stichting
mathematisch
centrum

MC

AFDELING MATHEMATISCHE BESLISKUNDE (DEPARTMENT OF OPERATIONS RESEARCH)

BW 148/81

SEPTEMBER

E.L. LAWLER, J.K. LENSTRA

MACHINE SCHEDULING WITH PRECEDENCE CONSTRAINTS

Preprint

kruislaan 413 1098 SJ amsterdam

Printed at the Mathematical Centre, 413 Kruislaan, Amsterdam.

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

1980 Mathematics Subject Classification: 90B35, 68C15, 68C25, 06A10

MACHINE SCHEDULING WITH PRECEDENCE CONSTRAINTS

E.L. LAWLER
University of California, Berkeley

J.K. LENSTRA
Mathematisch Centrum, Amsterdam

ABSTRACT

This is an expository survey of the theory of precedence constrained scheduling problems. Such problems ask for an optimal allocation of machines to jobs subject to a number of constraints, including a partial ordering of the jobs. After a brief introduction to computational complexity theory, we review several problem classes for which polynomial-time algorithms exist and complement these results by indicating related problems that are known to be NP-hard.

KEY WORDS AND PHRASES: *deterministic scheduling, computational complexity, polynomial-time algorithm, NP-hardness, single machine, parallel machines, precedence relation, series-parallel constraints, tree-like constraints.*
NOTE: This report will appear in the Proceedings of the Symposium on Ordered Sets, Banff, Canada, August 28-September 12, 1981.

1. INTRODUCTION

Machine scheduling theory is concerned with the allocation over time of scarce resources in the form of *machines* or *processors* to activities known as *jobs* or *tasks*. If the jobs can be performed in any order, they are said to be *independent*. However, this is often not the case; technological or other constraints may dictate that of two given jobs one must be performed before the other. Such *precedence constraints* of course impose a partial ordering on the job set.

Machine scheduling problems occur in many different situations. Suppose a conference organizer has to sequence n lectures, some of which are based on material presented in other lectures. If each lecturer has specified the times of his arrival and departure, the organizer is faced with the task of finding a feasible schedule for n jobs on a single machine subject to precedence constraints, release dates and deadlines (see Section 3). Suppose the conference participants are simultaneously arriving in n airplanes, which are waiting for permission to land on a single runway. If the objective is to keep total gasoline usage as low as possible, the air controller has to minimize the weighted sum of n job completion times on a single machine (see Section 4). A third example is the Chinese Cook Problem [Lawler *et al.* 1976]. Suppose the conference dinner consists of n courses and is to be prepared on a stove with m burners. In order to start as late as possible, the cook has to minimize maximum completion time for n jobs on m parallel machines; it is easily imagined that in this situation precedence constraints between the jobs are specified (see Section 5).

A variety of techniques has been developed for the solution of machine scheduling problems [Graham *et al.* 1979]. Such algorithmic results are often complemented by results stating that satisfactory solution techniques for related problems will probably never be found. The theory of *computational complexity* provides the tools to make a formal distinction between *well-solved* problems, which are solvable by an algorithm whose running time is bounded by a polynomial function of problem size, and *NP-hard* problems, for which the existence of such an algorithm is highly unlikely. In Section 2 we give a brief introduction to this theory.

There appear to be only three classes of precedence constrained scheduling problems for which polynomial-time algorithms exist: *single machine problems in the case of min-max optimization*, certain problems with *series-parallel precedence constraints*, and certain *parallel machine problems*. In Sections 3, 4 and 5 we describe these problem classes, indicate the nature of the algorithmic techniques

that have been used to successfully cope with precedence constraints in each of them, and present the relevant *NP*-hardness results. In Section 6 we make some concluding remarks.

2. COMPUTATIONAL COMPLEXITY THEORY

For many combinatorial optimization problems, highly efficient solution techniques exist. For many other such problems, there is little hope of obtaining an optimal solution in a reasonable amount of time, and one has to choose between the use of fast heuristics, which yield only an approximate solution, or of enumerative methods, which produce an optimal solution only after an often time consuming search through the set of feasible solutions. A distinction between these problem classes can be made by means of concepts from the theory of computational complexity. An informal exposition of these concepts is given below. The reader is referred to [Karp 1975; Lenstra & Rinnooy Kan 1979] for a more extensive introduction and to [Garey & Johnson 1979] for a comprehensive treatment of the theory.

