Annals of Discrete Mathematics 1 (1977) 343–362

COMPLEXITY OF MACHINE SCHEDULING PROBLEMS

J.K. LENSTRA
Mathematisch Centrum, Amsterdam, The Netherlands

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

P. BRUCKER
Universität Oldenburg, G.F.R.

We survey and extend the results on the complexity of machine scheduling problems. After a brief review of the central concept of NP-completeness we give a classification of scheduling problems on single, different and identical machines and study the influence of various parameters on their complexity. The problems for which a polynomial-bounded algorithm is available are listed and NP-completeness is established for a large number of other machine scheduling problems. We finally discuss some questions that remain unanswered.

1. Introduction

In this paper we study the complexity of machine scheduling problems. Section 2 contains a brief review of recent relevant developments in the theory of computational complexity, centering around the concept of NP-completeness. A classification of machine scheduling problems is given in Section 3. In Section 4 we present the results on the complexity of these problems: a large number of them turns out to be NP-complete. Quite often a minor change in some parameter transforms an NP-complete problem into one for which a polynomial-bounded algorithm is available. Thus, we have obtained a reasonable insight into the location of the borderline between "easy" and "hard" machine scheduling problems, although some questions remain open. They are briefly discussed in Section 5.

2. Complexity theory

Recent developments in the theory of computational complexity as applied to combinatorial problems have aroused the interest of many researchers. The main credit for this must go to S.A. Cook [7] and R.M. Karp [25], who first explored the relation between the classes $\mathcal{P}$ and $\mathcal{NP}$ of (language recognition) problems solvable by *deterministic* and *non-deterministic* Turing machines respectively, in a number of steps *bounded by a polynomial in the length of the input.* With respect to

combinatorial optimization, we do not really require mathematically rigorous definitions of these concepts; for our purposes we may safely identify $\mathcal{P}$ with the class of problems for which a *polynomial-bounded, good* [8] or *efficient algorithm* exists, whereas all problems in $\mathcal{NP}$ can be solved by *polynomial-depth backtrack search.*

In this context, all problems are stated in terms of *recognition* problems which require a yes/no answer. In order to deal with the complexity of a combinatorial *minimization* problem, we transform it into the problem of determining the existence of a solution with value at most equal to y, for some *threshold* y.

It is clear that $\mathcal{P} \subset \mathcal{NP}$, and the question arises if this inclusion is a proper one or if, on the contrary, $\mathcal{P} = \mathcal{NP}$. Although this is still an open problem, the equality of $\mathcal{P}$ and $\mathcal{NP}$ is considered to be very unlikely and most bets (e.g., in [28]) have been going in the other direction. To examine the consequences of an affirmative answer to the $\mathcal{P} = \mathcal{NP}$ question, we introduce the following concepts.

Problem P′ is *reducible* to problem P (notation: $P' \propto P$) if for any instance of P′ an instance of P can be constructed in polynomial-bounded time such that solving the instance of P will solve the instance of P′ as well.

P′ and P are *equivalent* if $P' \propto P$ and $P \propto P'$.

P is NP-*complete* [28] if $P \in \mathcal{NP}$ and $P' \propto P$ for every $P' \in \mathcal{NP}$. Informally, the reducibility of P′ to P implies that P′ can be considered as a special case of P; the NP-completeness of P indicates that P is, in a sense, the most difficult problem in $\mathcal{NP}$.

In a remarkable paper [7], NP-completeness was established with respect to the so-called Satisfiability problem. This problem can be formulated as follows.

Given clauses $C_1, \ldots, C_u$, *each being a disjunction of literals from the set* $X = \{x_1, \ldots, x_t, \bar{x}_1, \ldots, \bar{x}_t\}$, *is the conjunction of the clauses satisfiable, i.e., does there exist a subset* $S \subset X$ *such that* S *does not contain a complementary pair of literals* $(x_i, \bar{x}_i)$, *and* $S \cap C_j \neq \emptyset$ *for* $j = 1, \ldots, u$?

Cook proved this result by specifying a polynomial-bounded "master reduction" which, given $P \in \mathcal{NP}$, constructs for any instance of P an equivalent boolean expression in conjunctive normal form. By means of this reduction, a polynomial-bounded algorithm for the Satisfiability problem could be used to construct a polynomial-bounded algorithm for any problem in $\mathcal{NP}$. It follows that

$\mathcal{P} = \mathcal{NP}$ *if and only if* Satisfiability $\in \mathcal{P}$.

The same argument applies if we replace Satisfiability by any NP-complete problem. A large number of such problems has been identified by Karp [25; 26] and others (e.g., [17]); Theorem 1 mentions some of them. Since they are all notorious combinatorial problems for which typically no good algorithms have been found so far, these results afford strong circumstantial evidence that $\mathcal{P}$ is a proper subset of $\mathcal{NP}$.

Theorem 1. *The following problems are* NP*-complete*:

(a) Clique. *Given an undirected graph* $G = (V, E)$ *and an integer* k, *does* G *have a clique* (*i.e., a complete subgraph*) *on* k *vertices*?

(b) Linear arrangement. *Given an undirected graph* $G = (V, E)$ *and an integer* k, *does there exist a one-to-one function* $\pi : V \to \{1, \ldots, |V|\}$ *such that* $\sum_{(i,j)\in E} |\pi(i) - \pi(j)| \leq k$?

(c) Directed hamiltonian circuit. *Given a directed graph* $G = (V, A)$, *does* G *have a hamiltonian circuit* (*i.e., a directed cycle passing through each vertex exactly once*)?

(d) Directed hamiltonian path. *Given a directed graph* $G' = (V', A')$, *does* G' *have a hamiltonian path* (*i.e., a directed path passing through each vertex exactly once*)?

(e) Partition. *Given positive integers* $a_1, \ldots, a_t$, *does there exist a subset* $S \subset T = \{1, \ldots, t\}$ *such that* $\sum_{i\in S} a_i = \sum_{i\in T-S} a_i$?

(f) Knapsack. *Given positive integers* $a_1, \ldots, a_t, b$, *does there exist a subset* $S \subset T = \{1, \ldots, t\}$ *such that* $\sum_{i\in S} a_i = b$?

(g) 3-Partition. *Given positive integers* $a_1, \ldots, a_{3t}, b$, *does there exist a partition* $(T_1, \ldots, T_t)$ *of* $T = \{1, \ldots, 3t\}$ *such that* $|T_i| = 3$ *and* $\sum_{i\in T_j} a_i = b$ *for* $j = 1, \ldots, t$?

Proof. (a) See [7; 25].

(b) See [17].

(c, e, f) See [25].

(d) NP-completeness of this problem is implied by two observations:

(A) Directed hamiltonian path $\in \mathcal{NP}$;

(B) P $\propto$ Directed hamiltonian path for some NP-complete problem P.

(A) is trivially true, and (B) is proved by the following reduction.

Directed hamiltonian circuit $\propto$ Directed hamiltonian path.

Given $G = (V, A)$, we choose $v' \in V$ and construct $G' = (V', A')$ with

$$V' = V \cup \{v''\},$$

$$A' = \{(v, w) \mid (v, w) \in A,\ w \neq v'\} \cup \{(v, v'') \mid (v, v') \in A\}.$$

G has a hamiltonian circuit if and only if G' has a hamiltonian path.

(g) See [12]. □

Karp's work has led to a large amount of research on the location of the borderline separating the "easy" problems (in $\mathcal{P}$) from the "hard" (NP-complete) ones. It turns out that a minor change in a problem parameter (notably — for some as yet mystical reason — an increase from two to three) often transforms an easy problem into a hard one. Not only does knowledge of the borderline lead to fresh insights as to what characteristics of a problem determine its complexity, but there are also important consequences with respect to the solution of these problems. Establishing NP-completeness of a problem can be interpreted as a formal

justification to use enumerative methods such as branch-and-bound, since no substantially better method is likely to exist. Embarrassing incidents such as the presentation in a standard text-book of an enumerative approach to the undirected Chinese postman problem, for which a good algorithm had already been developed in [9], will then occur less readily.

The class of machine scheduling problems seems an especially attractive object for this type of research, since their structure is relatively simple and there exist standard problem parameters that have demonstrated their usefulness in previous research.

