series-parallel digraphs.

(3) No digraph is transitive series-parallel unless it can be obtained by a finite number of applications of Rules (1) and (2).

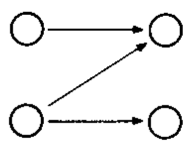


Fig. 1. Z-diagraph

A variety of interesting and useful digraphs (and their corresponding partial orders) are series-parallel. In particular, rooted trees, forests of such trees and level digraphs are series-parallel. The smallest acyclic digraph which is not series-parallel is the *Z-digraph* shown in Figure 1. An acyclic digraph is series-parallel if and only if its transitive closure does not contain the Z-diagraph as an induced subgraph. It is possible to determine whether or not an arbitrary digraph $G=(V,A)$ is series-parallel in $O(|V|+|A|)$ time (Valdes, et al, 1982).

The structure of a series-parallel digraph is displayed by a *decomposition tree* which represents one way in which the transitive closure of the digraph

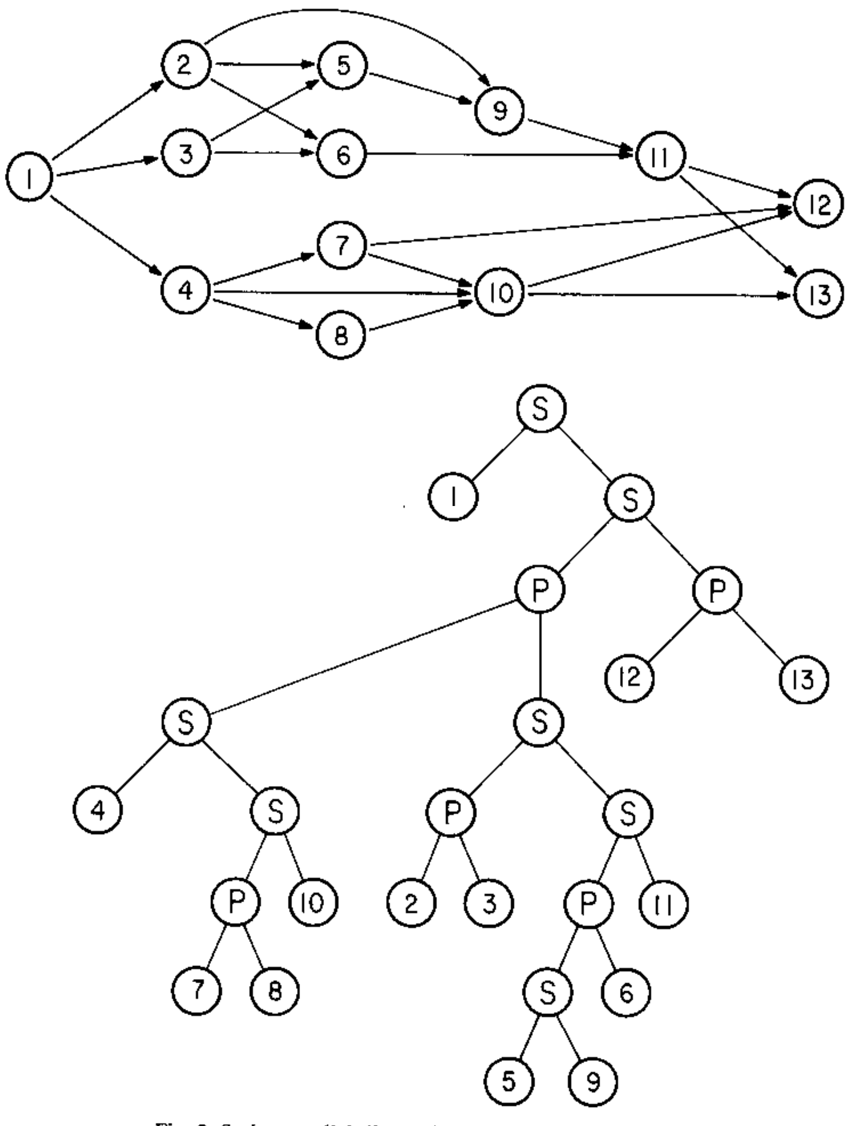


Fig. 2. Series-parallel diagraph and decomposition tree

can be obtained by successive applications of Rules (1) and (2). A series-parallel digraph and its decomposition tree are shown in Figure 2. Each leaf of the decomposition tree is identified with a vertex of the digraph. An S-node represents the application of series composition to the subdigraphs identified with its children; the ordering of these children is important: We adopt the convention that left precedes right. A P-node represents the application of parallel composition to the subdigraphs identified with its children; the ordering of these children is unimportant. The series or parallel relationship of any pair of vertices can be determined by finding their least common ancestor in the decomposition tree.

It was observed by (Monma and Sidney, 1979) that the series-parallel algorithm of (Lawler, 1978 A) has much broader application than just to the weighted completion time problem. We now indicate this by a very general problem formulation, as in (Lawler, 1978 B).

As in the previous section let f be a real-valued function of permutations. But now suppose that precedence constraints are specified in the form of a partial order $\rightarrow$. A permutation π is *feasible* if $j \rightarrow k$ implies that job j precedes job k under π. The objective is to find a feasible permutation π^* such that

$$f(\pi^*) = \min_{\pi \text{ feasible}} \{f(\pi)\}.$$

We need something stronger than a job interchange relation to solve this problem. A transitive and complete relation $\leqslant$ on subpermutations or *strings* of jobs with the property that for any two disjoint strings of jobs β, γ and any permutation of the form $\alpha\beta\gamma\delta$ we have

$$\beta \leqslant \gamma \Rightarrow f(\alpha\beta\gamma\delta) \leqslant f(\alpha\gamma\beta\delta)$$

is called a *string interchange relation*. Smith's rule generalizes to such a relation in a fairly obvious way: For any string α we define $\rho_\alpha = \sum_{j \in \alpha} p_j / \sum_{j \in \alpha} w_j$. However, it is not true that every function f which admits of a job interchange relation also has a string interchange relation.

The remainder of this section is devoted to an intuitive justification of the following result. Details can be found in (Lawler, 1978 B).

Theorem 2. If f admits of a string interchange relation $\leqslant$ and if the precedence constraints $\rightarrow$ are series-parallel, then an optimal permutation π^* can be found by an algorithm which requires $O(n \log n)$ comparisons of strings with respect to $\leqslant$.

Suppose we are to solve a sequencing problem for which a string interchange relation $\leqslant$ exists and a decomposition tree for the series-parallel constraints $\rightarrow$ is given. We can solve the problem by working from the bottom of the tree upward, computing a set of strings of jobs for each node of the tree from the sets of strings obtained for its children. Our objective is to obtain a set of strings at the root node such that sorting these strings according to $\leqslant$ yields an optimal feasible permutation.

We will accomplish our objective if the sets S of strings we obtain satisfy two conditions:

(i) Any ordering of the strings in a set S according to $\leqslant$ is feasible with respect to the precedence constraints $\rightarrow$.
(ii) At any point in the computation, let $S_1, \ldots, S_k$ be the sets of strings computed for nodes such that sets have not yet been computed for their parents. Then some ordering of the strings in $S_1 \cup \ldots \cup S_k$ yields an optimal feasible permutation.

If we order the strings computed at the root according to $\leqslant$, then condition (i) ensures that the resulting permutation is feasible and condition (ii) ensures that it is optimal.

For each leaf of the tree, we let $S = \{j\}$, where j is the job identified with the leaf. Condition (i) is satisfied trivially and condition (ii) is clearly satisfied for the union of the leaf-sets.

Suppose S_1 and S_2 have been obtained for the children of a P-node in the tree. There are no precedence constraints between the strings in S_1 and the strings in S_2. Accordingly, conditions (i) and (ii) remain satisfied if for the P-node we let $S = S_1 \cup S_2$.

Suppose S_1 and S_2 have been obtained for the left child and the right child of an S-node, respectively. Let

$$\sigma_1 = \max_{\leqslant} S_1, \quad \sigma_2 = \min_{\leqslant} S_2.$$

If $\sigma_2 \not\leqslant \sigma_1$, then conditions (i) and (ii) are still satisfied if for the S-node we let $S = S_1 \cup S_2$. If $\sigma_2 \leqslant \sigma_1$, we assert that there exists an optimal feasible permutation in which σ_1 and σ_2 are replaced by their concatenation $\sigma_1\sigma_2$. (The proof of this assertion involves simple interchange arguments; see (Lawler, 1978 B).) This suggests the following procedure:

```
begin
    $\sigma_1 := \max_{\leqslant} S_1$; $\sigma_2 := = \min_{\leqslant} S_2$;
    if $\sigma_2 \not\leqslant \sigma_1$
    then $S := S_1 \cup S_2$
    else $\sigma := \sigma_1\sigma_2$
        $S_1 := S_1 - \{\sigma_1\}$; $\sigma_1 := \max_{\leqslant} S_1$;
        $S_2 := S_2 - \{\sigma_2\}$; $\sigma_2 := \min_{\leqslant} S_2$;
        while $\sigma \leqslant \sigma_1 \vee \sigma_2 \leqslant \sigma$
        do if $\sigma \leqslant \sigma_1$
            then $\sigma := \sigma_1\sigma$;
                 $S_1 := S_1 - \{\sigma_1\}$; $\sigma_1 := \max_{\leqslant} S_1$
            else $\sigma := \sigma\sigma_2$;
                 $S_2 := S_2 - \{\sigma_2\}$; $\sigma_2 := \min_{\leqslant} S_2$
            fi
        od;
        $S := S_1 \cup \{\sigma\} \cup S_2$
    fi
end.
```

(We make here the customary assumption that $\max_{\leqslant} \phi$ and $\min_{\leqslant} \phi$ are very small and large elements, respectively.) It is not difficult to verify that conditions (i) and (ii) remain satisfied if for an S-node we compute a set of strings according to the above procedure.