Let us define the *size* of a problem as the number of bits needed to encode its data, and the *running time* of an algorithm as the number of elementary operations (such as additions and comparisons) required for its solution.

If a problem of size s can be solved by an algorithm with running time $O(p(s))$ where p is a *polynomial* function, then the problem is considered to be *well solved*. The justifications of this notion are the following: *machine independence* (theoretical computing devices like Turing machines and ordinary computers are all polynomially related), *asymptotic behavior* (any polynomial-time algorithm is faster than any superpolynomial one for sufficiently large problems), *accordance with experience* (polynomial-time algorithms tend to be efficient in practice), and *susceptability to theoretical analysis* (the subject of this section). Polynomial-time algorithms exist for a wide variety of combinatorial optimization problems, such as finding shortest paths, maximum flows and maximum matchings in graphs [Lawler 1976].

Computational complexity theory deals primarily with *decision problems*, which require a yes/no answer, rather than with optimization problems. Three decision problems that will recur below are:

SATISFIABILITY
Instance: A conjunctive normal form expression, *i.e.*, a conjunction of clauses, each of which is a disjunction of literals $x_1,\bar{x}_1, \ldots,x_t,\bar{x}_t$, where $x_1,\ldots,x_t$ are boolean variables and $\bar{x}_1,\ldots,\bar{x}_t$ denote their complements.
Question: Does there exist a truth assignment to the variables such that the expression assumes the value *true*?

CLIQUE
Instance: A graph $G = (V,E)$ and an integer c.
Question: Does there exist a subset $C \subset V$ of cardinality c such that $\{j,k\} \in E$ if $\{j,k\} \subset C$?

PARTITION
Instance: Positive integers $a_1,\ldots,a_t,b$ with $a_1+\ldots+a_t = 2b$.
Question: Does there exist a subset $S \subset \{1,\ldots,t\}$ such that $\Sigma_{j\in S}\, a_j = b$?

An instance of a decision problem is said to be *feasible* if the question can be answered affirmatively. Feasibility is usually characterized by the existence of an associated *structure* which satisfies a certain property. *E.g.*, in the case of CLIQUE the structure is a set of c pairwise adjacent vertices.

We now define two important problem classes. A decision problem is in the class P if, for any instance, one can determine its feasibility or infeasibility in polynomial time. It is in the class NP if, for any instance, one can determine in polynomial time whether a given structure affirms its feasibility. *E.g.*, CLIQUE is a member of NP, since for any clique C one can verify that all its vertices are pairwise adjacent in $O(c^2)$ time. Also SATISFIABILITY and PARTITION belong to NP.

It is clear that $P \subseteq NP$. The question if this inclusion is a proper one or if equality holds is probably the foremost open problem in theoretical computer science. However, *it is very unlikely that* $P = NP$, since NP contains many notorious combinatorial problems for which, in spite of considerable research efforts, no polynomial-time algorithms have been found so far.

Cook [Cook 1971] proved that SATISFIABILITY is the most difficult problem in NP by showing that every other problem in NP is *reducible* to it. That is, for any instance of any problem $P \in NP$ a corresponding instance of SATISFIABILITY can be constructed in polynomial time such that solving the latter instance will solve the instance of P as well. He also proved that CLIQUE is as hard as SATISFIABILITY, simply by giving a reduction of SATISFIABILITY to CLIQUE. Both problems are said to be *NP-complete* in the sense that (*i*) they belong to NP, and (*ii*) every other problem in NP is reducible to them.

Karp [Karp 1972] elaborated on these ideas and identified a large number of NP-complete problems P (including PARTITION) by verifying that $P \in NP$ and specifying a reduction of a known NP-complete problem to P. Since then, hundreds of combinatorial problems have been shown to be NP-complete; we refer to [Garey & Johnson 1979] for an impressive compilation of these results.

A polynomial-time algorithm for an NP-complete problem P could be used to solve any problem in NP in polynomial time by first reducing it to P and then applying the algorithm for P, and the existence of such an algorithm would thus imply that $P = NP$. Hence, *it is very unlikely that* $P \in P$ *for any* NP*-complete* P. This observa-

tion justifies the use of heuristic or enumerative methods for its solution.

In dealing with the computational complexity of *optimization problems*, one usually reformulates the problem of finding a feasible solution of, say, minimum value as the problem of deciding whether there exists a feasible solution with value at most equal to a given threshold. If this decision problem is *NP*-complete, then the optimization problem is said to be *NP-hard* in the sense that the existence of a polynomial-time algorithm for its solution would imply that $P = NP$.