Before describing this class of problems, let us emphasize that membership of $\mathcal{P}$ versus NP-completeness only yields a very coarse measure of complexity. On one hand, the question has been raised whether polynomial-bounded algorithms are really good [2]. On the other hand, there are significant differences in complexity within the class of NP-complete problems.

One possible refinement of the complexity measure may be introduced at this stage. It is based on the way in which the problem data are encoded. Taking the Knapsack and 3-Partition problems as examples and defining $a_* = \max_{i \in T}\{a_i\}$, we observe that the length of the input is $O(t \log a_*)$ in the standard *binary* encoding, and $O(ta_*)$ if a *unary* encoding is allowed. 3-Partition has been proved NP-complete even with respect to a unary encoding [12]. Knapsack is NP-complete with respect to a binary encoding [25], but solution by dynamic programming requires $O(tb)$ steps and thus yields a polynomial-bounded algorithm with respect to a unary encoding; similar situations exist for several machine scheduling problems. Such "pseudopolynomial" algorithms [35] need not necessarily be "good" in the practical sense of the word, but it may pay none the less to distinguish between complexity results with respect to unary and binary encodings (cf. [16]). *Unary* NP-*completeness* or *binary membership of* $\mathcal{P}$ would then be the strongest possible result, and it is quite feasible for a problem to be *binary* NP-*complete* and to allow a *unary polynomial-bounded* solution. The results in this paper hold with respect to the standard binary encoding; some consequences of using a unary encoding will be pointed out as well.

3. Classification

Machine scheduling problems can be verbally formulated as follows [6; 45]:
A *job* J_i $(i = 1, \ldots, n)$ consists of a sequence of *operations*, each of which corresponds to the uninterrupted processing of J_i on some *machine* M_k $(k = 1, \ldots, m)$ during a given period of time. Each machine can handle at most one job at a time. What is according to some overall *criterion* the optimal *processing order* on each machine?

The following data can be specified for each J_i:
a *number of operations* n_i;

a *machine order* ν_i, i.e. an ordered n_i-tuple of machines;

a *processing time* p_{ik} of its kth operation, $k = 1, \ldots, n_i$ (if $n_i = 1$ for all J_i, we shall usually write p_i instead of p_{i1});

a *weight* w_i;

a *release date* or *ready time* r_i, i.e. its earliest possible starting time (unless stated otherwise, we assume that $r_i = 0$ for all J_i);

a *due date* or *deadline* d_i;

a *cost function* $f_i : \mathbf{N} \to \mathbf{R}$, indicating the costs incurred as a nondecreasing function of the completion time of J_i.

We assume that all data (except ν_i and f_i) are nonnegative integers. Given a processing order on each M_k, we can compute for each J_i:

the *starting time* S_i;

the *completion time* C_i;

the *lateness* $L_i = C_i - d_i$;

the *tardiness* $T_i = \max\{0, C_i - d_i\}$;

$U_i = \textit{if } C_i \leqslant d_i \textit{ then } 0 \textit{ else } 1$.

Machine scheduling problems are traditionally classified by means of four parameters n, m, l, κ. The first two parameters are integer variables, denoting the numbers of jobs and machines respectively; the cases in which m is constant and equal to 1, 2, or 3 will be studied separately. If $m > 1$, the third parameter takes on one of the following values:

$l = F$ in a *flow-shop* where $n_i = m$ and $\nu_i = (M_1, \ldots, M_m)$ for each J_i;

$l = P$ in a *permutation flow-shop*, i.e. a flow-shop where passing is not permitted so that each machine has to process the jobs in the same order;

$l = G$ in a (*general*) *job-shop* where n_i and ν_i may vary per job;

$l = I$ in a *parallel-shop* where each job has to be processed on just one of m *identical* machines, i.e. $n_i = 1$ for all J_i and the ν_i are not defined.

Extensions to the more general situation where *several groups of parallel* (*possibly non-identical*) *machines* are available will not be considered.

The fourth parameter indicates the optimality criterion. We will only deal with *regular* criteria, i.e., monotone functions κ of the completion times $C_1, \ldots, C_n$ such that

$$C_i \leqslant C_i' \text{ for all } i \implies \kappa(C_1, \ldots, C_n) \leqslant \kappa(C_1', \ldots, C_n').$$

These functions are usually of one of the following types:

$\kappa = f_{\max} = \max_i \{f_i(C_i)\}$;

$\kappa = \sum f_i = \sum_{i=1}^{n} f_i(C_i)$.

The following specific criteria have frequently been chosen to be minimized:

$\kappa = C_{\max} = \max_i \{C_i\}$;

$\kappa = \sum w_i C_i = \sum_{i=1}^{n} w_i C_i$;

$\kappa = L_{\max} = \max_i \{L_i\}$;

$\kappa = \sum w_i T_i = \sum_{i=1}^{n} w_i T_i$;

$\kappa = \sum w_i U_i = \sum_{i=1}^{n} w_i U_i$.

We refer to [45] for relations between these and other objective functions.

Some relevant problem variations are characterized by the presence of one or more elements from a parameter set λ, such as

prec (precedence constraints between the jobs, where "J_i precedes J_j" (notation: $J_i < J_j$) implies $C_i \leqslant S_j$);

tree (precedence constraints between the jobs such that the associated precedence graph can be given as a *branching*, i.e. a set of directed trees with either indegree or outdegree at most one for all vertices);

$r_i \geqslant 0$ (possibly non-equal release dates for the jobs);

$C_i \leqslant d_i$ (all jobs have to meet their deadlines; in this case we assume that $\kappa \in \{C_{\max}, \sum w_i C_i\}$);

no wait (no waiting time for the jobs between their starting and completion times; hence, $C_i = S_i + \sum_k p_{ik}$ for each J_i);

$n_i \leqslant n_*$ (a constant upper bound on the number of operations per job);

$p_{ik} \leqslant p_*$ (a constant upper bound on the processing times);

$p_{ik} = 1$ (unit processing times);

$w_i = 1$ (equality of the weights; we indicate this case also by writing $\sum C_i, \sum T_i, \sum U_i$).

In view of the above discussion, we can use the notation $n \mid m \mid l, \lambda \mid \kappa$ to indicate specific machine scheduling problems.

4. Complexity of machine scheduling problems

All machine scheduling problems of the type defined in Section 3 can be solved by polynomial-depth backtrack search and thus are members of $\mathcal{NP}$. The results on their complexity are summarized in Table 1.

The problems which are marked by an asterisk (*) are solvable in polynomial-bounded time. In Table 2 we provide for most of these problems references where the algorithm in question can be found; we give also the order of the number of steps in the currently best implementations. The problems marked by a note of exclamation (!) are NP-complete. The reductions to these problems are listed in Table 3. Question-marks (?) indicate open problems. We will return to them in Section 5 to motivate our typographical suggestion that these problems are likely to be NP-complete.

Table 1 contains the "hardest" problems that are known to be in $\mathcal{P}$ and the "easiest" ones that have been proved to be NP-complete. In this respect, Table 1 indicates to the best of our knowledge the location of the borderline between easy and hard machine scheduling problems.

Before proving the theorems mentioned in Table 3, we will give a simple example of the interaction between tables and theorems by examining the status of the general job-shop problem, indicated by $n \mid m \mid G \mid C_{\max}$.