The entire algorithm can be implemented so as to require $O(n \log n)$ time plus the time for the $O(n \log n)$ comparisons with respect to $\leqslant$.

In addition to the total weighted completion time problem, several other sequencing problems admit of a string interchange relation and hence can be solved efficiently for series-parallel precedence constraints. Among these are the problems of minimizing *total weighted discounted completion time* (Lawler & Sivazlian, 1978), *expected cost of fault detection* (Garey, 1973; Monma & Sidney, 1979), *minimum initial resource requirement* (Abdel-Wahab & Kameda, 1978; Monma & Sidney, 1979), and the *two-machine permutation flow shop problem with time lags* (Sidney, 1979).

4. The Total Tardiness Problem

The *tardiness* of job j with respect to due date d_j is $T_j = \max\{0, C_j - d_j\}$. It is a far more difficult task to minimize total tardiness than to minimize total completion time. (Note that $\Sigma C_j = \Sigma L_j + \Sigma d_j$, where Σd_j is a constant.) Over the years a large number of papers have been written about the problem $1\|\Sigma T_j$, among the most important of which is (Emmons, 1969), in which certain "dominance" conditions were established. These are conditions under which there exists an optimal sequence in which one job j precedes another job k. (A simple case is that in which $p_j \leqslant p_k$ and $d_j \leqslant d_k$.)

In (Lawler, 1977), a "pseudopolynomial" algorithm with running time $O(n^4 \Sigma p_j)$ was described, and a "strong" NP-hardness proof (due to M. R. Garey and D. S. Johnson) was presented for $1\|\Sigma w_j T_j$. This pseudopolynomial algorithm also serves as the basis for a fully polynomial approximation scheme for the problem $1\|\Sigma T_j$ (Lawler, 1982 B).

The existence of a pseudopolynomial algorithm rules out the possibility that the total tardiness problem is NP-hard in the "strong" sense (unless $P = NP$). However, it leaves unresolved the foolowing question.

Open Problem: What is the status of $1\|\Sigma T_j$? Easy or NP-hard (in the "ordinary" sense)?

5. Minimizing the Number of Late Jobs

Another possible objective of single-machine scheduling is to find a sequence minimizing the number of jobs j which are late with respect to specified due dates d_j. We let

$$U_j(C_j) = \begin{cases} 0 & \text{if } C_j \leqslant d_j \\ 1 & \text{if } C_j > d_j \end{cases}$$

and denote this problem $1\|\Sigma U_j$.

The problem $1\|\Sigma U_j$ is clearly equivalent to that of finding a subset $S \subseteq N = \{1, \ldots, n\}$ such that all the jobs in S can be completed on time and $|S|$ is maximal. An optimal sequence then consists of the jobs in S, in EDD order, followed by the jobs in $N-S$ in arbitrary order. (Note that once a job is late it does not matter how late it is.) An elegant algorithm for finding a maximum subset S is given in (Moore, 1968). We state this without proof.

```
begin
    order the jobs so that $d_1 \leqslant \ldots \leqslant d_n$.
    $S := \phi$
    $P := 0$
    for $j := 1$ to $n$
        do $S := S \cup \{j\}$
           $P := P + p_j$
           if $P > d_j$
           then let $p_l = \max\{p_i | i \in S\}$
                    $S := S - \{l\}$
                    $P := P - p_l$
           fi
        od
end.
```

This algorithm can be easily implemented to run in $O(n \log n)$ time.

The weighted version of this problem, $1 \| \Sigma w_j U_j$, is easily shown to be NP-hard, but can be solved in pseudopolynomial time $O(n \max\{d_j\})$ by a dynamic-programming computation similar to that used for the knapsack problem (Lawler & Moore, 1969). However, if job weights are *agreeable*, i.e. there is an ordering of the jobs so that

$$p_1 \leqslant p_2 \leqslant \ldots \leqslant p_n,$$
$$w_1 \geqslant w_2 \geqslant \ldots \geqslant w_n,$$

then the problem can be solved in $O(n \log n)$ time by a simple modification of Moore's algorithm (Lawler, 1976).

If $1 \| \Sigma U_j$ is generalized to allow arbitrary release dates, it immediately becomes NP-hard. If release dates and due dates are *compatible*, i.e. there is an ordering of the jobs so that

$$r_1 \leqslant r_2 \leqslant \ldots \leqslant r_n,$$
$$d_1 \leqslant d_2 \leqslant \ldots \leq d_n,$$

then the problem is easy. An $O(n^2)$ algorithm was proposed in (Kise, et al, 1978). An $O(n \log n)$ algorithm, generalizing Moore's procedure, has been provided for this case in (Lawler, 1982C).

If preemption is permitted, then introducing arbitrary release dates does not cause NP-hardness. In (Lawler, 1982C) a rather complicated dynamic programming procedure for $1|pmtn, r_j|\Sigma w_j U_j$ is presented. This procedure has running time $O(n^3 W^3)$ where $W = \Sigma w_j$. (Job weights are assumed to be integers.) Thus for $1|pmtn, r_j|\Sigma U_j$ the running time becomes $O(n^6)$.

We should also mention the case in which each job has both a due date d_j and a deadline $\bar{d}_j$ (where without loss of generality $d_j \leqslant \bar{d}_j$). The problem $1|\bar{d}_j|\Sigma U_j$ is shown to be NP-hard in (Lawler, 1982C).

6. Minimizing the Sum of Cost Functions

Suppose, as in Section 1, for each job j there is specified a cost function f_j. Except that now we wish to find a sequence that minimizes the sum of the costs, $\sum_j f_j(C_j)$, instead of the maximum of the costs. A general dynamic programming procedure for this problem, which we denote by $1||\Sigma f_j$, was suggested in (Held & Karp, 1962) and (Lawler, 1964):

For any subset $S \subseteq N$, let $p(S) = \sum_{j \in S} p_j$ and let $F(S)$ denote the minimum total cost for a sequence of the jobs in S. Then we have

$$F(\phi) = 0$$
$$F(S) = \min_{j \in S} \{F(S - \{j\}) + f_j(p(S))\}. \tag{1}$$

The optimum value for the complete set of jobs N is $F(N)$ and this value can be obtained in $O(n2^n)$ time.

Note that this solution method does not require that the functions f_j be nondecreasing. It is only necessary to assume that jobs are performed without preemption and without idle time between them.

Interestingly, precedence constraints can only make the dynamic programming computation easier, not harder. In solving $1|prec|\Sigma f_j$, it is necessary to solve equations (1) only for subsets S that are *initial* sets of the partial order $\rightarrow$, in the sense that $j \rightarrow k$, $k \in S$ imply that $j \in S$. Moreover, the minimization in (1) need be carried out only over jobs $j \in S$ that have no successors in S. If the precedence constraints $\rightarrow$ are at all significant, then the number of initial sets is likely to be *very much* smaller than 2^n. One can then hope to solve much larger problems than in the absence of such constraints.

Dominance relations, such as those given by (Emmons, 1969) for the total tardiness problem, can be viewed as precedence constraints. Thus there is real motivation for finding dominance relations for various types of problems. This approach has been taken by (Fisher, et al, 1981) and others for solving the total tardiness problem and other related problems.

It should be noted that it is a nontrivial problem to devise suitable data structures for solving $1|prec|\Sigma f_i$ by dynamic programming. In (Baker, & Schrage, 1978), various "labeling" schemes have been proposed for assigning addresses to initial sets. The difficulty with these schemes is that the address space is not necessarily fully utilized and that memory may be wasted. In (Lawler, 1979), a time- and memory-efficient procedure is proposed for generating the necessary data structures, and tests by (Kao & Queyranne, 1980) seem to confirm its advantages.

There have been many proposals for solving problems of the form $1||\Sigma f_i$ and $1|prec|\Sigma f_i$, by other dynamic programming schemes (Fisher, 1976), and by branch-and-bound, using a variety of bounding methods. Some bounding techniques, are quite sophisticated, as in (Fisher, 1976), where Lagrangean techniques are applied to problem relaxations. The author believes that "hybrid" techniques, combining the best features of branch-and-bound and dynamic programming have potential.

III. Identical and Uniform Parallel Machines

1. Introduction

As before, there are n jobs to be scheduled and job j has a *processing requirement* p_j. But we now suppose there are *m parallel machines* available to do the processing. Each job can be worked on by at most one machine at a time and each machine can work on at most one job at a time. Machine i processes job j with *speed* s_{ij}. Hence if machine i does all the work on job j, it requires a total amount of time p_j/s_{ij} for its processing.

We distinguish three types of parallel machines:

(P) *Identical Machines:* All s_{ij} are equal. In this case we may assume, without loss of generality, that $s_{ij}=1$, for all i,j.

(Q) *Uniform Machines:* $s_{ij}=s_{ik}$, for all i,j,k. In other words, each machine i performs all jobs at the same speed s_i. Without loss of generality, we assume $s_1 \geqslant s_2 \geqslant \ldots \geqslant s_m$.

(R) *Unrelated Machines:* There is no particular relationship between the s_{ij} values.

In the next several sections, we shall consider problems involving identical and uniform machines. Unrelated machines, which require rather different algorithmic techniques, are dealt with in Part IV of this paper.

The difficulty of parallel machine problems is profoundly affected by whether or not preemption is permitted. It is easy to see that even the problem of nonpreemptively scheduling two identical machines so as to minimize makespan, $P2\|C_{\max}$ in out notation, is NP-hard. However, in recent years there have been many new good algorithms developed for preemptive scheduling of parallel machines.

2. Nonpreemptive Scheduling to Minimize the Sum of Completion Times

The solution to the problem $P\|\Sigma C_j$ is quite easy (Conway, et al, 1967): Order the jobs so that $p_1 \leqslant p_2 \leqslant \ldots \leqslant p_n$. Having scheduled jobs $1, 2, \ldots, j$, find the earliest available machine and schedule job $j+1$ on that machine, thereby completing it as soon as possible. This greedy approach, clearly a variant of the SPT rule, yields a schedule like that indicated by the Gannt chart in Figure 3.