3. SINGLE MACHINE SCHEDULING TO MINIMIZE MAXIMUM COST

Suppose n *jobs* are to be processed on a *single machine* which can execute at most one job at a time. Each job j requires an uninterrupted *processing time* p_j. Moreover, *precedence constraints* between the jobs are 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. Associated with each job j is a monotone nondecreasing *cost function* f_j; if job j is completed at time C_j, a cost $f_j(C_j)$ is incurred. It is possible to find a schedule which minimizes the *maximum of the job completion costs* $\max_j\{f_j(C_j)\}$ by the following simple 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.

The correctness of this rule can be proved as follows. Let $N = \{1,\ldots,n\}$ be the set of all jobs, let $q = p_1+\ldots+p_n$, 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. If job $l \in L$ is chosen such that

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

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

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

Since neither of the maximands is larger than $f^*(N)$, there exists an optimal schedule in which job l is in the last position.

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.

We next consider an extension of this problem, in which each job j becomes available for processing at a specified *release date* r_j. In this situation it may be advantageous to relax the requirement that each job is to be processed without interruption.

Instance of PARTITION:
$t = 7$, $a_j = j$ $(j = 1,\ldots,7)$, $b = 14$.
Solution:
$S = \{1,6,7\}$.
Corresponding schedule:

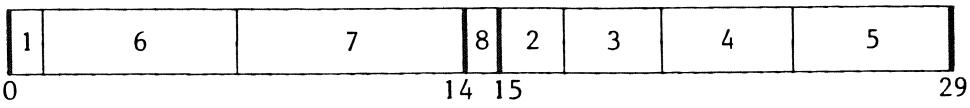


Figure 1. Illustration of the reduction from PARTITION.

The above algorithm has been generalized to the situation in which *preemption* is permitted, *i.e.*, the processing of any job may arbitrarily often be interrupted and resumed at a later time, without penalty [Baker *et al.* 1982].

In contrast, the *nonpreemptive* version of this problem is NP-hard. More specifically, the problem of deciding whether there exists a *feasible* schedule in which each job j is processed for an uninterrupted period of length p_j between a given *release date* r_j and a given *deadline* d_j is NP-complete, even in the absence of precedence constraints [Garey & Johnson 1977; Lenstra *et al.* 1977]. The proof is given below.

The feasibility problem belongs to NP, since any given schedule can be tested for feasibility in linear time. It suffices to prove that a known NP-complete problem is reducible to the feasibility problem. Consider the PARTITION problem as formulated in Section 2. Given any instance of PARTITION, defined by positive integers $a_1,\ldots,a_t,b$ with $a_1+\ldots+a_t = 2b$, we construct a corresponding instance of the feasibility problem as follows: the number of jobs is given by

$$n = t+1;$$

there are t *partition jobs* j $(j = 1,\ldots,n)$ with

$$r_j = 0, \quad p_j = a_j, \quad d_j = 2b+1;$$

there is one *splitting job* n with

$$r_n = b, \quad p_n = 1, \quad d_n = b+1.$$

It should be obvious that PARTITION has a solution if and only if there exists a feasible schedule (*cf.* Figure 1), and hence PARTITION is reducible to the feasibility problem.

The remainder of this section deals with the special case in which the cost function of each job j measures its *lateness* $f_j(C_j) = C_j-d_j$ with respect to a given *due date* d_j. Of course, the problem of minimizing *maximum lateness* is NP-hard if processing times, release dates and due dates are arbitrary, but it is possible to

find an optimal schedule in polynomial time if all p_j are equal, all r_j are equal, or all d_j are equal.

First assume that the jobs are *independent*. One of the oldest results in scheduling theory is the "earliest due date rule" [Jackson 1955], which states that the case of equal release dates is solved by sequencing the jobs in order of nondecreasing due dates; this rule can be viewed as a specialization of Lawler's algorithm. Similarly, the case of equal due dates is solved by sequencing the jobs in order of nondecreasing release dates.

The case of unit processing times and integer release and due dates is solved by an extension of Jackson's rule [Baker & Su 1974], which schedules at any time an available job with smallest due date. The case of equal processing times requires a more sophisticated approach but is still solvable in polynomial time [Simons 1978; Garey *et al.* 1981A].