Table 1. Complexity of machine scheduling problems

n jobs	1 machine	2 machines	m machines
C_{max}	* *prec*, $r_i \geqslant 0$	* *F* * *F, no wait* ! *F, tree* ! *F*, $r_i \geqslant 0$	! $m = 3$: *F* ? $m = 3$: *F, no wait* ! *F, no wait*
		* *G*, $n_i \leqslant 2$! *G*, $n_i \leqslant 3$	* $n = 2$: *G* ! $m = 3$: *G*, $n_i \leqslant 2$
		! *I* * *I, prec*, $r_i \geqslant 0$, $C_i \leqslant d_i$, $p_i = 1$! *I, prec*, $p_i \leqslant 2$	* *I, tree*, $p_i = 1$? $m = 3$: *I, prec*, $p_i = 1$! *I, prec*, $p_i = 1$
$\sum w_i C_i$	* *tree* ! *prec*, $p_i = 1$! *prec*, $w_i = 1$! $r_i \geqslant 0$, $w_i = 1$ * $C_i \leqslant d_i$, $w_i = 1$! $C_i \leqslant d_i$	! *F*, $w_i = 1$? *F, no wait*, $w_i = 1$	! *F, no wait*, $w_i = 1$
		! *I* * *I, prec*, $p_i = 1$, $w_i = 1$! *I, prec*, $p_i \leqslant 2$, $w_i = 1$	* *I*, $r_i \geqslant 0$, $p_i = 1$ * *I*, $w_i = 1$! *I, prec*, $p_i = 1$, $w_i = 1$
L_{max}	* *prec* * *prec*, $r_i \geqslant 0$, $p_i = 1$! $r_i \geqslant 0$	! *F* ! *I*	
$\sum w_i T_i$	* $r_i \geqslant 0$, $p_i = 1$? $w_i = 1$! ! *prec*, $p_i = 1$, $w_i = 1$! $r_i \geqslant 0$, $w_i = 1$	! *F*, $w_i = 1$! *I*, $w_i = 1$	
$\sum w_i U_i$	* $r_i \geqslant 0$, $p_i = 1$ * $w_i = 1$! ! *prec*, $p_i = 1$, $w_i = 1$! $r_i \geqslant 0$, $w_i = 1$	! *F*, $w_i = 1$! *I*, $w_i = 1$	

*: problem in $\mathscr{P}$; see Table 2.
?: open problem; see Section 5.
!: NP-complete problem; see Table 3.

Table 2. References to polynomial-bounded algorithms

Problem	References	Order
$n \mid 1 \mid prec,\ r_i \geqslant 0 \mid C_{\max}$	—	$O(n^2)$
$n \mid 1 \mid tree \mid \sum w_i C_i$	[20; 1; 46][a]	$O(n \log n)$
$n \mid 1 \mid C_i \leqslant d_i \mid \sum C_i$	[47]	$O(n \log n)$
$n \mid 1 \mid prec \mid L_{\max}$	[33]	$O(n^2)$
$n \mid 1 \mid prec,\ r_i \geqslant 0,\ p_i = 1 \mid L_{\max}$	[30]	$O(n^2)$
$n \mid 1 \mid r_i \geqslant 0,\ p_i = 1 \mid \sum w_i T_i$	[32]	$O(n^3)$
$n \mid 1 \mid r_i \geqslant 0,\ p_i = 1 \mid \sum w_i U_i$	[29]	$O(n^2)$
$n \mid 1 \parallel \sum U_i$	[41][b]	$O(n \log n)$
$n \mid 2 \mid F \mid C_{\max}$	[24]	$O(n \log n)$
$n \mid 2 \mid F,\ no\ wait \mid C_{\max}$	[18; 44]	$O(n^2)$
$n \mid 2 \mid G,\ n_i \leqslant 2 \mid C_{\max}$	[23]	$O(n \log n)$
$n \mid 2 \mid I,\ prec,\ r_i \geqslant 0,\ C_i \leqslant d_i,\ p_i = 1 \mid C_{\max}$	[14][c]	$O(n^3)$
$n \mid 2 \mid I,\ prec,\ p_i = 1 \mid \sum C_i$	[5; 11]	$O(n^2)$
$2 \mid m \mid G \mid C_{\max}$	[48; 19]	$O(m^2)$
$n \mid m \mid I,\ tree,\ p_i = 1 \mid C_{\max}$	[22]	$O(n)$
$n \mid m \mid I,\ r_i \geqslant 0,\ p_i = 1 \mid \sum w_i C_i$	[32]	$O(n^3)$
$n \mid m \mid I \mid \sum C_i$	[6][d]	$O(n \log n)$

[a] An $O(n \log n)$ algorithm for the more general case of *series parallel* precedence constraints is given in [36].

[b] An $O(n \log n)$ algorithm for the more general case of *agreeable weights* (i.e. $p_i < p_j \Rightarrow w_i \geqslant w_j$) is given in [34].

[c] $O(n^3)$ and $O(n^2)$ algorithms for the $n \mid 2 \mid I,\ prec,\ p_i = 1 \mid C_{\max}$ problem are given in [10] and [5] respectively; see also [13].

[d] Polynomial-bounded algorithms for the more general case of *parallel non-identical* machines are given in [21; 4].

In Table 1, we see that the $n \mid 2 \mid G, n_i \leqslant 2 \mid C_{\max}$ problem is a member of $\mathcal{P}$ and that two minor extensions of this problem, $n \mid 2 \mid G, n_i \leqslant 3 \mid C_{\max}$ and $n \mid 3 \mid G$, $n_i \leqslant 2 \mid C_{\max}$, are NP-complete. By Theorem 2(c, h), these problems are special cases of the general job-shop problem, which is thus shown to be NP-complete by Theorem 2(b). Table 2 refers to an $O(n \log n)$ algorithm [23] for the $n \mid 2 \mid G$, $n_i \leqslant 2 \mid C_{\max}$ problem. Table 3 tells us that reductions of Knapsack to both NP-complete problems are presented in Theorem 4(a, b); the NP-completeness of Knapsack has been mentioned in Theorem 1(f).

Theorem 2 gives some elementary results on reducibility among machine scheduling problems. It can be used to establish either membership of $\mathcal{P}$ or NP-completeness for problems that are, roughly speaking, either not harder than the polynomially solvable ones or not easier than the NP-complete ones in Table 1.

Theorem 2. (a) *If* $n' \mid m' \mid l', \lambda' \mid \kappa' \propto n \mid m \mid l, \lambda \mid \kappa$ *and* $n \mid m \mid l,\ \lambda \mid \kappa \in \mathcal{P}$, *then* $n' \mid m' \mid l',\ \lambda' \mid \kappa' \in \mathcal{P}$.

Table 3. Reductions to NP-complete machine scheduling problems

Reduction	References
Linear arrangement $\propto n \mid 1 \mid prec,\ p_i = 1 \mid \sum w_i C_i$	[36; 38; 40]
Linear arrangement $\propto n \mid 1 \mid prec \mid \sum C_i$	[36; 38; 40]
3-Partition $\propto n \mid 1 \mid r_i \geqslant 0 \mid \sum C_i$	*h.l.*, Theorem 5
Knapsack $\propto n \mid 1 \mid C_i \leqslant d_i \mid \sum w_i C_i$	*h.l.*, Theorem 4(j)
Knapsack $\propto n \mid 1 \mid r_i \geqslant 0 \mid L_{\max}$	*h.l.*, Theorem 4(g)
Knapsack $\propto n \mid 1 \parallel \sum w_i T_i$	*h.l.*, Theorem 4(i)
Clique $\propto n \mid 1 \mid prec,\ p_i = 1 \mid \sum T_i$	[38; 40]
$n \mid 1 \mid r_i \geqslant 0 \mid L_{\max} \propto n \mid 1 \mid r_i \geqslant 0 \mid \sum T_i$	*h.l.*, Theorem 2(j)
Knapsack $\propto n \mid 1 \parallel \sum w_i U_i$	[25]; *h.l.*, Theorem 4(h)
Clique $\propto n \mid 1 \mid prec,\ p_i = 1 \mid \sum U_i$	[13; 38; 40]
$n \mid 1 \mid r_i \geqslant 0 \mid L_{\max} \propto n \mid 1 \mid r_i \geqslant 0 \mid \sum U_i$	*h.l.*, Theorem 2(j)
Knapsack $\propto n \mid 2 \mid F,\ tree \mid C_{\max}$	*h.l.*, Theorem 4(f)
Knapsack $\propto n \mid 2 \mid F,\ r_i \geqslant 0 \mid C_{\max}$	*h.l.*, Theorem 4(d)
Knapsack $\propto n \mid 2 \mid G,\ n_i \leqslant 3 \mid C_{\max}$	*h.l.*, Theorem 4(a)
Partition $\propto n \mid 2 \mid I \mid C_{\max}$	*h.l.*, Theorem 3(a); cf. [4]
Clique $\propto n \mid 2 \mid I,\ prec,\ p_i \leqslant 2 \mid C_{\max}$	[40]; cf. [49]
3-Partition $\propto n \mid 2 \mid F \mid \sum C_i$	[16]
Partition $\propto n \mid 2 \mid I \mid \sum w_i C_i$	*h.l.*, Theorem 3(b); cf. [4]
$n' \mid 2 \mid I,\ prec,\ p_i \leqslant 2 \mid C_{\max} \propto n \mid 2 \mid I,\ prec,\ p_i \leqslant 2 \mid \sum C_i$	*h.l.*, Theorem 2(l); cf. [40]
Knapsack $\propto n \mid 2 \mid F \mid L_{\max}$	*h.l.*, Theorem 4(e)
$n \mid 2 \mid I \mid C_{\max} \propto n \mid 2 \mid I \mid L_{\max}$	*h.l.*, Theorem 2(i)
$n \mid 2 \mid F \mid L_{\max} \propto n \mid 2 \mid F \mid \sum T_i$	*h.l.*, Theorem 2(j)
$n \mid 2 \mid I \mid L_{\max} \propto n \mid 2 \mid I \mid \sum T_i$	*h.l.*, Theorem 2(j)
$n \mid 2 \mid F \mid L_{\max} \propto n \mid 2 \mid F \mid \sum U_i$	*h.l.*, Theorem 2(j)
$n \mid 2 \mid I \mid L_{\max} \propto n \mid 2 \mid I \mid \sum U_i$	*h.l.*, Theorem 2(j)
Knapsack $\propto n \mid 3 \mid F \mid C_{\max}$	*h.l.*, Theorem 4(c)
Directed hamiltonian path $\propto n \mid m \mid F,\ no\ wait \mid C_{\max}$	*h.l.*, Theorem 6(a)
Knapsack $\propto n \mid 3 \mid G,\ n_i \leqslant 2 \mid C_{\max}$	*h.l.*, Theorem 4(b)
Clique $\propto n \mid m \mid I,\ prec,\ p_i = 1 \mid C_{\max}$	[40]; cf. [49]
Directed hamiltonian path $\propto n \mid m \mid F,\ no\ wait \mid \sum C_i$	*h.l.*, Theorem 6(b)
$n' \mid m \mid I,\ prec,\ p_i = 1 \mid C_{\max} \propto n \mid m \mid I,\ prec,\ p_i = 1 \mid \sum C_i$	*h.l.*, Theorem 2(l); cf. [40]