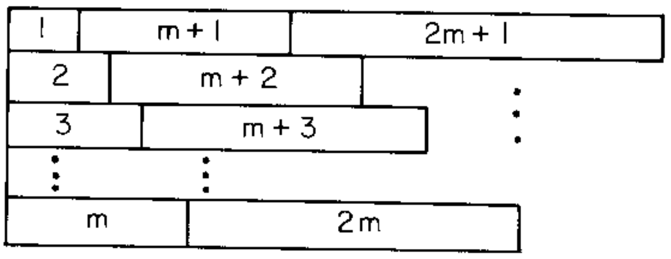


Fig. 3. Gannt chart for SPT schedule

As justification, consider the jobs to be performed on any single machine and for simplicity let these be $1, 2, \ldots, k$ in that order. For these jobs we have

$$\Sigma C_j = kp_1 + (k-1)p_2 + \ldots + 2p_{k-1} + p_k. \tag{2}$$

Considering all m machines, it is apparent that we have available as coefficients m 1's, m 2's, $\ldots$, m n's, and the n smallest of these mn coefficients should be assigned to the n jobs, with the smallest coefficients being assigned to the largest jobs. This is precisely what the SPT rule accomplishes.

Now consider $Q\|\Sigma C_j$, and suppose jobs $1, 2, \ldots, k$ are performed on machine i. Then instead of (2) we have

$$\Sigma C_j = \frac{k}{s_i} p_1 + \frac{(k-1)}{s_i} p_2 + \ldots + \frac{2}{s_i} p_{k-1} + \frac{1}{s_i} p_k.$$

It is now apparent that what we have are mn coefficients of the form k/s_i, $k = 1, 2, \ldots, n$; $i = 1, 2, \ldots, m$ and our task is to assign the n smallest of these coefficients to the n jobs, with the smallest coefficients being assigned to the largest jobs. This can be done in $O(n \log n)$ time with the algorithm of (Howowitz & Sahni, 1976):

```
begin
    order the jobs so that $p_1 \leqslant p_2 \leqslant \ldots \leqslant p_n$
    initialize a priority queue Q with the values $\frac{1}{s_i}$, $i = 1, 2, \ldots, m$.
    for $j = n$ to 1
        do let $\frac{k}{s_i}$ be a smallest value in $Q$
           assign $\frac{k}{s_i}$ as the coefficient of $p_j$
           replace $\frac{k}{s_i}$ in $Q$ by $\frac{k+1}{s_i}$
        od
end
```

Unfortunately, the problem of nonpreemptively scheduling identical parallel machines so as to minimize *weighted* total completion time, i.e. $P\|\Sigma w_j C_j$, is NP-hard.

3. Preemptive Scheduling to Minimize the Sum of Completion Times

The following theorem is proved by rearrangement arguments.

Theorem 3. (McNaughton, 1959). For any instance of $P|pmtn|\Sigma w_j C_j$ there exists a schedule with no preemptions for which the value of $\Sigma w_j C_j$ is as small as that for any schedule with a finite number of preemptions.

(The finiteness restriction can be removed by results implicit in (Lawler & Labetoulle, 1978).)

From McNaughton's theorem it follows immediately that the procedure of the previous section solves $P|pmtn|\Sigma C_j$, since there is no advantage to preemption. Also, from the fact that $P\|\Sigma w_j C_j$ is NP-hard, it follows that $P|pmtn|\Sigma w_j C_j$ (and *a fortiori* $Q|pmtn|\Sigma w_j C_j$) is NP-hard.

A simple example suffices to show that there *is* advantage to preemption in $Q|pmtn|\Sigma C_j$, i.e. the procedure of (Horowitz & Sahni, 1976) does not necessarily yield an optimal solution. We now state a lemma which is also proved by rearrangement arguments.

Lemma 4. (Lawler & Labetoulle, 1978). Let $p_1 \leqslant p_1 \leqslant \dots \leqslant p_n$. Then there is an optimal schedule for $Q|pmtn|\Sigma C_j$ in which $C_1 \leqslant C_2 \leqslant \dots \leqslant C_n$.

This lemma is essential to prove the validity of a solution to $Q|pmtn|\Sigma C_j$ presented in (Gonzalez, 1977). Let $p_1 \leqslant p_2 \leqslant \dots \leqslant p_n$. Having scheduled jobs $1, 2, \dots, j$, schedule job $j+1$ to be completed as easily as possible. This variant of the SPT rule yields a schedule as shown in Figure 4.

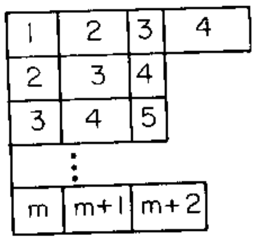


Fig. 4. SPT rule for preemptive scheduling of uniform machines

We comment that the same strategy is optimal for certain stochastic scheduling problems. Cf. (Weiss & Pinedo, 1980).

In (Gonzalez, 1977), a procedure is presented for minimizing total completion time, subject to a common deadline for all jobs. It seems quite feasible to extend the same ideas to the situation in which the deadlines $\bar{d}_j$ and the processing times p_j are *compatible*, i.e. there is a numbering of the jobs so that

$$p_1 \leqslant p_2 \leqslant \dots \leqslant p_n$$
$$\bar{d}_1 \leqslant \bar{d}_2 \leqslant \dots \leqslant \bar{d}_n.$$

However, the status of the problem for arbitrary deadlines is open.

Open Problem: What is the status of $Q|pmtn, \bar{d}_j|\Sigma C_j$? Easy or NP-hard?

Perhaps even more perplexing, almost nothing is known about the case of release dates.

Open Problem: What is the status of $Q|pmtn, r_j|\Sigma C_j$? (Or, for that matter, $P2|pmtn, r_j|\Sigma C_j$?) Easy or NP-hard?

4. Preemptive Scheduling to Minimize Makespan

The problem of preemptively scheduling m identical parallel machines so as to minimize makespan is quite easy. Clearly we must have

$$C_{\max} \geqslant \max\left\{\max_j \{p_j\}, \frac{1}{m}\Sigma p_j\right\}.$$

Yet one can achieve the lower bound given by the right hand side of the inequality by applying the "wraparound" rule of (McNaughton, 1959). Imagine the processing times of the n jobs are laid out end-to-end, in arbitrary order, in a strip. Now cut the strip at intervals whose length is given by the lower bound. Each of the shorter strips becomes a row of the Gannt chart for an optimal schedule, as shown in Figure 5.

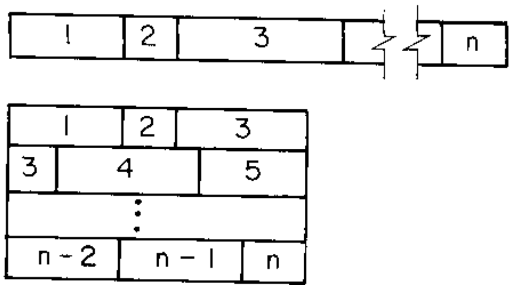


Fig. 5. McNaughton's "wraparound" rule

It is easy to see that this procedure requires $O(n)$ time and creates at most $m-1$ preemptions.

Now let us consider the case of uniform machines. Assuming $s_1 \geqslant s_2 \geqslant \ldots \geqslant s_m$, it is clear that we must have

$$C_{\max} \geqslant \max\left\{\max_{1 \leqslant k \leqslant m-1}\left\{\sum_{j=1}^{k} p_j \Big/ \sum_{i=1}^{k} s_i\right\}, \sum_{j=1}^{n} p_j \Big/ \sum_{i=1}^{m} s_i\right\} \tag{3}$$

It is possible to achieve the lower bound given by the right-hand size of this inequality, and to show this most easily, we now generalize the problem a bit.

Let us suppose that the speeds of the machines are time-varying and that the speed of machine i at time t is $s_i(t)$. Assume $s_1(t) \geqslant s_2(t) \geqslant \ldots \geqslant s_m(t)$, for all t. Define the *capacity* of machine i in the interval $[0, d]$ as

$$S_i = \int_0^d s_i(u)\,du.$$

Assume $p_1 \geqslant p_2 \geqslant \ldots \geqslant p_n$. Then, by the same reasoning behind (3), a necessary condition for there to exist a feasible schedule of length d is that all of the following inequalities be satisfied:

$$\begin{aligned}
&S_1 && \geqslant p_1 \\
&S_1+S_2 && \geqslant p_1+p_2 \\
& && \vdots \\
&S_1+S_2+\ldots S_{m-1} && \geqslant p_1+p_2+\ldots+p_{m-1} \\
&S_1+S_2+\ldots S_{m-1}+S_m && \geqslant p_1+p_2+\ldots+p_{m-1}+\ldots+p_n .
\end{aligned} \tag{4}$$

The inequalities simply assert that, for $k=1,2,\ldots,m-1$, the sum of the capacities of the k fastest machines is at least as large as the sum of the processing requirements of the k largest jobs, and that the sum of the capacities of all m machines is at least as large as the sum of the requirements of all n jobs. Satisfaction of inequalities (4) is also a sufficient condition for the existence of a feasible schedule. Our proof is based on ideas of (Gonzalez & Sahni, 1978).

Let us choose an arbitrary job j to schedule in the interval $[0,d]$. Find the largest index k such that $S_k \geqslant p_j$. If $k=m$, then let machine $m+1$ be a dummy machine with zero speed in the discussion which follows. Let

$$f(t)=\int_0^t s_k(u)\,du, \qquad g(t)=\int_t^d s_{k+1}(u)\,du.$$

From $f(0)=0$, $f(d)\geqslant p_j$, $g(0)<p_j$, $g(d)=0$, and the fact that f and g are continuous functions, it follows that there exists a t', $0<t'\leqslant d$, such that $f(t')+g(t')=p_j$. Accordingly, schedule job j for processing by machine k in the interval $[0,t]$ and by machine $k+1$ in the interval $[t',d]$.