Now suppose that *precedence constraints* are specified. Consider a feasible schedule in which two jobs j and k with $j \rightarrow k$ are completed at times C_j and C_k, respectively; note that $C_j \leq C_k - p_k$. The feasibility of the schedule is not affected if we set

$$r_k := \max\{r_j + p_j, r_k\},$$

since we have for the starting time of job k that $C_k - p_k \geq C_j \geq r_j + p_j$. The criterion value of the schedule does not change if we set

$$d_j := \min\{d_j, d_k - p_k\},$$

since we have for the lateness of job k that $C_k - d_k \geq C_j - (d_k - p_k)$. Hence, we may modify release and due dates as indicated above, so that $r_j < r_k$ and $d_j < d_k$ whenever $j \rightarrow k$ [Lageweg *et al.* 1976].

The reader should have no difficulty in verifying that the algorithms for the cases that all r_j are equal, all d_j are equal or all $p_j = 1$ will, after modification of the r_j and d_j according to $\rightarrow$, automatically respect the precedence constraints. The algorithm for the case that all p_j are equal has the same property. However, in the general case this modification is not sufficient and the precedence constraints have to be taken explicitly into account.

4. SERIES-PARALLEL SCHEDULING

Another classical result in scheduling theory is the "ratio rule" [Smith 1956]. Suppose again that n *independent jobs* are to be processed on a *single machine*, and that in addition to a *processing time* p_j each job j has a specified *weight* w_j. It is possible to find a schedule which minimizes the *weighted sum of the job completion times* in $O(n \log n)$ time by sequencing the jobs in order of nondecreasing ratios $\rho_j = p_j / w_j$.

The problem becomes NP-hard if arbitrary *precedence constraints*

are permitted, even if all p_j are equal or all w_j are equal [Lawler 1978A; Lenstra & Rinnooy Kan 1978]. However, a number of special cases has been dealt with successfully, including precedence constraints admitting of various types of decompositions [Sidney 1975], rooted trees [Horn 1972; Adolphson & Hu 1973], and series-parallel precedence constraints [Lawler 1978A]. The algorithm for the last case has been generalized to apply to a diverse variety of other sequencing problems [Lawler 1978B; Monma & Sidney 1979]. In the following we give a brief review of the theory of series-parallel sequencing that has resulted from this generalization.

First, let us pose a very general type of sequencing problem. 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 $\leq$ (*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 $\alpha bc\delta$ we have

$$b \leq c \Rightarrow f(\alpha bc\delta) \leq f(\alpha cb\delta).$$

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". Smith's ratio rule observes this property.

It is a simple matter to verify the following.

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

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 $\leq$ 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

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

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 $\rho_\alpha = \Sigma_{j\in\alpha} p_j / \Sigma_{j\in\alpha} w_j$. 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 1978B].

THEOREM 2. *If f admits of a string interchange relation $\leq$ 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 $\leq$.*

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 $G = (\{j\},\emptyset)$ 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).

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 2. An acyclic digraph is series-parallel if and only if its transitive closure does not contain the Z-digraph 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.* 1981].

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 can be obtained by successive applications of Rules (1) and (2). A series-parallel digraph and its decomposition tree are shown in Figure 3. 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.

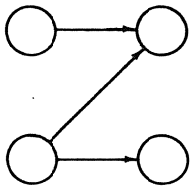
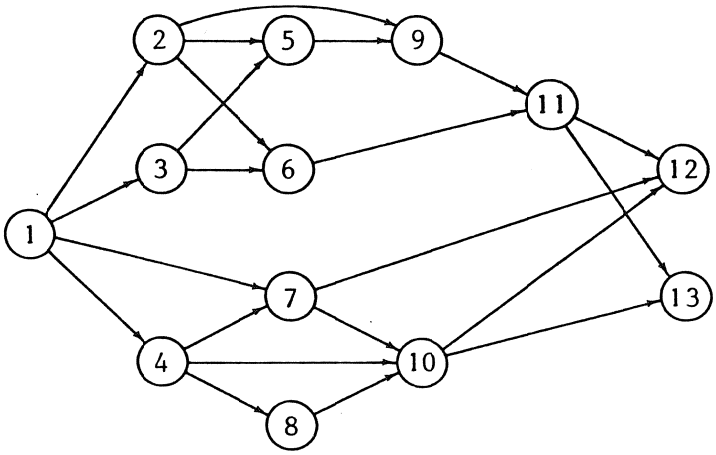


Figure 2. Z-digraph.