(b) *If* $n' \mid m' \mid l',\ \lambda' \mid \kappa' \propto n \mid m \mid l,\ \lambda \mid \kappa$ *and* $n' \mid m' \mid l',\ \lambda' \mid \kappa'$ *is* NP*-complete, then* $n \mid m \mid l,\ \lambda \mid \kappa$ *is* NP*-complete.*

(c) $n \mid m' \mid l,\ \lambda \mid \kappa \propto n \mid m \mid l,\ \lambda \mid \kappa$ *if* $m' \leqslant m$ *or if* m' *is constant and* m *is variable.*

(d) $n \mid 2 \mid F \mid \kappa$ *and* $n \mid 2 \mid P \mid \kappa$ *are equivalent.*

(e) $n \mid 3 \mid F \mid C_{\max}$ *and* $n \mid 3 \mid P \mid C_{\max}$ *are equivalent.*

(f) $n \mid m \mid F,\ \lambda \mid \kappa \propto n \mid m \mid G,\ \lambda \mid \kappa$.

(g) $n \mid m \mid l,\ \lambda \mid \kappa \propto n \mid m \mid l,\ \lambda \cup \lambda' \mid \kappa$ *if* $\lambda' \subset \{prec, tree,\ r_i \geqslant 0,\ C_i \leqslant d_i\}$.

(h) $n \mid m \mid l,\ \lambda \cup \lambda' \mid \kappa \propto n \mid m \mid l,\ \lambda \mid \kappa$ *if* $\lambda' \subset \{n_i \leqslant n_*,\ p_{ik} \leqslant p_*,\ p_{ik} = 1,\ w_i = 1\}$.

(i) $n \mid m \mid l,\ \lambda \mid C_{\max} \propto n \mid m \mid l,\ \lambda \mid L_{\max}$.

(j) $n \mid m \mid l,\ \lambda \mid L_{\max} \propto n \mid m \mid l,\ \lambda \mid \kappa$ *if* $\kappa \in \{\sum T_i,\ \sum U_i\}$.

(k) $n \mid m \mid l,\ \lambda \mid \sum w_i C_i \propto n \mid m \mid l,\ \lambda \mid \sum w_i T_i$.

(l) $n' \mid m \mid I,\ prec,\ p_i \leqslant p_* \mid C_{\max} \propto n \mid m \mid I,\ prec,\ p_i \leqslant p_* \mid \sum C_i$.

Proof. Let P′ and P denote the problems on the left-hand side and right-hand side respectively.

(a, b) Clear from the definition of reducibility.

(c) Trivial.

(d, e) P′ has an optimal solution with the same processing order on each machine [6; 45].

(f, g, h) In each case P′ obviously is a special case of P.

(i) Given any instance of P′ and a threshold value y', we construct a corresponding instance of P by defining $d_i = y'$ $(i = 1, \ldots, n)$. P′ has a solution with value $\leqslant y'$ if and only if P has a solution with value $\leqslant 0$.

(j) Given any instance of P′ with due dates d_i' $(i = 1, \ldots, n)$ and a threshold value y', we construct a corresponding instance of P by defining $d_i = d_i' + y'$ $(i = 1, \ldots, n)$. P′ has a solution with value $\leqslant y'$ if and only if P has a solution with value $\leqslant 0$.

(k) Take $d_i = 0$ $(i = 1, \ldots, n)$ in P.

(l) Given any instance of P′ and a y', $0 \leqslant y' \leqslant n'p_*$, we construct a corresponding instance of P by defining

$$\begin{aligned} n'' &= (n' - 1)\, y', \\ n &= n' + n'', \\ y &= ny' + \tfrac{1}{2} n''(n'' + 1), \end{aligned}$$

and adding n'' jobs $J_{n'+j}$ $(j = 1, \ldots, n'')$ to P′ with

$$p_{n'+j,1} = 1,$$

$$J_i < J_{n'+j} \quad (i = 1, \ldots, n' + j - 1).$$

Now P′ has a solution with value $\leqslant y'$ if and only if P has a solution with value $\leqslant y$:

$$C_{\max} \leqslant y' \Rightarrow \Sigma C_i \leqslant n'y' + \Sigma_{j=1}^{n''}(y' + j) = y;$$

$$C_{\max} > y' \Rightarrow \Sigma C_i > y' + \Sigma_{j=1}^{n''}(y' + 1 + j) = y. \quad \square$$

Remark. The proof of Theorem 2(c) involves processing times equal to 0, implying that the operations in question require an infinitesimally small amount of time. Whenever these reductions are applied, the processing times can be transformed into strictly positive integers by sufficiently (but polynomially) inflating the problem data. Examples of such constructions can be found in the proofs of Theorem 4(c, d, e, f).

In Theorems 3 to 6 we present a large number of reductions of the form $\mathrm{P} \propto n \mid m \mid l, \lambda \mid \kappa$ by specifying $n \mid m \mid l, \lambda \mid \kappa$ and some y such that P has a solution if and only if $n \mid m \mid l, \lambda \mid \kappa$ has a solution with value $\kappa \leqslant y$. This equivalence is proved for some principal reductions; in other cases, it is trivial or clear from the analogy to a reduction given previously. The NP-completeness of $n \mid m \mid l, \lambda \mid \kappa$ then follows from the NP-completeness of P as established in Theorem 1.

First, we briefly deal with the problems on *identical* machines. Theorem 3 presents two reductions which are simplified versions of the reductions given in [4].

Theorem 3. Partition *is reducible to the following problems*:
(a) $n\,|\,2\,|\,I\,|\,C_{\max}$;
(b) $n\,|\,2\,|\,I\,|\,\sum w_iC_i$.