Next replace elementary machines k and $k+1$ by a *composite* machine formed from the remaining available time on those machines. This new machine has speed $s_{k+1}(t)$ in the interval $[0,t']$, speed $s_k(t)$ in $[t',d]$, and total processing capacity $S_k+S_{k+1}-p_j$ in the interval $[0,d]$. If $k<m$, create a dummy machine with zero speed. (We already created such a machine if $k=m$.) This gives us m machines with capacities S'_i as follows. If $k<m$, then

$$S'_i=\begin{cases} S_i & \text{for } i=1,\ldots,k-1 \\ S_k-S_{k+1}-p_j & \text{for } i=k, \\ S_{i+1} & \text{for } i=k+1,\ldots,m-1 \\ 0 & \text{for } i=m. \end{cases}$$

And if $k=m$, then

$$S'_i=\begin{cases} S_i & \text{for } i=1,\ldots,m-1 \\ S_m-p_j & \text{for } i=m. \end{cases}$$

The problem thus reduces to one involving m machines and $n-1$ jobs. The new machine capacities $S'_i, i=1,2,\ldots,m$, and processing requirements $p_1,p_2,\ldots,p_{j-1},p_{j+1},\ldots,p_n$ satisfy inequalities (4). (This can be verified by an analysis of the cases $j\leqslant k<m$, $k<j\leqslant m$, $k=m$, which we omit.) Induction on the number of jobs proves that repeated application of the scheduling rule yields a schedule in which all jobs are completed on time. (It can also be shown that such a schedule requires no more than $2(m-1)$ preemptions.) Satisfaction

of inequalities (4) is thus a sufficient condition for the existence of a feasible schedule and we have proved the following result.

Theorem 5. Satisfaction of inequalities (4) is a necessary and sufficient condition for the existence of a schedule in which all n jobs are completed within the interval $[0, d]$.

5. Preemptive Scheduling with Due Dates

Now suppose the n given jobs have arbitrary due dates. Is it possible to schedule the jobs so that they are are all completed on time? The procedure we shall describe for solving this problem is adapted from (Sahni & Cho, 1980).

Let $d_0 = 0$, assume $d_0 < d_1 \leq d_2 \leq \ldots \leq d_n$ and let S_i denote the processing capacity of machine i in the interval $[0, d_1]$. If there exists a schedule in which all jobs are completed on time, then in any such schedule *all* of the processing requirement of job 1 and some amount $p_j' \leq p_j$ of the requirement of each job $j, j = 2, 3, \ldots, n$, is processed in the interval $[0, d_1]$. The unknown values $p_j', j \neq 2$, together with the known value p_1 (appropriately reordered) satisfy inequalities (4). It follows that, if there exists a feasible schedule, there exists a feasible schedule in which job 1 is scheduled in accordance with the rule presented in the previous section.

So suppose we find the largest index k such that $S_k \geq p_1$, determine the value t', $0 < t' \leq d_1$, and schedule job 1. This leaves us with m machines with processing capacities S_i', $i = 1, 2, \ldots, m$, in the interval $[0, d_1]$. We now also have m (composite) machines with well-defined speeds $s_i'(t)$, in the interval $[0, d_n]$, as follows. If $k < m$, then

$$s_i'(t) = \begin{cases} s_i(t) \quad (0 \leq t \leq d_n) & \text{for } i = 1, \ldots, k-1, \\ \left.\begin{array}{ll} s_{k+1}(t) & (0 \leq t \leq t') \\ s_k(t) & (t' < t \leq d_n) \end{array}\right\} & \text{for } i = k, \\ \left.\begin{array}{ll} s_{i+1}(t) & (0 \leq t \leq d_1) \\ s_i(t) & (d_1 < t \leq d_n) \end{array}\right\} & \text{for } i = k+1, \ldots, m-1 \\ \left.\begin{array}{ll} 0 & (0 \leq t \leq d_1) \\ s_m(t) & (d_1 < t \leq d_n) \end{array}\right\} & \text{for } i = m. \end{cases}$$

And if $k = m$, then

$$s_i'(t) \begin{cases} s_i(t) \quad (0 \leq t \leq d_n) & \text{for } i = 1, \ldots, m-1, \\ \left.\begin{array}{ll} 0 & (0 \leq t \leq t') \\ s_m(t) & (t' < t \leq d_n) \end{array}\right\} & \text{for } i = m. \end{cases}$$

The problem thus reduces to one involving m machines and $n-1$ jobs: There exists a feasible schedule for the n jobs $1, 2, \ldots, n$ on machines with speeds $s_i(t)$, $i = 1, 2, \ldots, m$, if and only if there exists a feasible schedule for the $n-1$ jobs $2, 3, \ldots, n$ on machines with speeds $s_i'(t)$, $i = 1, 2, \ldots, m$.

6. Preemptive Scheduling to Minimize the Number of Late Jobs

The algorithm described in the previous section provides a key to the solution of the problem of minimizing the weighted number of jobs that do not meet their deadlines, i.e. the problem $Q|pmtn\Sigma w_j U_j$. The algorithm is rather complicated to explain and details can be found in (Lawler, 1979). We state only that the running time is $O(W^2 n^2)$ for $m=2$ and $O(W^2 n^{3m-5})$ for $m \leqslant 3$, where $W = \Sigma w_j$.

Note that for *fixed m* the algorithm is pseudopolynomial and that for fixed m and unweighted jobs, i.e. $W=n$, the algorithm is strictly polynomial This is probably the best that we can hope for, since the problem of minimizing the (unweighted) number of jobs is NP-hard, even for *identical* machines, if m is *variable*, i.e. specified as part of the problem instance.

Theorem 6. The problem $P|pmtn|\Sigma U_j$ is NP-hard.

Proof: An instance of the scheduling problem consists of n pairs of positive integers (p_j, d_j), $j=1,2,\dots,n$, representing processing requirements and due dates of n jobs, a positive integer m, representing the number of identical machines, and a positive integer $k \leqslant n$. The question asked is: Does there exist a feasible preemptive schedule in which k (or more) jobs are on time?

We shall describe a polynomial transformation from the known NP-complete problem PARTITION. An instance of this problem consists of t positive integers a_i, $i=1,2,\dots,t$, with $\sum a_i = 2b$. Is there a subset S such that

$$\sum_{i \in S} a_i = \sum_{i \notin S} a_i = b?$$

Given an instance of PARTITION, create an instance of the scheduling problem with $n=5t-2$ jobs as follows:
(1) t jobs with $p_j = b + a_j$, $d_j = b + 2a_j$ $(j=1,\dots,t)$;
(2) t jobs with $p_j = b$, $d_j = b$ $(j=t+1,\dots,2t)$;
(3) $3t-2$ jobs with $p_j = b$, $d_j = 4b$ $(j=2t+1,\dots,5t-2)$.
Let $m=t$ and $k=4t-1$.

If there exists a solution S to PARTITION, then there exists a solution to the scheduling problem. Let $|S|=s$ and construct a set of jobs consisting of the s jobs of type (1) whose indices are in S, any $t-s+1$ jobs of type (2), and all $3t-2$ jobs of type (3). We assert htat there is a feasible schedule for this set of $4t-1$ jobs, by the following argument.

Each of the s jobs of type (1) and $t-s$ of the jobs of type (2) can be scheduled on a separate machine. Each type (2) job is processed continuously in the interval $[0,b]$. Each type (1) job j is processed continuously in the interval $[b, b+2a_j]$, leaving a_j units of idle time on its machine in the interval $[0,b]$. The idle time remaining in $[0,b]$ can be distributed so that the $(t+1)st$ type (2) job can be scheduled. For example, if $S=\{1,2,3\}$, with $a_1+a_2+a_3=b$, the last type (2) job can be scheduled on one machine in the interval $[0,a_1]$, on a second machine in the interval $[a_1, a_1+a_2]$, and on a third machine in the interval $[a_1+a_2, b]$. After the $t+1$ jobs of type (1) and type (2) are scheduled, a total of

$(3t-2)b$ units of idle time remain on the t machines in the interval $[b, 4b]$, with at least $3b - 2\max_j\{a_j\} \geqslant b$ units of idle time on each machine. The $3t-2$ jobs of type (3) can be scheduled in "wrap-around" fashion in the idle time in the interval $[b, 4b]$.

Conversely, if there exists a solution to the scheduling problem, then there exists a solution to PARTITION, by the following argument. Suppose there exists a feasible set of size $4t-1$. Type (3) jobs have later due dates and no greater processing requirements than jobs of type (1) or (2). Hence by a simple substitution argument we may assume, without loss of generality, that the feasible set contains all $3t-2$ of the type (3) jobs. Let S denote the set of indices of the type (1) jobs. There are s, where $s=|S|$, type (1) jobs and $t-s+1$ type (2) jobs. The type (1) jobs require at least $sb - \sum_{j \in S} a_j$ units of processing in the interval $[0, b]$. It follows that the type (2) jobs can be scheduled only if $\sum_{j \in S} a_j \geqslant b$. The total amount of processing required by the type (1) and type (2) jobs is $(t+1)b + \sum_{j \in S} a_j$, which leaves $3(t-1)b - \sum_{j \in S} a_j$ units in the interval $[0, 4b]$ for the type (3) jobs. It follows that the type (3) jobs can be scheduled only if $\sum_{j \in S} a_j \leqslant b$ and we have $\sum_{j \in S} a_j = b$.

This NP-hardness proof differs significantly from previous proofs of NP-hardness for other preemptive scheduling problems. Virtually all other such proofs have proceeded by first showing that there is "no advantage" to preemption for the problem at hand, or for a restricted set of instances of the problem. That is, if there exists a feasible preemptive schedule there must also exist a feasible nonpreemptive schedule. The proof then proceeds by showing that the corresponding nonpreemptive scheduling problem (or an appropriately restricted set of instances of it) is NP-hard. We did not do this. And, in fact, there *is* advantage to preemption in our problem and the nonpreemptive version is well known to be NP-hard even if the number of identical machines is fixed at two.