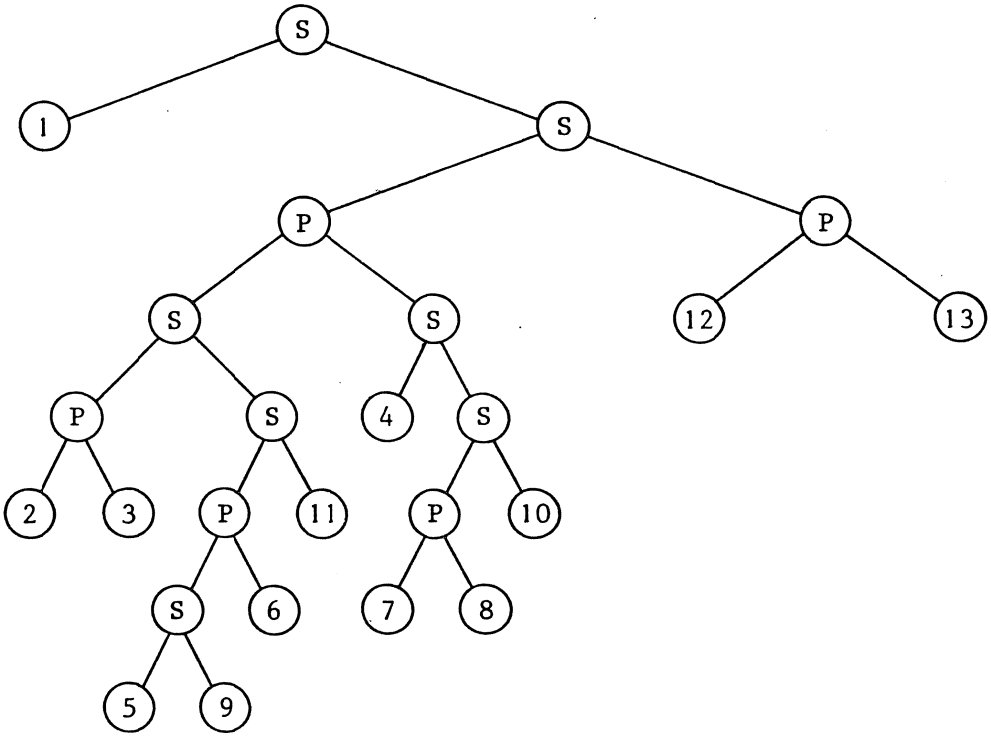


Figure 3. Series-parallel digraph and its decomposition tree.

Suppose we are to solve a sequencing problem for which a string interchange relation $\leq$ 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 $\leq$ 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 $\leq$ does not violate 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 $\leq$, 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

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

If $\sigma_2 \not\leq \sigma_1$, then conditions ($i$) and ($ii$) are still satisfied if for the S-node we let $S = S_1 \cup S_2$. If $\sigma_2 \leq \sigma_1$, we assert that there exists an optimal feasible permutation in which σ_1 and σ_2 are adjacent, *i.e.*, 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 1978B].) This suggests the following procedure:

```
BEGIN
     σ1 := max≤ S1; σ2 := min≤ S2;
     IF σ2 ≰ σ1
     THEN S := S1∪S2
     ELSE σ := σ1σ2;
          S1 := S1-{σ1}; σ1 := max≤ S1;
          S2 := S2-{σ2}; σ2 := min≤ S2;
          WHILE σ ≤ σ1 ∨ σ2 ≤ σ
          DO   IF σ ≤ σ1
               THEN σ := σ1σ;
                    S1 := S1-{σ1}; σ1 := max≤ S1
```

```
            ELSE σ := σσ2;
                 S2 := S2-{σ2}; σ2 := min≤ S2
            FI
        OD;
        S := S1∪{σ}∪S2
    FI
END.
```

(We make here the customary assumption that $\max_{\leq} \emptyset$ and $\min_{\leq} \emptyset$ 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 $\leq$.

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] or *minimum initial resource requirement* [Abdel-Wahab & Kameda 1978; Monma & Sidney 1979] on a single machine, and the *two-machine permutation flow shop problem with time lags* [Sidney 1979].

5. PARALLEL MACHINE SCHEDULING

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 the 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. Below we discuss these three basic results and mention a number of refinements and extensions.