Proof. *Define* $A = \sum_{i\in T} a_i$.
(a) Partition $\propto n\,|\,2\,|\,I\,|\,C_{\max}$:

$$n = t;$$
$$p_i = a_i \ (i \in T);$$
$$y = \tfrac{1}{2}A.$$

(b) Partition $\propto n\,|\,2\,|\,I\,|\,\sum w_iC_i$:

$$n = t;$$
$$p_i = w_i = a_i \ \ (i \in T);$$
$$y = \sum_{1\leqslant i\leqslant j\leqslant t} a_ia_j - \tfrac{1}{4}A^2.$$

Suppose that $\{J_i \mid i \in S\}$ is assigned to M_1 and $\{J_i \mid i \in T-S\}$ to M_2; let $c = \sum_{i\in S} a_i - \frac{1}{2}A$. Since $p_i = w_i$ for all i, the value of $\sum w_iC_i$ is not influenced by the ordering of the jobs on the machines and only depends on the choice of S [6]:

$$\sum w_iC_i = \kappa(S).$$

It is easily seen (cf. Fig. 1) that

$$\kappa(S) = \kappa(T) - \Big(\sum_{i\in S} a_i\Big)\Big(\sum_{i\in T-S} a_i\Big)$$

$$= \sum_{1\leqslant i\leqslant j\leqslant t} a_ia_j - (\tfrac{1}{2}A + c)(\tfrac{1}{2}A - c) = y + c^2,$$

and it follows that Partition has a solution if and only if this $n\,|\,2\,|\,I\,|\,\sum w_iC_i$ problem has a solution with value $\leqslant y$. □

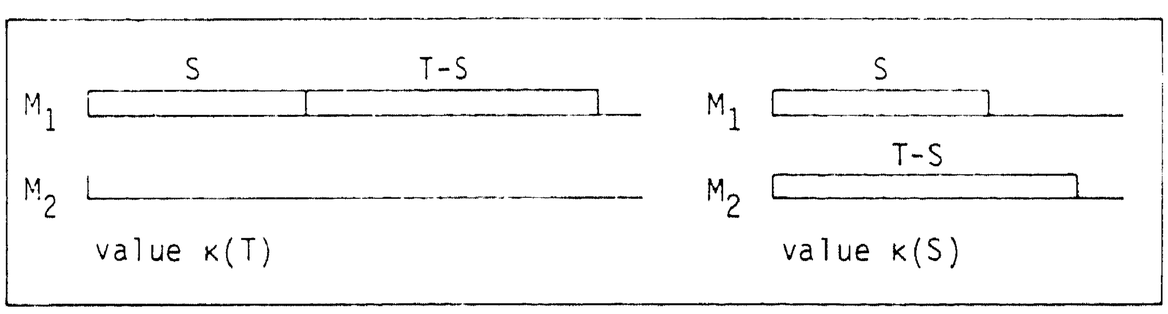

Fig. 1

Most of our results on *different* machines involve the Knapsack problem, as demonstrated by Theorem 4.

Theorem 4. Knapsack *is reducible to the following problems*:

(a) $n \mid 2 \mid G,\ n_i \leqslant 3 \mid C_{\max}$;
(b) $n \mid 3 \mid G,\ n_i \leqslant 2 \mid C_{\max}$;
(c) $n \mid 3 \mid F \mid C_{\max}$;
(d) $n \mid 2 \mid F,\ r_i \geqslant 0 \mid C_{\max}$;
(e) $n \mid 2 \mid F \mid L_{\max}$;
(f) $n \mid 2 \mid F,\ tree \mid C_{\max}$;
(g) $n \mid 1 \mid r_i \geqslant 0 \mid L_{\max}$;
(h) $n \mid 1 \parallel \sum w_i U_i$;
(i) $n \mid 1 \parallel \sum w_i T_i$;
(j) $n \mid 1 \mid C_i \leqslant d_i \mid \sum w_i C_i$.

Proof. Define $A = \sum_{i \in T} a_i$. We may assume that $0 < b < A$.

(a) Knapsack $\propto n \mid 2 \mid G,\ n_i \leqslant 3 \mid C_{\max}$:

$n = t + 1$;
$\nu_i = (M_1),\ p_{i1} = a_i\ \ (i \in T)$;
$\nu_n = (M_2, M_1, M_2),\ p_{n1} = b,\ p_{n2} = 1,\ p_{n3} = A - b$;
$y = A + 1$.

If Knapsack has a solution, then there exists a schedule with value $C_{\max} = y$, as illustrated in Fig. 2. If Knapsack has no solution, then $\sum_{i \in S} a_i - b = c \neq 0$ for each $S \subset T$, and we have for a processing order $(\{J_i \mid i \in S\},\ J_n,\ \{J_i \mid i \in T - S\})$ on M_1 that

$$c > 0 \Rightarrow C_{\max} \geqslant \sum_{i \in S} p_{i1} + p_{n2} + p_{n3} = A + c + 1 > y;$$

$$c < 0 \Rightarrow C_{\max} \geqslant p_{n1} + p_{n2} + \sum_{i \in T - S} p_{i1} = A - c + 1 > y.$$

It follows that Knapsack has a solution if and only if this $n \mid 2 \mid G,\ n_i \leqslant 3 \mid C_{\max}$ problem has a solution with value $\leqslant y$.

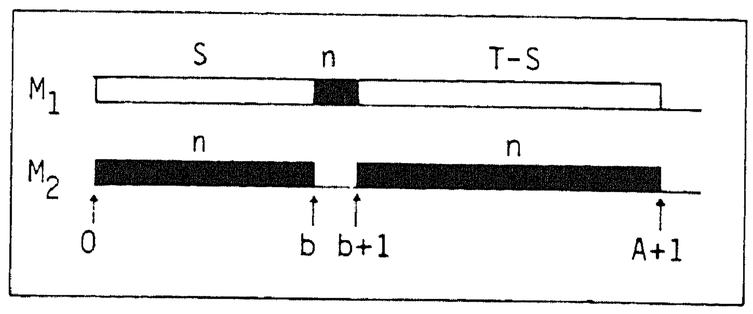

Fig. 2

(b) Knapsack $\propto n \mid 3 \mid G, n_i \leqslant 2 \mid C_{\max}$:

$n = t + 2$;
$\nu_i = (M_1, M_3),\ p_{i1} = p_{i2} = a_i\ \ (i \in T)$;
$\nu_{n-1} = (M_1, M_2),\ p_{n-1,1} = b,\ p_{n-1,2} = 2(A - b)$;
$\nu_n = (M_2, M_3),\ p_{n1} = 2b,\ p_{n2} = A - b$;
$y = 2A$.

If Knapsack has a solution, then there exists a schedule with value $C_{\max} = y$, as illustrated in Fig. 3. If Knapsack has no solution, then $\sum_{i\in S} a_i - b = c \neq 0$ for each $S \subseteq T$, and we have for a processing order $(\{J_i \mid i \in S\}, J_{n-1}, \{J_i \mid i \in T - S\})$ on M_1 that

$$c > 0 \Rightarrow C_{\max} \geqslant \sum_{i\in S} p_{i1} + p_{n-1,1} + p_{n-1,2} = 2A + c > y,$$

$$c < 0 \Rightarrow C_{\max} \geqslant \min\left\{\sum_{i\in S} p_{i1} + p_{n-1,1} + 1, p_{n1}\right\} + p_{n2} + \sum_{i\in T-S} p_{i2} = 2A + 1 > y,$$

which completes the equivalence proof.

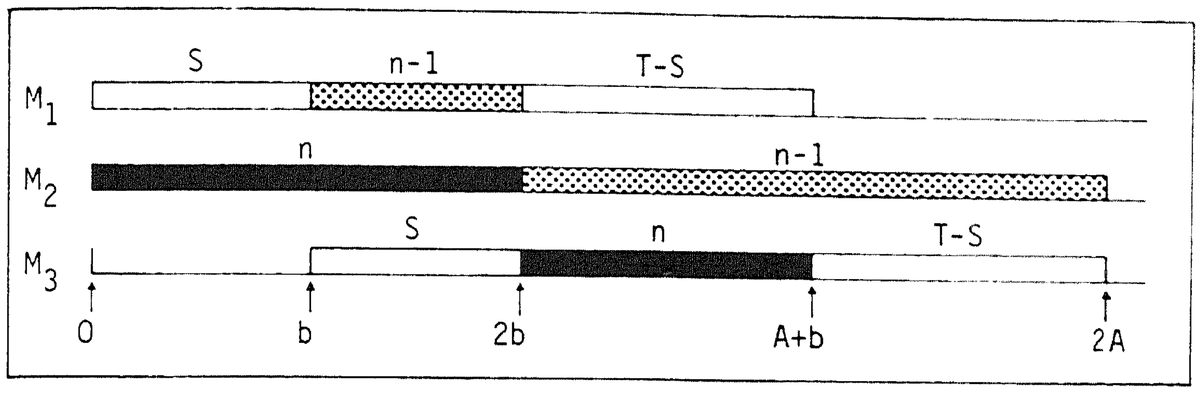

Fig. 3

(c) Knapsack $\propto n \mid 3 \mid F \mid C_{\max}$:

$n = t + 1;$
$p_{i1} = 1,\ p_{i2} = ta_i,\ p_{i3} = 1\ (i \in T);$
$p_{n1} = tb,\ p_{n2} = 1,\ p_{n3} = t(A - b);$
$y = t(A + 1) + 1.$

If Knapsack has a solution, then there exists a schedule with value $C_{\max} = y$, as illustrated in Fig. 4. If Knapsack has no solution, then $\sum_{i\in S} a_i - b = c \neq 0$ for each $S \subset T$, and we have for a processing order $(\{J_i \mid i \in S\}, J_n, \{J_i \mid i \in T - S\})$ that

$$c > 0 \Rightarrow C_{\max} > \sum_{i\in S} p_{i2} + p_{n2} + p_{n3} = t(A + c) + 1 \geqslant y;$$

$$c < 0 \Rightarrow C_{\max} > p_{n1} + p_{n2} + \sum_{i\in T-S} p_{i2} = t(A - c) + 1 \geqslant y.$$

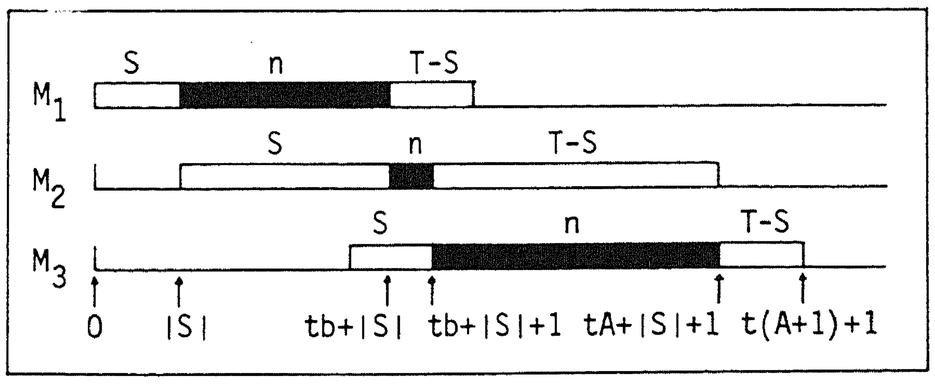

Fig. 4

(d) Knapsack $\propto n\,|\,2\,|\,F,\ r_i \geqslant 0\,|\,C_{\max}$:

$n = t + 1$;
$r_i = 0,\ p_{i1} = ta_i,\ p_{i2} = 1\ (i \in T)$;
$r_n = tb,\ p_{n1} = 1,\ p_{n2} = t(A - b)$;
$y = t(A + 1)$.

Cf. reduction 4(c).

(e) Knapsack $\propto n\,|\,2\,|\,F\,|\,L_{\max}$:

$n = t + 1$;
$p_{i1} = 1,\ p_{i2} = ta_i,\ d_i = t(A + 1)\ (i \in T)$;
$p_{n1} = tb,\ p_{n2} = 1,\ d_n = t(b + 1)$;
$y = 0$.

Cf. reduction 4(c).

(f) Knapsack $\propto n\,|\,2\,|\,F,\ tree\,|\,C_{\max}$:

$n = t + 2$;
$p_{i1} = ta_i,\ p_{i2} = 1\ (i \in T)$;
$p_{n-1,1} = 1,\ p_{n-1,2} = tb$;
$p_{n1} = 1,\ p_{n2} = t(A - b)$;
$J_{n-1} < J_n$;
$y = t(A + 1) + 1$.

We have for a processing order $(\{J_i \mid i \in R\},\ J_{n-1},\ \{J_i \mid i \in S\},\ J_n,\ \{J_i \mid i \in T - S - R\})$ on M_1 that

$$R \neq \emptyset \Rightarrow C_{\max} \geqslant t + p_{n-1,1} + p_{n-1,2} + p_{n1} + p_{n2} = t(A + 1) + 2 > y.$$

The remainder of the equivalence proof is analogous to that of reduction 4(c).

(g) Knapsack $\propto n\,|\,1\,|\,r_i \geqslant 0\,|\,L_{\max}$:

$n = t + 1$;
$r_i = 0,\ p_i = a_i,\ d_i = A + 1\ (i \in T)$;
$r_n = b,\ p_n = 1,\ d_n = b + 1$;
$y = 0$.

Cf. reduction 4(a) and Fig. 5.

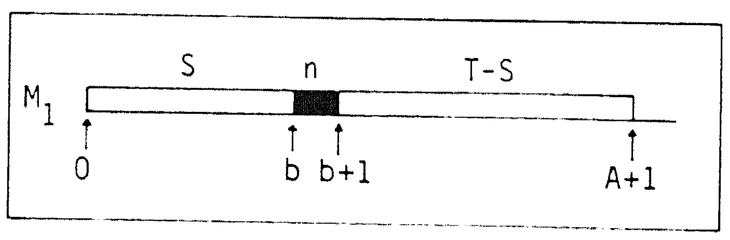

Fig. 5

(h) Knapsack $\propto n\,|\,1\,\|\sum w_iU_i$:

$n = t$;
$p_i = w_i = a_i,\ d_i = b\ (i \in T)$;
$y = A - b$.

Cf. [25] and Fig. 6.

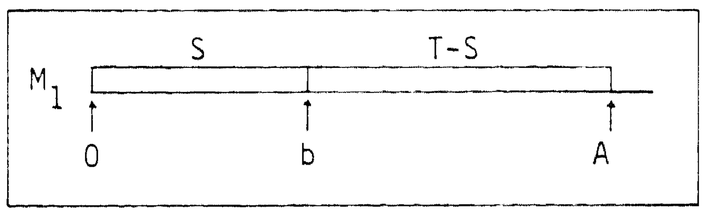

Fig. 6

(i) Knapsack $\propto n \mid 1 \parallel \sum w_i T_i$:

$$n = t + 1;$$
$$p_i = w_i = a_i,\ d_i = 0\ (i \in T);$$
$$p_n = 1,\ w_n = 2,\ d_n = b + 1;$$
$$y = \sum_{1 \leqslant i \leqslant j \leqslant t} a_i a_j + A - b.$$

Cf. Fig. 5. We have for a processing order $(\{J_i \mid i \in S\},\ J_n,\ \{J_i \mid i \in T - S\})$ that $\Sigma_{i \in S} a_i - b = L_n$. Since $p_i = w_i$ and $d_i = 0$ for all $i \in T$, the value of $\Sigma_{i \in T} w_i T_i$ is not influenced by the ordering of S and $T - S$ (cf. the proof of Theorem 3(b)), and we have

$$\begin{aligned}\sum_{i \in T} w_i T_i &= \sum a_i C_i + 2T_n \\ &= \sum_{1 \leqslant i \leqslant j \leqslant t} a_i a_j + \sum_{i \in T - S} a_i + 2\max\{0, L_n\} \\ &= y + |L_n| \geqslant y.\end{aligned}$$

The equivalence follows immediately.

(j) Knapsack $\propto n \mid 1 \mid C_i \leqslant d_i \mid \sum w_i C_i$:

$$n = t + 1;$$
$$p_i = w_i = a_i,\ d_i = A + 1\ (i \in T);$$
$$p_n = 1,\ w_n = 0,\ d_n = b + 1;$$
$$y = \sum_{1 \leqslant i \leqslant j \leqslant t} a_i a_j + A - b.$$

Cf. reduction 4(i) and Fig. 5.

This completes the proof of Theorem 4. □

Theorem 5. 3-Partition *is reducible to* $n \mid 1 \mid r_i \geqslant 0 \mid \sum C_i$.

Proof. A reduction 3-Partition $\propto n \mid 1 \mid r_i \geqslant 0 \mid \sum C_i$ can be obtained by adapting

(a) the transformation of Knapsack to $n \mid 1 \mid r_n \geqslant 0 \mid \sum C_i$, presented in [45];

(b) the reduction 3-Partition $\propto n \mid 2 \mid F \mid \sum C_i$, presented in [16].