7. A "Nearly On-Line" Algorithm for Preemptive Scheduling with Release Dates

The algorithm of Section 5 can be applied to solve the problem $Q|pmtn, r_j|C_{\max}$. (Simply turn the problem around and deadlines assume the role of release dates.) However, the algorithm is then "off line"; full knowledge of all problem parameters must be known in advance of computation. It can be shown that there is no fully "on-line" algorithm for the problem. However a "nearly" on-line algorithm is possible. By this we mean that at each release date r_j, the next release date r_{j+1} is known. However, there need be no knowledge of the processing times $p_{j+1}, \ldots, p_n$ nor the release dates $r_{j+2}, \ldots, r_n$.

What we want to accomplish in this. At time r_n we want the remaining processing requirements of jobs $1, 2, \ldots, n-1$, together with the processing requirement of job n, to be such that the value of $C_{\max}$ can be minimized. This will clearly be the case if the k longest of these requirements, for $k=1, 2, \ldots, m-1$, and for $k=n$, is as small as possible. The interesting fact is that this objective can be achieved at each release date $r_j, j=1, 2, \ldots, n$.

So the strategy is as follows. Having determined the schedule up to time r_j, we then determine the schedule for the interval $[r_j, r_{j+1}]$ so that the remaining processing requirements of jobs $1, 2, \ldots, j$ at time r_{j+1} are as "uniform" as possible. Without going into details, we simply indicate that at time r_j we have a "staircase" of remaining processing times, from longest to shortest, as indicated in Figure 6. We then choose to process an amount of each job in the interval $[r_j, r_{j+1}]$, as indicated by the shaded region in the figure. These amounts are chosen to satisfy the requirement that the sum of the k longest remaining processing requirements be as small as possible, for all k.

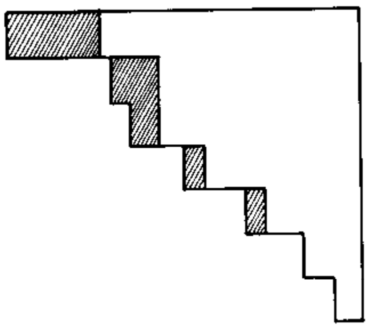


Fig. 6. "Staircase" of remaining processing times

Details may be found in (Sahni & Cho, 1979; and Labetoulle, et al, 1979). The nearly on-line algorithm requires $O(n \log n + mn)$ time. At most $O(mn)$ preemptions are introduced into the schedule.

8. Preemptive Scheduling with Release Dates and Due Dates

Consider the problem of preemptively scheduling jobs on identical parallel machines subject to release dates and deadlines. This was given a network flow formulation in (Horn, 1974).

Let $E=\{r_j\} \cup \{\bar{d}_j\}$ and order the numbers in E. This yields at most $2n-1$ time intervals $[e_h, e_{h+1}]$. Now create a flow network with a node for each job j, a node for each time interval $[e_h, e_{h+1}]$, a dummy source s and a dummy sink t. There is an arc (s, j) from the source to each job node j, and the capacity of this arc is p_j. There is an arc from job node j to the node for interval $[e_h, e_{h+1}]$ if and only if job j can be processed in the interval, i.e. $r_j \leqslant e_h$ and $e_{h+1} \leqslant \bar{d}_j$, and the capacity of this arc is $e_{h+1}-e_h$. There is an arc from each interval node $[e_h, e_{h+1}]$ to the sink, and the capacity of this arc is $m(e_{h+1}-e_h)$. We assert that there exists a feasible preemptive schedule if and only if there exists a flow of value $P = \Sigma p_j$ in this network.

In the case of uniform parallel machines, it is not possible to give such a simple formulation to the problem. What are needed are capacity constraints on *sets* of arcs into a given interval node $[e_h, e_{h+1}]$. In particular, it is necessary that the sum of the k largest flows not exceed $\left(\sum_{i=1}^{k} s_i\right)(e_{h+1} + e_h)$ for $k = 1, 2, \ldots, m-1$, and the arc to the sink is given capacity $\left(\sum_{i=1}^{m} s_i\right)(e_{h+1} - e_h)$. What is interesting is that these capacity constraints are *submodular* and the network is an example of a so-called "polymatroidal" flow network. See (Martel, 1981) for a solution to the specific scheduling problem and (Lawler & Martel, 1982 A, B) for a discussion of polymatroidal network flows in general.

9. Nonpreemptive Scheduling of Unit-Time Jobs Subject to Precedence Constraints

Suppose that n unit-time jobs are to be processed on m identical parallel machines. Each job can be assigned to any machine, and the schedule has to respect given precedence constraints. The objective is to find a schedule which minimizes the maximum of job completion times.

This problem is NP-hard in general, but it can be solved in polynomial time if either the precedence constraints are in the form of a rooted tree or if there are only two machines.

First, let us assume that the precedence constraints are in the form of an *intree*, i.e. each job has exactly one immediate succesor, except for one job which has no successors and which is called the *root*. It is possible to minimize the maximum completion time in $O(n)$ time by applying an algorithm due to Hu (Hu, 1961). The *level* of a job is defined as the number of jobs in the unique path to the root. At the beginning of each time unit, as many available jobs as possible are scheduled on the m machines, where highest priority is granted to the jobs with the largest levels. Thus, Hu's algorithm is a *list scheduling* algorithm, whereby at each step the available job with the highest ranking on a priority list is assigned to the first machine that becomes available. It can also be viewed as a *critical path scheduling* algorithm: the next job chosen is the one which heads the longest current chain of unexecuted jobs.

To validate Hu's algorithm, we will show that, if it yields a schedule of length t^*, then no feasible schedule of length $t < t^*$ exists.

Choose any $t < t^*$ and define a label for each job by subtracting its level from t; note that the root has label t and that each other job has a label one less than its immediate sucessor. The algorithm gives priority to the jobs with the smallest labels. Since it yields a schedule of length larger than t, in some unit-time interval s a job is scheduled with a label smaller than s. Let s be the earliest such interval and let there be a job with label $l < s$ scheduled in it. We claim that there are m jobs scheduled in each earlier interval $s' < s$. Suppose there is an interval $s' < s$ with fewer than m jobs scheduled. If $s' = s - 1$, then the only reason that the job with label l was not scheduled in s' could have been that an immediate predecessor would have label $l-1<s-1$; which contradicts the definition of s. If $s'<s-1$, then there are fewer jobs scheduled in s' than in $s'+1$, which is impossible from the structure of the intree. Hence, each interval $s'<s$ has m jobs scheduled. Since each of these jobs has a label smaller than s, at least one job with a label smaller than s must be scheduled in interval s, so that there is no feasible schedule of length $t<t^*$ possible. This completes the correctness proof of Hu's algorithm.

An alternative linear-time algorithm for this problem has been proposed by Davida and Linton (Davida & Linton, 1976). Assume that the precedence constraints are in the form of an *outtree*, i.e. each job has at most one immediate predecessor. The *weight* of a job is defined as the total number of its successors. The jobs are now scheduled according to decreasing weights.

If the problem is to minimize the *maximum lateness* with respect to given due dates rather than the maximum completion time, then the case that the precedence constraints are in the form of an *intree* can be solved by an adaptation of Hu's algorithm, but the case of an *outtree* turns out to be NP-hard (Brucker, et al, 1977). Polynomial-time algorithms and NP-hardness results for the maximum completion time problem with various other special types of precedence constraints are reported in (Dolev, 1981; Garey, et al, 1981; Warmuth, 1980).

Next, let us assume that there are *arbitrary precedence constraints* but only *two machines*. It is possible to minimize the maximum completion time by a variety of algorithms.

The earliest and simplest approach is due to Fujii, Kasami and Ninomiya (Fujii, et al, 1969, 1971). A graph is constructed with vertices corresponding to jobs and edges $\{j,k\}$ whenever jobs j and k can be executed simultaneously, i.e. $j \not\to k$ and $k \not\to j$. A *maximum cardinality matching* in this graph, i.e. a maximum number of disjoint edges, is then used to derive an optimal schedule; if the matching contains c pairs of jobs, the schedule has length $n-c$. Such a matching can be found in $O(n^3)$ time.

A completely different approach (Coffman & Graham, 1972) leads to a *list scheduling* algorithm. The jobs are labeled in the following way. Suppose labels $1,\dots,l$ have been applied and S is the subset of unlabeled jobs all of whose successors have been labeled. Then a job in S is given the label $l+1$ if the labels of its immediate successors are *lexicographically minimal* with respect to all jobs in S. The priority list is formed by ordering the jobs according to decreasing labels. This method requires $O(n^2)$ time.

Recently, an even more efficient algorithm has been developed by Gabow (Gabow, 1980). His method uses labels, but with a number of rather sophisticated embellishments. The running time can be made strictly linear in $n+a$, where a is the number of arcs in the precedence graph (Gabow, 1982B).

If the problem is to find a *feasible* two-machine schedule under arbitrary precedence constraints when each job becomes available at a given integer *release date* and has to meet a given integer *deadline*, polynomial-time algorithms exist (Garey & Johnson, 1976, 1977). These algorithms can be applied to minimize *maximum lateness* in polynomial time.

It was shown in (Ullman, 1975) that the problem of minimizing maximum completion time for *n* unit-time jobs on *m* identical parallel machines is NP-hard. (Cf. (Lenstra & Rinnooy Kan, 1978) or (Lawler & Lenstra, 1982) for a simpler proof.) However, this NP-completeness proof requires that the number of machines *m* be specified as part of the problem instance. The status of the problem for any *fixed* number of machines, in particular three, is open.