First, let us assume that the precedence constraints are in the form of an *intree*, *i.e.*, each job has exactly one immediate successor, 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 successor. 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 of it was scheduled in s'; but then this 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.* 1981B; 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 [Lawler 1976].

A completely different approach by Coffman and Graham [Coffman & Graham 1972] leads to a *list scheduling* algorithm. The jobs are labeled in the following way. Suppose labels $1,\ldots,l$ 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 is almost linear in $n+a$, where a is the number of arcs in the precedence graph.

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 still exist [Garey & Johnson 1976, 1977]. These algorithms can be applied to minimize *maximum lateness* in polynomial time.

It is unlikely that the minimal common generalization of the two well-solved problems discussed above is solvable in polynomial time. More specifically, the problem of minimizing maximum completion time for n unit-time jobs on m identical parallel machines subject to arbitrary precedence constraints is *NP*-hard. This result is due to Ullman [Ullman 1975]; the proof to be given below is from [Lenstra & Rinnooy Kan 1978].

It suffices to prove that a known *NP*-complete problem is reducible to the scheduling problem. Consider the CLIQUE problem as formulated in Section 2. Given any instance of CLIQUE, defined by a graph $G = (V,E)$ and an integer c, let $v = |V|$, $e = |E|$, $d = \frac{1}{2}c(c-1)$, $c' = v-c$ and $d' = e-d$. We construct a corresponding instance of the scheduling problem as follows: the numbers of machines and jobs are given by

$$m = \max\{c, d+c', d'\}+1, \quad n = 3m;$$

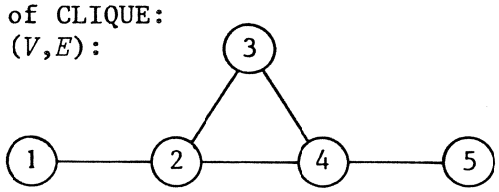
there are v *vertex jobs* j ($j \in V$) and e *edge jobs* $\{k,l\}$ ($\{k,l\} \in E$) with

$$j \to \{k,l\} \quad \text{whenever } j \in \{k,l\};$$

there are $n-v-e$ *dummy jobs* $\langle 1,i\rangle$ ($i = 1,\ldots,m-c$), $\langle 2,i'\rangle$ ($i' = 1,\ldots,m-d-c'$), $\langle 3,i''\rangle$ ($i'' = 1,\ldots,m-d'$) with

Instance of CLIQUE:

$G = (V,E)$:



$c = 3$.

Solution:

$C = \{2,3,4\}$.

Corresponding schedule:

	2	{2,3}	{1,2}
d'	3	{2,4}	{4,5}
c	4	{3,4}	<3,1>
	<1,1>	1	<3,2>
$d+c'$	<1,2>	5	<3,3>
m	<1,3>	<2,1>	<3,4>
	0	1	2 3

Figure 4. Illustration of the reduction from CLIQUE.

$\langle 1,i\rangle \rightarrow \langle 2,i'\rangle$ and $\langle 2,i'\rangle \rightarrow \langle 3,i''\rangle$ for any i,i',i''.

We claim that CLIQUE has a solution if and only if there exists a feasible schedule of length at most 3. The basic idea behind the reduction is the following. In any schedule of length 3, the dummy jobs create a pattern of idle machines during three unit-time intervals, available for the vertex and edge jobs. This pattern consists of c, $d+c'$ and d' idle machines during the first, second and third interval, respectively, and it can be filled properly if and only if CLIQUE has a solution (*cf.* Figure 4).

More precisely, suppose that CLIQUE has a solution $C \subset V$. We construct a feasible schedule of length 3 by processing the $m-c$ dummy jobs $\langle 1,i\rangle$ and the c clique vertex jobs j ($j \in C$) in the first interval, the $m-d-c'$ dummy jobs $\langle 2,i'\rangle$, the d clique edge jobs $\{k,l\}$ ($k,l \in C$) and the c' remaining vertex jobs in the second interval, and the $m-d'$ dummy jobs $\langle 3,i''\rangle$ and the d' remaining edge jobs in the third interval. Note that this schedule respects the precedence constraints.

Conversely, suppose that CLIQUE has no solution and assume that the dummy jobs $\langle 1,i\rangle$, $\langle 2,i'\rangle$ and $\langle 3,i''\rangle$ are processed in the first, second and third interval, respectively. Any set of c vertex jobs processed in the first interval releases at most $d-1$ edge jobs for processing in the second interval and since there are only c' vertex jobs left, at least one machine must remain idle during the

second interval. We conclude that any feasible schedule has length at least 4.