Both procedures can be carried out in a straightforward way and lead to essentially the same construction. □

The NP-completeness proofs for the problems with a *no wait* assumption are based on the well-known relation between these problems and the travelling salesman problem (TSP) of finding a minimum weight hamiltonian circuit in the complete directed graph on the vertex set V with weights on the arcs.

Given an $n \mid m \mid F$, *no wait* $\mid \kappa$ problem, we define c_{ij} to be the minimum length of the time interval between S_i and S_j if J_j is scheduled directly after J_i. If we define

$$P_{hk} = \sum_{l=1}^{k} p_{hl}, \tag{1}$$

it is easily proved [43; 44; 50; 39] that

$$c_{ij} = \max_{1 \leq k \leq m} \{P_{ik} - P_{j,k-1}\}. \tag{2}$$

Finding a schedule that minimizes $C_{\max}$ is now equivalent to solving the TSP with $V = \{0, \ldots, n\}$ and weights c_{ij} defined by (2) and by $c_{0h} = 0$, $c_{h0} = P_{hm}$ for $h \neq 0$.

Theorem 6. Directed hamiltonian path *is reducible to the following problems*:
(a) $n \mid m \mid F$, *no wait* $\mid C_{\max}$;
(b) $n \mid m \mid F$, *no wait* $\mid \sum C_i$.

Proof.

(a) Directed hamiltonian path $\propto n \mid m \mid F$, *no wait* $\mid C_{\max}$. Given $G' = (V', A')$, we define

$$n = |V'|,$$
$$m = n(n-1)+2.$$

All jobs have the same machine order $(M_1, M_2, \ldots, M_{m-1}, M_m)$. To each pair of jobs (J_i, J_j) $(i, j = 1, \ldots, n, \quad i \neq j)$ there corresponds one machine $M_k = M_{\kappa(i,j)}$ $(k = 2, \ldots, m-1)$, such that for no J_h some $M_{\kappa(i,h)}$ directly follows an $M_{\kappa(h,j)}$. Such an ordering of the pairs (i, j) can easily be constructed. Due to this property of the ordering, partial sums of the processing times can be defined unambiguously by

$$P_{hk} = \begin{cases} k\mu + \lambda & \text{if } k = \kappa(h,j) \text{ and } (h,j) \in A', \\ k\mu + \lambda + 1 & \text{if } k = \kappa(h,j) \text{ and } (h,j) \notin A', \\ k\mu - \lambda & \text{if } k+1 = \kappa(i,h) \text{ and } (i,h) \in A', \\ k\mu - \lambda - 1 & \text{if } k+1 = \kappa(i,h) \text{ and } (i,h) \notin A', \\ k\mu & \text{otherwise,} \end{cases}$$

for $k = 1, \ldots, m$, $h = 1, \ldots, n$, where

$$\lambda \geq 1,$$
$$\mu \geq 2\lambda + 3.$$

The processing times are given by (cf. (1))

$$p_{h1} = P_{h1},$$
$$p_{hk} = P_{hk} - P_{h,k-1} \ (k = 2, \ldots, m).$$

Through the choice of μ, these processing times are all strictly positive integers.

We can now compute the c_{ij}, as defined by (2). Through the choice of λ, it is immediate that $P_{ik} - P_{j,k-1}$ is maximal for $k = \kappa(i, j)$. Hence,

$$c_{ij} = \begin{cases} \mu + 2\lambda & \text{if } (i,j) \in A', \\ \mu + 2\lambda + 2 & \text{if } (i,j) \notin A'. \end{cases}$$

Since $P_{im} = m\mu$ for all J_i, it now follows that G has a hamiltonian path if and only if this $n \mid m \mid F$, *no wait* $\mid C_{\max}$ problems has a solution with value

$$C_{\max} \leqslant (n-1)(\mu + 2\lambda) + m\mu.$$

(b) Directed hamiltonian path $\propto n \mid m \mid F$, *no wait* $\mid \Sigma C_i$.

G' has a hamiltonian path if and only if the $n \mid m \mid F$, *no wait* $\mid \Sigma C_i$ problem, constructed as in (a), has a solution with value

$$\sum C_i \leqslant \tfrac{1}{2} n(n-1)(\mu + 2\lambda) + nm\mu. \quad \square$$

Let us finally point out some consequences of the use of a unary encoding with respect to the binary NP-complete problems, appearing in Theorems 3 to 6.

The $n \mid 2 \mid I \mid C_{\max}$ and $n \mid 2 \mid I \mid \Sigma w_i C_i$ problems, dealt with in Theorem 3, can be solved in unary polynomial-bounded time by straightforward dynamic programming techniques.

A similar situation exists for the $n \mid 1 \parallel \Sigma w_i U_i$ problem from Theorem 4(h), which can be solved by an $O(n \Sigma p_i)$ algorithm [37]. For most other problems discussed in Theorem 4, however, one can easily prove unary NP-completeness by converting the Knapsack reduction to a 3-Partition reduction. The following adaptation of reduction 4(i) might serve as a typical example (cf. the slightly different construction given in [35]).

3-Partition $\propto n \mid 1 \parallel \Sigma w_i T_i$:

$$n = 4t - 1;$$
$$p_i = w_i = a_i, \ d_i = 0 \ (i \in T);$$
$$p_i = 1, \ w_i = 2, \ d_i = (i - 3t)(b+1) \ (i = 3t+1, \ldots, 4t-1);$$
$$y = \sum_{1 \leqslant i \leqslant j \leqslant 3t} a_i a_j + \tfrac{1}{2} t(t-1) b.$$

Furthermore, reductions of 3-Partition to $n \mid 2 \mid G \mid C_{\max}$ and $n \mid 3 \mid F \mid C_{\max}$ can be found in [16].

With respect to Theorem 5, the situation is different. In the reductions of 3-Partition to $n \mid 1 \mid r_i \geqslant 0 \mid \Sigma C_i$ and $n \mid 2 \mid F \mid \Sigma C_i$, the resulting numbers of jobs are polynomials in both t and b. The (unary) NP-completeness proofs therefore depend essentially on the unary NP-completeness of 3-Partition and no truly polynomial-bounded transformation of Knapsack to these problems is known.

The reductions presented in Theorem 6 clearly prove unary NP-completeness for both *no wait* problems.

5. Concluding remaks

The results presented in Section 4 offer a reasonable insight into the location of the borderline between "easy" and "hard" machine scheduling problems. Computational experience with many problems proved to be NP-complete confirms the impression that a polynomial-bounded algorithm for one and thus for all of them is highly unlikely to exist. As indicated previously, NP-completeness thus functions as a formal justification to use enumerative methods of solution such as branch-and-bound.

Most classical machine scheduling problems have now been shown to be efficiently solvable or NP-complete. Some notable exceptions are indicated by question-marks in Table 1. These open problem are briefly discussed below.

The most notorious one is the $n \mid 1 \parallel \sum T_i$ problem. Extensive investigations have failed to uncover either a polynomial-bounded algorithm or a reduction proving its NP-completeness. The existence of an $O(n^4 \sum p_i)$ algorithm [35] implies that the problem is definitely not *unary* NP-complete. However, we conjecture that it is *binary* NP-complete, which would indicate a major difference between the $\sum T_i$ and $\sum U_i$ problems, as demonstrated by Table 1.

The complexity of the $n \mid 3 \mid F$, *no wait* $\mid C_{\max}$ and $n \mid 2 \mid F$, *no wait* $\mid \sum C_i$ problems is not clear; it is quite possible that both problems are in $\mathcal{P}$. To stimulate research in this direction, we will award an authentic clog to the first scientist who finds a polynomial-bounded algorithm for any one of these problems.

The question of the complexity of the $n \mid 3 \mid I$, *prec*, $p_i = 1 \mid C_{\max}$ problem has been raised already in [49].