Open Problem: What is the status of the problem $P3|prec, p_j = 1|C_{\max}$? (Scheduling unit-time jobs on three identical parallel machines subject to arbitrary precedence constraints, so as to minimize maximum completion time.) Easy or NP-hard?

10. Preemptive Scheduling Subject to Precedence Constraints

In the previous section we discussed problems involving the *nonpreemptive* scheduling of *unit-time* jobs. There has been a parallel investigation of problems concerning *preemptive* scheduling of jobs of *arbitrary length*. Interesting, there is a preemptive counterparts for virtually all of the unit-time scheduling algorithms. In particular, in (Gonzalez & Johnson, 1980), a preemptive algorithm is given which is analogous to that of (Davida & Linton, 1976). In (Lawler, 1982A) algorithms are proposed that are the preemptive counterparts of those found in (Brucker, et al, 1977) and (Garey & Johnson, 1976, 1977). These preemptive scheduling algorithms employ essentially the same techniques for dealing with precedence constraints as the corresponding algorithms for unit-time jobs. However, they are considerably more complex, and we shall not deal with them here.

IV. Open Shops and Unrelated Parallel Machines

1. Introduction

Problems involving unrelated parallel machines have been solved primarily by linear programming techniques, the validity of which depends on results from the theory of open shops. Moreover, it turns out that it is possible to solve scheduling problems for models that generalize both open shops and unrelated parallel machine shops (Lawler, Luby & Vazirani, 1982). Accordingly, we shall consider these two types of shops together.

2. Open Shops

In an *open shop* there are *n jobs*, to be scheduled for processing by *m machines*. Each machine is to work on job j for a total *processing time* p_{ij}. A machine can work on only one job at a time and a job can be worked on by only one machine at a time. There is no restriction on the order in which a given job can be worked on by the different machines or on the order in which a machine can work on different jobs. (Hence the term "open shop".)

We let the letter "O" denote an open shop. It is easily established that $O\|C_{\max}$ is NP-hard. However, the two-machine problem, $O2\|C_{\max}$, can be solved in $O(n)$ time. Moreover, it can be established that there is no advantage to preemption in the two-machine case, so the linear time algorithm of (Gonzalez & Sahni, 1978) for $O2\|C_{\max}$ solves $O2\|pmtn|C_{\max}$ as well.

We shall now describe the algorithm of (Gonzalez & Sahni, 1978) for $O|pmtn|C_{\max}$. The description is adapted from (Lawler & Labetoulle, 1978).

As noted, an instance of the $O|pmtn|C_{\max}$ problem is defined by an $m \times n$ *matrix* $P=(p_{ij})$. It is evident that a lower bound on the length of a feasible schedule is given by

$$C_{\max}=\max\left\{\max_i\left\{\sum_j p_{ij}\right\},\ \max_j\left\{\sum_i p_{ij}\right\}\right\}. \tag{5}$$

We shall show how to construct a feasible schedule with this value of $C_{\max}$. Since no schedule can be shorter, the constructed schedule will clearly be optimal.

Let us call row i (column j) of matrix P *tight* if $\sum_j p_{ij}=C_{\max}\left(\sum_i p_{ij}=C_{\max}\right)$, and *slack* otherwise. Suppose we are able to find a subset of strictly positive elements of P, with exactly one element of the subset in each tight row and in each tight column and no more than one element in any slack row or column. We shall call such a subset a *decrementing set*, and use it to construct a *partial schedule* of length δ, for some suitably chosen $\delta>0$. In this partial schedule processor i works on job j for $\min\{p_{ij},\delta\}$ units of time, for each element p_{ij} in the decrementing set. We then replace p_{ij} by $\max\{0,p_{ij}-\delta\}$, for each element in the decrementing set, thereby obtaining a new matrix P', for which $C'_{\max}=C_{\max}-\delta$ satisfies condition (5).

For example, suppose $C_{\max}=11$ and

$$P=\begin{array}{cc}\begin{pmatrix}3 & \textcircled{4} & 0 & 4\\ \textcircled{4} & 0 & 6 & 0\\ 4 & 0 & 0 & \textcircled{6}\end{pmatrix} & \begin{array}{c}11\\10,\\10\end{array}\\ \begin{array}{cccc}11 & 4 & 6 & 10\end{array} & \end{array}$$

with row and column sums as indicated on the margins of the matrix. One possible decrementing set is indicated by the encircled elements. Choosing $\delta=4$, we obtain $C'_{\max}=7$ and P' as shown below, with the partial schedule indicated to the right:

$$P'=\begin{array}{ccc}\begin{pmatrix}\textcircled{3} & 0 & 0 & 4\\ 0 & 0 & \textcircled{6} & 0\\ 4 & 0 & 0 & \textcircled{2}\end{pmatrix} & \begin{array}{c}7\\6\\6\end{array} & \begin{array}{|c|}\hline 2\\ \hline 1\\ \hline 4\\ \hline\end{array}\\ \begin{array}{cccc}7 & 0 & 6 & 6\end{array} & & 4\end{array}$$

A decrementing set of P' is indicated by the encircled elements.

There are various constraints that must be satisfied by δ, in order for $C'_{\max} = C_{\max} - \delta$ to satisfy condition (5) with respect to P'. First, if p_{ij} is an element of the decrementing set in a tight row or column, then clearly it is necessary that $\delta \leqslant p_{ij}$, else there will be a row or column sum of P which is strictly greater than $C_{\max} - \delta$. Similarly, if p_{ij} is an element of the decrementing set in a slack row (slack column), then it is necessary that

$$\delta \leqslant p_{ij} + C_{\max} - \sum_k p_{ik} \qquad \left(\delta \leqslant p_{ij} + C_{\max} - \sum_k p_{kj}\right).$$

And if row i (column j) contains no element of the decrementing set (and is therefore necessarily slack), it is necessary that

$$\delta \leqslant C_{\max} - \sum_j p_{ij} \qquad \left(\delta \leqslant C_{\max} - \sum_i p_{ij}\right).$$

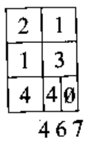
Thus for the example above we have

$$\delta \leqslant p_{12} = 4, \qquad \delta \leqslant p_{21} = 4,$$
$$\delta \leqslant p_{34} + C_{\max} - \sum_k p_{3k} = 7, \qquad \delta \leqslant t_{34} + C_{\max} - \sum_k p_{k4} = 7$$
$$\delta \leqslant C_{\max} - \sum_k p_{k3} = 5.$$

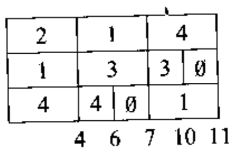
Suppose δ is chosen to be maximum, subject to conditions indicated above. Then either P' will contain at least one less strictly positive element than P or else P' will contain at least one more tight column or tight row (with respect to $C'_{\max}$) than P. It is thus apparent that no more than $r+m+n$ iterations, where r is the number of strictly positive elements in P, are necessary to construct a feasible schedule of length $C_{\max}$.

To illustrate this point, we continue with the example. Choosing $\delta = 3$, we obtain from P' the matrix P', with the augmented partial schedule shown to the right:

$$P'' = \begin{array}{c|c} \begin{pmatrix} 0 & 0 & 0 & \textcircled{4} \\ 0 & 0 & \textcircled{3} & 0 \\ \textcircled{4} & 0 & 0 & 0 \end{pmatrix} & \begin{array}{c} 4 \\ 3 \\ 4 \end{array} \\ \begin{array}{cccc} 4 & 0 & 3 & 4 \end{array} & \end{array}$$



(The symbol "ϕ" indicates idle time.) The final decrementing set yields the following complete schedule:



To complete our proof, we need the following lemma.

Lemma 7. For any nonnegative matrix P and $C_{\max}$ satisfying condition (5), there exists a decrementing set.

Proof: From the $m \times n$ matrix P construct an $(m+n) \times (m+n)$ matrix U, as indicated below:

$$U = \left(\begin{array}{c|c} P & D_m \\ \hline D_n & P' \end{array}\right)$$

Here P' denote the transpose of P, D_m and D_n are $m \times m$ and $n \times n$ diagonal matrices of nonnegative "slacks", determined in such a way that each row sum and column sum of U is equal to $C_{\max}$. It follows that $(1/C_{\max})\ U$ is a doubly stochastic matrix. The well-known Birkhoff-von Neumann theorem states that a doubly stochastic matrix is a convex combination of permutation matrices. It is easily verified that any one of the permutation matrices in such a convex combination is identified with a decrementing set of P. □

There are several possible ways to construct a decrementing set. For our purposes, it is sufficient to note that one can construct the matrix U from P and then solve an assignment problem over U, which can be done in polynomial time. This observation, together with the observation that no more than a polynomial number of such assignment problems need be solved, is sufficient to establish a polynomial bound for the schedule construction procedure. In (Gonzalez & Sahni, 1976) time bounds of $O(r^2)$ and $O(r(\min\{r, m^2\} + m \log n))$, are obtained, where r is the number of strictly positive elements in P.

3. Preemptive Scheduling of Unrelated Parallel Machines

Let us now consider $R|pmtn|C_{\max}$, the problem of finding a minimum-length preemptive schedule for unrelated parallel machines. Let p_{ij} denote the total amount of time that machine i is to work on machine j. We assume that the processing requirements have been normalized so that for each job j we must have $\sum_i s_{ij} p_{ij} = 1$. It is evident that the values of $C_{\max}$ and p_{ij} for any feasible schedule must constitute a feasible solution to the following linear programming problem (Lawler & Labetoulle, 1978):

$$\begin{array}{ll} \text{minimize } C_{\max} & \\ \text{subject to} & \\ \sum_i s_{ij} p_{ij} = 1, & j = 1, \ldots, n \\ \sum_i p_{ij} \leqslant C_{\max} & j = 1, \ldots, n \\ \sum_j p_{ij} \leqslant C_{\max} & i = 1, \ldots, m \\ p_{ij} \geqslant 0. & \end{array}$$

It follows from the results of the previous section that the converse is also true: For any feasible solution to this linear programming problem there is a feasible preemptive schedule with the same values of p_{ij} and $C_{\max}$. Moreover, such a schedule can be constructed in polynomial time.