Two observations concerning this result are in order. If $P(t)$ denotes the problem of deciding whether there exists a feasible schedule of length at most t, then $P(3)$ has been shown to be *NP*-complete, whereas $P(2)$ can trivially be solved in $O(n^2)$ time. The problem thus exhibits the *mystical property of 3-ness*, which is encountered in many other combinatorial problems. Moreover, the *NP*-completeness of $P(3)$ implies that no polynomial-time algorithm for the minimization problem can guarantee a worst-case performance bound less that 4/3, unless $P = NP$.

In the above problem formulation, the number of machines is a variable specified as part of the problem instance. The complexity status of the problem is open, however, for any fixed $m \geq 3$. In particular, the case $m = 3$ remains one of the most vexing open problems in scheduling theory.

In this section we have concentrated on problems involving *nonpreemptive* scheduling of *unit-time* jobs. There has been a parallel investigation of problems concerning *preemptive* scheduling of jobs of *arbitrary length*. Gonzalez and Johnson [Gonzalez & Johnson 1980] have developed a polynomial-time algorithm for preemptive scheduling which is analogous to that from [Davida & Linton 1976]. Further, in [Lawler 1982] algorithms are proposed which are the preemptive counterparts of those found in [Brucker *et al.* 1977; 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 attempt to deal with them here.

6. CONCLUDING REMARKS

We have given an introductory survey of the most important polynomial-time algorithms and *NP*-hardness proofs for machine scheduling with precedence constraints. These results represent only a small fraction of the theory of deterministic sequencing and scheduling. Other models which have been studied in detail involve *nonidentical parallel machines*, *multi-operation jobs*, and a wide variety of *optimality criteria*. Also, a considerable amount of work has been done on the design and analysis of *approximation algorithms*. A comprehensive survey of the entire area is given in [Graham *et al.* 1979].

Given a well-defined class S of combinatorial optimization problems, it is often an easy matter to derive a *partial ordering* $\rightarrow$ on the class with the following property: if $P,Q \in S$ and $P \rightarrow Q$, then P is reducible to Q. This implies that, if there exists a polynomial-time algorithm for Q, then P is also well solved, and if P is *NP*-hard, then the same is true for Q.

For an extensive class of scheduling problems, a computer program has been developed that employs such a partial ordering to determine the complexity status of all problems in the class [Lageweg *et al.* 1981A, 1981B]. More precisely, the program receives as input the known research results in the form of a listing of well-solved problems and a listing of *NP*-hard problems. It then partitions the problem class into three subclasses of well-solved, open and *NP*-hard problems, and produces as output a count of these classes as well as listings of the problems in each of the subclasses which are *minimal* or *maximal* with respect to the partial ordering. The maximal well-solved problems and the minimal *NP*-hard problems represent the strongest results that have been obtained so far. The listings of minimal and maximal open problems have proved to be very useful as guidelines for further research.

It can be shown that, in a classification scheme like this, the problem of determining the minimum number of research results which together would resolve all remaining open problems, is itself *NP*-hard.

ACKNOWLEDGMENTS

This paper is partly based on material from [Graham *et al.* 1979; Lawler 1978B; Lenstra & Rinnooy Kan 1978]. The research by the first author was supported by NSF grant MCS78-20054.

REFERENCES

H.M. ABDEL-WAHAB, T. KAMEDA (1978) Scheduling to minimize maximum cumulative cost subject to series-parallel precedence constraints. *Oper. Res. 26*,151-158.

D. ADOLPHSON, T.C. HU (1973) Optimal linear ordering. *SIAM J. Appl. Math. 25*,403-423.

K.R. BAKER, E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN (1982) Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints. *Oper. Res.*, to appear.

K.R. BAKER, Z.-S. SU (1974) Sequencing with due-dates and early start times to minimize maximum tardiness. *Naval Res. Logist. Quart. 21*,171-176.

P. BRUCKER, M.R. GAREY, D.S. JOHNSON (1977) Scheduling equal-length tasks under tree-like precedence constraints to minimize maximum lateness. *Math. Oper. Res. 2*,275-284.

E.G. COFFMAN, JR., R.L. GRAHAM (1972) Optimal scheduling for two-processor systems. *Acta Informat. 1*,200-213.