Finally, let us stress again that the complexity measure provided by the NP-completeness concept does not capture certain intuitive variations in complexity within the class of NP-complete problems. Note, for example, that an $n \mid 1 \mid r_i \geqslant 0 \mid L_{\max}$ algorithm has figured successfully in a lower bound computation for the $n \mid m \mid G \mid C_{\max}$ problem [3; 31], although both problems are NP-complete and thus equivalent up to a polynomial-bounded transformation. One possible refinement of the complexity measure by means of differentiation between unary and binary encodings has already been discussed. Another indication of a problem's complexity may be based on the analysis of approximation algorithms [15; 27]. For relatively simple NP-complete problems, there often exist heuristics whose performance is arbitrarily close to optimal; on the other hand, there are situations in which even the problem of finding a feasible solution within any fixed percentage from the optimum has been proved NP-complete. Altogether, the development of a measure that allows further distinction within the class of NP-complete problems remains a major research challenge.

Acknowledgements

We gratefully acknowledge the valuable cooperation with B.J. Lageweg, E.L. Lawler and H.W. Lenstra, Jr., and the useful comments by the referee and D.S. Johnson.

References

[1] D. Adolphson and T.C. Hu, Optimal linear ordering, *SIAM J. Appl. Math.* 25 (1973) 403–423.
[2] J.M. Anthonisse and P. van Emde Boas, Are polynomial algorithms really good? Report BW 40, Mathematisch Centrum, Amsterdam, 1974.
[3] P. Bratley, M. Florian and P. Robillard, On sequencing with earliest starts and due dates with application to computing bounds for the $(n/m/G/F_{max})$ problem, *Naval Res. Logist. Quart.* 20 (1973) 57–67.
[4] J. Bruno, E.G. Coffman, Jr. and R. Sethi, Scheduling independent tasks to reduce mean finishing time, *Comm. ACM* 17 (1974) 382–387.
[5] E.G. Coffman, Jr. and R.L. Graham, Optimal scheduling for two-processor systems, *Acta Informat.* 1 (1972) 200–213.
[6] R.W. Conway, W.L. Maxwell and L.W. Miller, *Theory of Scheduling* (Addison-Wesley, Reading, MA, 1967).
[7] S.A. Cook, The complexity of theorem-proving procedures, *Proc. 3rd Annual ACM Symp. Theory Comput.* (1971) 151–158.
[8] J. Edmonds, Paths, trees, and flowers, *Canad. J. Math.* 17 (1965) 449–467.
[9] J. Edmonds, The Chinese postman's problem, *Operations Res.* 13 Suppl. 1 (1965) B73.
[10] M. Fujii, T. Kasami and K. Ninomiya, Optimal sequencing of two equivalent processors, *SIAM J. Appl. Math.* 17 (1969) 784–789; Erratum, 20 (1971) 141.
[11] M.R. Garey, Private communication, 1975.
[12] M.R. Garey and D.S. Johnson, Complexity results for multiprocessor scheduling under resource constraints, *SIAM J. Comput.* 4 (1975) 397–411.
[13] M.R. Garey and D.S. Johnson, Scheduling tasks with nonuniform deadlines on two processors, *J. Assoc. Comput. Mach.* 23 (1976) 461–467.
[14] M.R. Garey and D.S. Johnson, Two-processor scheduling with start-times and deadlines, to appear.
[15] M.R. Garey and D.S. Johnson, Approximation algorithms for combinatorial problems: an annotated bibliography, in: J.F. Traub, ed., *Algorithms and Complexity: New Directions and Recent Results.* (Academic Press, New York, 1976) 41–52.
[16] M.R. Garey, D.S. Johnson and R. Sethi, The complexity of flowshop and jobshop scheduling, *Math. Operations Res.* 1 (1976) 117–129.
[17] M.R. Garey, D.S. Johnson and L. Stockmeyer, Some simplified *NP*-complete graph problems, *Theoret. Comput. Sci.* 1 (1976) 237–267.
[18] P.C. Gilmore and R.E. Gomory, Sequencing a one-state variable machine: a solvable case of the traveling salesman problem, *Operations Res.* 12 (1964) 655–679.
[19] W.W. Hardgrave and G.L. Nemhauser, A geometric model and a graphical algorithm for a sequencing problem, *Operations Res.* 11 (1963) 889–900.
[20] W.A. Horn, Single-machine job sequencing with treelike precedence ordering and linear delay penalties, *SIAM J. Appl. Math.* 23 (1972) 189–202.
[21] W.A. Horn, Minimizing average flow time with parallel machines, *Operations Res.* 21 (1973) 846–847.
[22] T.C. Hu, Parallel sequencing and assembly line problems, *Operations Res.* 9 (1961) 841–848.
[23] J.R. Jackson, An extension of Johnson's results on job lot scheduling, *Naval Res. Logist. Quart.* 3 (1956) 201–203.
[24] S.M. Johnson, Optimal two- and three-stage production schedules with setup times included, *Naval Res. Logist. Quart.* 1 (1954) 61–68.

[25] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller and J.W. Thatcher, eds., *Complexity of Computer Computations* (Plenum Press, New York, 1972) 85–103.
[26] R.M. Karp, On the computational complexity of combinatorial problems, *Networks* 5 (1975) 45–68.
[27] R.M. Karp, The fast approximate solution of hard combinatorial problems, *Proc. 6th Southeastern Conf. Combinatorics, Graph Theory, and Computing* (1976) 15–31.
[28] D.E. Knuth, A terminological proposal, *SIGACT News* 6.1 (1974) 12–18.
[29] B.J. Lageweg and E.L. Lawler, Private communication, 1975.
[30] B.J. Lageweg, J.K. Lenstra and A.H.G. Rinnooy Kan, Minimizing maximum lateness on one machine : computational experience and some applications, *Statistica Neerlandica* 30 (1976) 25–41.
[31] B.J. Lageweg, J.K. Lenstra and A.H.G. Rinnooy Kan, Job-shop scheduling by implicit enumeration, *Management Sci.*, to appear.
[32] E.L. Lawler, On scheduling problems with deferral costs, *Mangement Sci.* 11 (1964) 280–288.
[33] E.L. Lawler, Optimal sequencing of a single machine subject to precedence constraints, *Management Sci.* 19 (1973) 544–546.
[34] E.L. Lawler, Sequencing to minimize the weighted number of tardy jobs, *Rev. Française Automat. Informat. Recherche Opérationnelle* 10.5 Suppl. (1976) 27–33.
[35] E.L. Lawler, A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness, *Ann. Discrete Math.* 1 (1977) 331–342.
[36] E.L. Lawler, Sequencing jobs to minimize total weighted completion time subject to precedence constraints, *Ann. Discrete Math.*, to appear.
[37] E.L. Lawler and J.M. Moore, A functional equation and its application to resource allocation and sequencing problems, *Management Sci.* 16 (1969) 77–84.
[38] J.K. Lenstra, *Sequencing by Enumerative Methods*, Mathematical Centre Tract 69 (Mathematisch Centrum, Amsterdam, 1977).
[39] J.K. Lenstra and A.H.G. Rinnooy Kan, Some simple applications of the travelling salesman problem, *Operational Res. Quart.* 26 (1975) 717–733.
[40] J.K. Lenstra and A.H.G. Rinnooy Kan, Complexity of scheduling under precedence constraints, *Operations Res.*, to appear.
[41] J.M. Moore, An n job, one machine sequencing algorithm for minimizing the number of late jobs, *Management Sci.* 15 (1968) 102–109.
[42] H. Müller-Merbach, *Optimale Reihenfolgen* (Springer, Berlin, 1970).
[43] J. Piehler, Ein Beitrag zum Reihenfolgeproblem, *Unternehmensforschung* 4 (1960) 138–142.
[44] S.S. Reddi and C.V. Ramamoorthy, On the flow-shop sequencing problem with no wait in process, *Operational Res. Quart.* 23 (1972) 323–331.
[45] A.H.G. Rinnooy Kan, *Machine Scheduling Problems: Classification, Complexity and Computations* (Nijhoff, The Hague, 1976).
[46] J.B. Sidney, Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs, *Operations Res.* 23 (1975) 283–298.
[47] W.E. Smith, Various optimizers for single-stage production, *Naval Res. Logist. Quart.* 3 (1956) 59–66.
[48] W. Szwarc, Solution of the Akers–Friedman scheduling problem, *Operations Res.* 8 (1960) 782–788.
[49] J.D. Ullman, *NP*-complete scheduling problems, *J. Comput. System Sci.* 10 (1975) 384–393.
[50] D.A. Wismer, Solution of the flowshop-scheduling problem with no intermediate queues, *Operations Res.* 20 (1972) 689–697.