Because of the existence of the ellipsoid method for linear programming, the linear programming formulation of $R|pmtn|C_{\max}$ can be regarded yielding a polynomial-bounded algorithm for the problem. In the case of two machines, $R2|pmtn|C_{\max}$, the linear programming problem can be solved in $O(n)$ time by special techniques (Gonzalez, et al, 1982).

It is a fairly straightforward matter to formulate a linear programming problem for $R|pmtn|L_{\max}$ and $R|r_j, pmtn|C_{\max}$. Moreover, $R|r_j, pmtn|L_{\max}$ can be solved as a series of linear programming problems.

Open Problem: An upper bound of $4m^2-5m+2$ on the number of preemptions required for an optimal schedule for $R|pmtn|C_{\max}$ is established in (Lawler & Labetoulle, 1978). However, this upper bound is certainly not tight. It is known that no more than two preemptions are required for the case $m=2$ (Gonzalez, et al, 1981), and no example is known to require more than $2(m-1)$ preemptions. What is a tight upper bound?

4. Minimizing the Sum of Completion Times

The problem $R\|\Sigma C_j$ is solved by the technique of (Bruno & Coffman, 1976). Note that if jobs $1, \ldots, k$ are performed by machine i in that order, we have for the sum of the completion times of the jobs performed on that machine,

$$\Sigma C_j = \frac{k}{s_{i1}} + \frac{(k-1)}{s_{i2}} + \ldots + \frac{2}{s_{i,k-1}} + \frac{1}{s_{ik}}.$$

In other words, if job j is the last job performed on machine i, its contribution to the objective function is $\frac{1}{s_{ij}}$, if j is the second-to-last job on machine i, its contribution $\frac{2}{s_{ij}}$, and so forth. A little reflection shows that we can formulate the problem $R\|\Sigma C_j$ as follows.

Let $S' = \left(\frac{1}{s_{ij}}\right)$ and let S'' be the matrix obtained by stacking multiples of S', as shown in Figure 7. A solution to the problem is obtained by finding a subset of elements of S'' of minimum total value, subject to the conditions that at least one element is chosen from each column of S'' (each job is performed by some machine) and at most one element is chosen from each row (no two jobs are

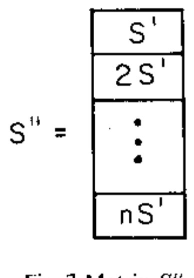


Fig. 7. Matrix S''

performed in the same position on a given machine). In other words, the problem reduces to a simple transportation problem.

Curiously, no good algorithm is known for $R|pmtn|\Sigma C_j$. If a "birdie" were to tell us the order in which jobs are to be completed in an optimal schedule, we could easily formulate and solve a linear programming problem to find such an optimal schedule. But this observation is not much help.

Open Question: What is the status of $R|pmtn|\Sigma C_j$? Easy of NP-hard? (If this problem is NP-hard, it will be the first such problem known to the author in which the nonpreemptive version is easy.)

V. Flow Shops and Job Shops

1. Introduction

Flow shops and *job shops* are much like open shops, except that for each job there is a prescribed order in which the job must be processed by the m machines. In a flow shop this order is the same for all jobs, i. e. each job must first be processed by machine 1, then machine 2, ... and finally by machine m. In a job shop, the prescribed order may differ from job to job. (Quite commonly, job shops are defined in terms of "operations" which must be performed on a given job in a prescribed order, and two or more operations may be performed by the same machine. However, we are not concerned with this distinction here.)

We shall not have much to say about flow shops or job shops in this survey paper, because nearly all flow shops and job shop problems are NP-hard. Essentially the only known polynomial algorithms are for special cases of the two-machine flow shop problem. Moreover, empirical work has shown that flow shop and job shop problems, other than these special cases, are extremely difficult to solve.

2. The Two-Machine Flow Shop

One of the most significant pioneering works of scheduling theory is (Johnson, 1954), in which an $O(n \log n)$ procedure was given for minimizing makespan in the two-machine flow shop. Johnson's results have been extended to encompass "time-lags" (Mitten, 1958), and series parallel precedence constraints (Sidney, 1979).

3. The "No-Wait" Two-Machine Flow Shop

The "no wait" flow shop is one in which each job must start its processing on machine $i+1$ the instant its processing is completed on machine i. Interestingly, the two-machine "no wait" problem is actually a special case of the traveling salesman problem that can be solved efficiently (Reddi & Ramamoorthy, 1972; Gilmore & Gomory, 1964).

4. The General Job Shop Problem

The general job shop problem is one of the most computationally intractable combinatorial problems existing. One indication of this is given by the fact that a certain 10-job, 10-machine problem formulated in 1963 (Muth & Thompson, 1963) still has not been solved, despite the efforts of many investigators. This challenge for future research perhaps provides us with an appropriate note on which to end this survey.

References

D. L. Adolphson, "Single Machine Job Scheduling With Precedence Constraints", *SIAM J. Comput.*, 6 (1977) 40-54.

D. L. Adolphson, T. C. Hu, "Optimal Linear Ordering", *SIAM J. Appl. Math.*, 25 (1973) 403-423.

H. M. Abdel-Wahab, T. Kameda, "Scheduling to Minimize Maximum Cumulative Cost Subject to Series Parallel Precedence Constraints", *Operations Res.*, 26 (1978) 141-158.

K. R. Baker, *Introduction to Sequencing and Scheduling*, Wiley, New York, 1974.

K. R. Baker, L. E. Schrage, "Finding an Optimal Sequence by Dynamic Programming: An Extension to Precedence-Constrained Tasks", *Operations Res.*, 26 (1978) 111-120.

K. R. Baker, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, "Preemptive Scheduling of a Single Machine to Minimize Maximum Cost Subject to Release Dates and Precedence Constraints", (1982) to appear in *Operations Res.*

P. Brucker, M. R. Garey, D. S. Johnson, "Scheduling Equal-Length Tasks under Tree-Like Precedence Constraints to Minimize Maximum Lateness", *Math. Operations Res.*, 2 (1977) 275-284.

E. G. Coffman, Jr., R. L. Graham, "Optimal Scheduling for Two-Processor Systems", *Acta Informat.*, 1 (1972) 200-213.

R. W. Conway, W. L. Maxwell, L. W. Miller, *Theory of Scheduling*, Addison-Wesley, Reading, Mass. 1967.

G. I. Davida, D. J. Linton, "A New Algorithm for the Scheduling of Tree Structured Tasks", *Proc. Conf. Inform. Sci. and Systems.*, Baltimore, MD., 1976.

D. Dolev, "Scheduling Wide Graphs", (1981), unpublished manuscript.

H. Emmons, "One-Machine Sequencing to Minimize Certain Functions of Job Tardiness", *Operations Res.*, 17 (1969) 701-715.

M. L. Fisher, "A Dual Algorithm for the One-Machine Scheduling Problem", *Math. Programming*, 11 (1976) 229-251.

M. L. Fisher, B. J. Lageweg, J. K. Lenstra, A. H. G. Rinnooy Kan, "Surrogate Duality Relaxation for Job Shop Scheduling", Report, *Mathematisch Centrum*, Amsterdam, 1981.

M. Fujii, T. Kasami, K. Ninomiya, "Optimal Sequencing of Two Equivalent Processors", *SIAM J. Appl. Math.*, 17 (1969) 784–789; *Erratum*, 20 (1971) 141.

H. N. Gabow, "An Almost-Linear Algorithm for Two-Processor Scheduling", *J. Assoc. Comput. Mach.*, 29 (1982 A) 766–780.

H. N. Gabow, private communication, 1982 B.

M. R. Garey, "Optimal Task Sequencing with Precedence Constraints", *Discrete Math.*, 4 (1973) 37–56.

M. R. Garey, D. S. Johnson, "Scheduling Tasks with Nonuniform Deadlines on Two Processors", *J. Assoc. Comput. Mach.*, 23 (1976) 461–467.

M. R. Garey, D. S. Johnson, "Two-Processor Scheduling with Start-Times and Deadlines", *SIAM J. Comput.*, 6 (1977) 416–426.

M. R. Garey, D. S. Johnson, R. E. Tarjan, M. Yannakakis, "Scheduling Opposing Forests", unpublished manuscript, (1981).

P. C. Gilmore, R. E. Gomory, "Sequencing a One-State Variable Machine: A Solvable Case of the Traveling Salesman Problem", *Oper. Res.*, 12 (1964) 655–679.

T. Gonzalez, "Optimal Mean Finish Time Preemptive Schedules", Technical Report 220, (1977) Computer Science Department, Pennsylvania State University.

T. Gonzalez, "A Note on Open Shop Preemptive Schedules", *IEEE Trans. Computers*, C-28 (1979) 782–786.

T. Gonzalez, D. B. Johnson, "A New Algorithm for Preemptive Scheduling of Trees", *J. Assoc. Comput. Mach.*, 27 (1980) 287–312.

T. Gonzalez, E. L. Lawler, S Sahni, "Optimal Preemptive Scheduling of Two Unrelated Processors in Linear Time", (1981), to appear.

T. Gonzalez, S. Sahni, "Open Shop Scheduling to Minimize Finish Time", *J. Assoc. Comput. Mach.*, 23 (1976) 665–679.

T. Gonzalez, S. Sahni, "Preemptive Scheduling of Uniform Processor Systems", *J. Assoc. Comput. Mach.*, 25 (1978) 92–101.

R. L. Graham, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, "Optimization and Approximation in Deterministic Sequencing and Scheduling : A Survey", *Ann. Discrete Math.*, 5 (1979) 287–326.