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

G.I. DAVIDA, D.J. LINTON (1976) A new algorithm for the scheduling of tree structured tasks. *Proc. Conf. Inform. Sci. and Syst.*,

Baltimore, MD, 543-548.

D. DOLEV (1981) Scheduling wide graphs. Unpublished manuscript.

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

H.N. GABOW (1980) An almost-linear algorithm for two processor scheduling. Technical Report CU-CS-169-80, Department of Computer Science, University of Colorado, Boulder.

M.R. GAREY (1973) Optimal task sequencing with precedence constraints. *Discrete Math. 4*,37-56.

M.R. GAREY, D.S. JOHNSON (1976) Scheduling tasks with nonuniform deadlines on two processors. *J. Assoc. Comput. Mach. 23*,461-467.

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

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

M.R. GAREY, D.S. JOHNSON, B.B. SIMONS, R.E. TARJAN (1981A) Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM J. Comput. 10*,256-269.

M.R. GAREY, D.S. JOHNSON, R.E. TARJAN, M. YANNAKAKIS (1981B) Scheduling opposing forests. Unpublished manuscript.

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

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

W.A. HORN (1972) Single-machine job sequencing with treelike precedence ordering and linear delay penalties. *SIAM J. Appl. Math. 23*,189-202.

T.C. HU (1961) Parallel sequencing and assembly line problems. *Oper. Res. 9*,841-848.

J.R. JACKSON (1955) Scheduling a production line to minimize maximum tardiness. Research Report 43, Management Science Research Project, University of California, Los Angeles.

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

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

B.J. LAGEWEG, E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN (1981A) Computer aided complexity classification of combinatorial problems. Report BW 137, Mathematisch Centrum, Amsterdam.

B.J. LAGEWEG, E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN (1981B) Computer aided complexity classification of deterministic scheduling problems. Report BW 138, Mathematisch Centrum, Amsterdam.

B.J. LAGEWEG, J.K. LENSTRA, A.H.G. RINNOOY KAN (1976) Minimizing maximum lateness on one machine: computational experience and some applications. *Statist. Neerlandica 30*,25-41.

E.L. LAWLER (1973) Optimal sequencing of a single machine subject to precedence constraints. *Management Sci. 19*,544-546.

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

E.L. LAWLER (1978A) Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Ann. Discrete Math. 2*,75-90.

E.L. LAWLER (1978B) Sequencing problems with series parallel precedence constraints. *Proc. Summer School on Combinatorial Optimization*, Urbino, Italy, July 1978, to appear.

E.L. LAWLER (1982) Preemptive scheduling of precedence-constrained jobs on parallel machines. In: M.A.H. DEMPSTER, J.K. LENSTRA, A.H.G. RINNOOY KAN (eds.) (1982) *Deterministic and Stochastic Scheduling*, Reidel, Dordrecht.

E.L. LAWLER, H.W. LENSTRA, A.H.G. RINNOOY KAN, T.J. WANSBEEK (eds.) (1976) *Een Tuyltje Boskruyd: Essays in Honor of Dr. J.-K. Lenstra*, Paris/Amsterdam/Delft/Leyden.

E.L. LAWLER, B.D. SIVAZLIAN (1978) Minimization of time varying costs in single machine sequencing. *Oper. Res. 26*,563-569.

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

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

J.K. LENSTRA, A.H.G. RINNOOY KAN, P. BRUCKER (1977) Complexity of machine scheduling problems. *Ann. Discrete Math. 1*,343-362.

C.L. MONMA, J.B. SIDNEY (1979) Sequencing with series-parallel precedence constraints. *Math. Oper. Res. 4*,215-224.

J.B. SIDNEY (1975) Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs. *Oper. Res. 23*,283-298.

J.B. SIDNEY (1979) The two-machine maximum flow time problem with series parallel precedence relations. *Oper. Res. 27*,782-791.

B. SIMONS (1978) A fast algorithm for single processor scheduling. *Proc. 19th Annual IEEE Symp. Foundations of Computer Science*, 246-252.

W.E. SMITH (1956) Various optimizers for single-stage production. *Naval Res. Logist. Quart. 3*,59-66.

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

J. VALDES, R.E. TARJAN, E.L. LAWLER (1981) The recognition of series parallel digraphs. *SIAM J. Comput.*, to appear.

M.K. WARMUTH (1980) M Processor unit-execution-time scheduling reduces to M-1 weakly connected components. M.S. Thesis, Department of Computer Science, University of Colorado, Boulder.