M. Held, R. M. Karp, "A Dynamic Programming Approach to Sequencing Problems", *SIAM J. Appl. Math.*, 10 (1972) 196–210.

W. A. Horn, "Single-Machine Job Sequencing with Treelike Precedence Ordering and Linear Dealy Penalties", *SIAM J. Appl. Math.*, 23 (1972) 189–202.

W. A. Horn, "Minimizing Average Flow Time with Parallel Machines", *Oper. Res.*, 21 (1973) 846–847.

W. A. Horn, "Some Simple Scheduling Algorithms", *Naval Res. Logist. Quart.*, 21 (1974) 177–185.

E. Horowitz, S. Sahni, "Exact and Approximate Algorithms for Scheduling Nonidentical Processors", *J. Assoc. Comput. Mach.*, 23 (1976) 317–327.

T. C. Hu, "Parallel Sequencing and Assembly Line Problems", *Oper. Res.*, 9 (1961) 841–848.

J. R. Jackson, "Scheduling a Production Line to Minimize Maximum Tardiness", Research Report 43, (1955) Management Science Research Project, University of California, Los Angeles.

J. R. Jackson, "An Extension of Johnson's Results on Job Lot Scheduling", *Naval Res. Logist. Quart.*, 3 (1956) 201–203.

S. M. Johnson, "Optimal Two- and Three-Stage Production Schedules with Setup Times Included", *Naval Res. Logist. Quart.*, 1 (1954) 61–68.

S. M. Johnson, "Discussion: Sequencing *n* Jobs on Two Machines with Arbitrary Time Lags", *Management Sci.*, 5 (1958) 299–303.

E. P. C. Kao, M. Queyranne, "On Dynamic Programming Methods for Assembly Line Balancing", working paper, Dept Quantitative Management Sci., University of Houston, Texas, (1980).

H. Kise, T. Ibaraki, H. Mine, "A Solvable Case of the One-Machine Scheduling Problem with Ready and Due Times", *Oper. Res.*, 26 (1978) 121-126.

J. Labetoulle, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, "Preemptive Scheduling of Uniform Machines Subject to Release Dates", Report BW 99, (1979) Mathematisch Centrum, Amsterdam.

B. J. Lageweg, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, "Computer Aided Complexity Classification of Deterministic Scheduling Problems", Report BW 138, (1981) Mathematisch Centrum, Amsterdam.

E. L. Lawler, "On Scheduling Problems with Deferral Costs", *Management Science*, 11 (1964) 270-288.

E. L. Lawler, "Optimal Sequencing of a Single Machine Subject to Precedence Constraints", *Management Sci.*, 19 (1973) 544-546.

E. L. Lawler, "Sequencing to Minimize the Weighted Number of Tardy Jobs", *RAIRO Rech. Oper.*, 10 suppl. (1976) 27-33.

E. L. Lawler, "A 'Pseudopolynomial' Algorithm for Sequencing Jobs to Minimize Total Tardiness", *Ann. Discrete Math.*, 1 (1977) 331-342.

E. L. Lawler, "Sequencing Jobs to Minimize Total Weighted Completion Time Subject to Precedence Constraints", *Ann. Discrete Math.*, 2 (1978 A) 75-90.

E. L. Lawler, "Sequencing Problems With Series Parallel Precedence Constraints", to appear in *Proc. Confer. on Combinatorial Optimization*, (N. Christofides, ed.), Urbino, Italy, 1978 B.

E. L. Lawler, "Efficient Implementation of Dynamic Programming Algorithms for Sequencing Problems", Report BW 106/79, (1979 A) Mathematisch Centrum, Amsterdam.

E. L. Lawler, "Preemptive Scheduling of Uniform Parallel Machines to Minimize the number of Late Jobs", Report BW 105, (1979 B) Mathematisch Centrum, Amsterdam.

E. L. Lawler, "Preemptive Scheduling of Precedence-Constrained Jobs on Parallel Machines", in *Deterministic and Stochastic Scheduling*, (M. A. H. Dempster, et al., eds.), D. Reidel, Dordrecht, Holland, 1982 A, pp. 101-124.

E. L. Lawler, "A Fully Polynomial Approximation Scheme for the Total Tardiness Problems", (1982 B), submitted for publication.

E. L. Lawler, "On Scheduling a Single Machine to Minimize the Number of Late Jobs", (1982 C), submitted for publication.

E. L. Lawler, J. Labetoulle, "On Preemptive Scheduling of Unrelated Parallel Processors by Linear Programming", *J. Assoc. Comput. Mach.*, 25 (1978) 612-619.

E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, "Minimizing Maximum Lateness in a Two-Machine Open Shop, *Math. Oper. Res.*, 6 (1981) 153-158.

E. L. Lawler, C. U. Martel, "Scheduling Periodically Occurring Tasks on Multiple Professors", *Info. Proc. Letters*, 12 (1981) 9-12.

E. L. Lawler, C. U. Martel, "Computing 'Polymatroidal' Network Flows", *Math. of Operations Res.*, 7 (1982 A) 334-347.

E. L. Lawler, C. U. Martel, "Flow Network Formulations of Polymatroid Optimization Problems", *Annals Discrete Math.*, 16 (1982 B) 189-200.

E. L. Lawler, J. K. Lenstra, "Machine Scheduling with Precedence Constraints", in *Ordered Sets*, (I. Rival, ed.), D. Reidel, Dordrecht, Holland, 1982, pp. 655-675.

E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, "Recent Developments in Deterministic Sequencing and Scheduling: A Survey", in *Deterministic Sequencing and Sched-*

uling, (M. A. H. Dempster, et al., eds.), D. Reidel Co., Dordrecht, Holland, 1982A, pp. 35–74.

E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, "A Gift for Alexander!: At Play in the Fields of Scheduling Theory", *OPTIMA*, Mathematical Programming Society Newsletter, No. 7, (1982B).

E. L. Lawler, M. G. Luby, V. V. Vazirani, "Scheduling Open Shops with Parallel Machines", (1982), to appear in *Operations Res. Letters*.

E. L. Lawler, J. M. Moore, "A Functional Equation and Its Application to Resource Allocation and Sequencing Problems", *Management Sci.*, 16 (1969) 77–84.

E. L. Lawler, B. D. Sivazlian, "Minimization of Time Varying Costs in Single Machine Scheduling", *Operations Res.*, 26 (1978) 563–569.

J. K. Lenstra, *Sequencing by Enumerative Methods*, Mathematical Centre Tracts 69, Mathematisch Centrum, Amsterdam, 1977.

J. K. Lenstra, A. H. G. Rinnooy Kan, "Complexity of Scheduling under Precedence Constraints", *Operations Res.*, 26 (1978) 22–35.

C. Martel, "Scheduling Uniform Machines With Release Times, Deadlines and Due Times", *J. Assoc. Comput. Mach.*, (1981), to appear.

R. McNaughton, "Scheduling With Deadlines and Loss Functions", *Management Sci.*, 6 (1959) 1–12.

L. G. Mitten, "Sequencing *n* Jobs On Two Machines With Arbitrary Time Lags", *Management Sci.*, 5 (1958) 293–298.

C. L. Monma, A. H. G. Rinnooy Kan, "Efficiently Solvable Special Cases of the Permutation Flow-Shop Problem", Report 8105, (1981) Erasmus University, Rotterdam.

C. L. Monma, J. B. Sidney, "Sequencing With Series-Parallel Precedence Constraints", *Math. Oper. Res.*, 4 (1979) 215–224.

J. M. Moore, "An *n* Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs", *Management Sci.*, 15 (1968) 102–109.

J. F. Muth, G. L. Thompson, eds., *Industrial Scheduling*, Prentice Hall, Englewood Cliffs, N. J., 1963, p. 236.

S. S. Reddi, C. V. Ramamoorthy, "On the Flow-Shop Sequencing Problem With No Wait in Process", *Oper. Res. Quart.*, 23 (1972) 323–331.

M. H. Rothkopf, "Scheduling Independent Tasks on Parallel Processors", *Management Sci.*, 12 (1966) 437–447.

S. Sahni, Y. Cho, "Nearly On Line Scheduling of a Uniform Processor System With Release Times", *SIAM J. Comput.*, 8 (1979) 275–285.

S. Sahni, Y. Cho, "Scheduling Independent Tasks With Due Times on a Uniform Processor System", *J. Assoc. Comput. Mach.*, 27 (1980) 550–563.

J. B. Sidney, "An Extension of Moore's Due Date Algorithm", in *Symposium on the Theory of Scheduling and Its Applications*, Lecture Notes in Economics and Mathematical Systems 86, (S. E. Elmaghraby, ed.), Springer, Berlin, 1973, pp. 393–398.

J. B. Sidney, "Decomposition Algorithms for Single-Machine Sequencing With Precedence Relations and Deferral Costs", *Oper. Res.*, 23 (1975) 283–298.

J. B. Sidney, "The Two-Machine Maximum Flow Time Problem With Series Parallel Precedence Relations", *Oper. Res.*, 27 (1979) 782–791.

B. Simons, "A Fast Algorithm for Single Processor Scheduling", *Proc. 19th Annual IEEE Symp. Foundations of Computer Science*, (1978) 50–53.

W. E. Smith, "Various Optimizers for Single-Stage Production", *Naval Res. Logist. Quart.*, 3 (1956) 59–66.

J. D. Ullman, "NP-Complete Scheduling Problems", *J. Comput. System Sci.*, 10 (1975) 384–393.

J. Valdes, R. E. Tarjan, E. L. Lawler, "The Recognition of Series Parallel Digraphs", *SIAM J. Computing*, 11 (1982) 298–313.

M. K. Warmuth, "M Processor Unit-Execution-Time Scheduling Reduces to M-1 Weakly Connected Components", M. S. Thesis, (1980) Department of Computer Science, University of Colorado, Boulder.

G. Weiss, M. Pinedo, "Scheduling Tasks with Exponential Service Times on Non Identical Processors to Minimize Various Functions", *J. Appl. Probab.*, 17 (1980) 187–202.