MATHEMATICS OF OPERATIONS RESEARCH
Vol. 1, No. 2, May, 1976
Printed in U.S.A.

THE COMPLEXITY OF FLOWSHOP AND JOBSHOP SCHEDULING*

M. R. GAREY,† D. S. JOHNSON† AND RAVI SETHI**

NP-complete problems form an extensive equivalence class of combinatorial problems for which no nonenumerative algorithms are known. Our first result shows that determining a shortest-length schedule in an m-machine flowshop is *NP*-complete for $m \geqslant 3$. (For $m = 2$, there is an efficient algorithm for finding such schedules.) The second result shows that determining a minimum mean-flow-time schedule in an m-machine flowshop is *NP*-complete for every $m \geqslant 2$. Finally we show that the shortest-length schedule problem for an m-machine jobshop is *NP*-complete for every $m \geqslant 2$. Our results are strong in that they hold whether the problem size is measured by number of tasks, number of bits required to express the task lengths, or by the sum of the task lengths.

1. Introduction. A *flowshop* is an ordered set of processors $P = \langle P_1, P_2, \ldots, P_m \rangle$ such that the first operation of each job is performed on processor P_1, the second on processor P_2, and so on, until the job completes execution on P_m. A prime example of a flowshop is an assembly line, where the workers on the line are the processors. In this paper we examine the problem of scheduling a collection of jobs on a flowshop so as to optimize such measures as the average finishing time and the maximum finishing time. Such problems have received considerable attention in the literature [2], [4], [5], [10], [13], [17], [21]. However, except for a noted special case algorithm due to S. M. Johnson [10], there has been a pronounced absence of efficient schedule optimization algorithms for flowshops. We shall help to explain this lack by proving results about the inherent computational complexity of such scheduling problems.

Formally, a particular instance of an m-machine flowshop problem consists of a collection $\mathbf{C}$ of *chains* (or "jobs"), where each chain $C \in \mathbf{C}$ is a sequence of *tasks* $C[1], C[2], \ldots, C[m]$, and each task $C[i]$ has a *length* $\tau(C[i])$, which for our purposes may be assumed to be a nonnegative integer.

A "schedule" for a set of chains specifies the time at which each task is to begin processing. The schedule must ensure that (a) once a task begins execution it continues until it finishes, (b) no processor executes more than one task at a time, (c) task $C[i]$ must be completed by P_i before task $C[i+1]$ can begin execution on processor P_{i+1}, and of course (d) no task is executed by more than one processor at a time.

Formally, a *schedule* for a collection of chains $\mathbf{C}$ is a function $S : \{C[i] \mid C \in \mathbf{C}, 1 \leqslant i \leqslant m\} \to Z^+$ (the nonnegative integers) which gives the *starting time* of each task on its appropriate processor under the following two constraints:

(i) once task $C[i]$ starts at time $S(C[i])$, it remains as the only task on processor P_i for the next $\tau(C[i])$ units, i.e., for all $t \geqslant 0$ and all i, $1 \leqslant i \leqslant m$,

$$|\{C \mid S(C[i]) < t < S(C[i]) + \tau(C[i])\}| \leqslant 1,$$

and

(ii) *chain constraint*: for all $C \in \mathbf{C}$ and all i, $1 \leqslant i \leqslant m$, task $C[i+1]$ may not start

* Received June 24, 1975.
AMS 1970 subject classification. Primary 90B35. Secondary 68A20, 90B30.
IAOR 1973 subject classification. Main: Scheduling. Cross references: Computational analysis, production.
Key words. Jobshop scheduling, computational complexity, *NP*-complete.
† Bell Laboratories, Murray Hill, New Jersey.
** The Pennsylvania State University.

for at least $\tau(C[i])$ units after $C[i]$ starts i.e.,

$$S(C[i]) + \tau(C[i]) \leq S(C[i+1]).$$

The *finishing time* for task $C[i]$ under schedule S, denoted $f_S(C[i])$, is defined by $f_S(C[i]) = S(C[i]) + \tau(C[i])$. Using f_S we can define our two schedule performance measures. The *length* (or "makespan") of a schedule S for $\mathbf{C}$ is the maximum finishing time:

$$\text{length}(S) = \max\{ f_S(C[m]) \mid C \in \mathbf{C}\}.$$

The *mean flow time* of S is defined to be

$$\text{MFT}(S) = \sum_{C \in \mathbf{C}} f_S(C[m]).$$

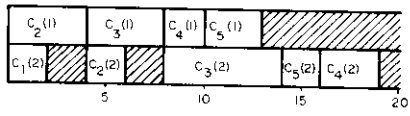
(Note that MFT(S) is not really the average finishing time, which is given by MFT(S)/$|\mathbf{C}|$. However, minimizing MFT(S) is equivalent to minimizing the average finishing time.)

An example of a collection $\mathbf{C}$ of chains for a 2-machine flowshop and a schedule for it are given in Figure 1. It will be convenient to use $\tau(C)$ to denote $\langle \tau(C[1]), \tau(C[2]), \ldots, \tau(C[m])\rangle$ with an analogous meaning for $S(C)$. Note that we do allow tasks with $\tau(C[i]) = 0$. By convention, these tasks do not need to be executed on processor P_i. In the diagram the rectangles represent the individual tasks and the time during which they occupy the corresponding processor. The cross-hatched rectangles represent times when a particular processor is idle. For the schedule S given in the figure we have length(S) = 19, MFT(S) = 57.

The problem of finding a minimum-length schedule for a 2-machine flowshop has been solved by S. M. Johnson [10]. The algorithm needs time at most proportional to $n \cdot \log n$, where n is the number of chains. This is essentially the most complex of the flowshop and jobshop problems for which an efficient optimization algorithm is known.

We shall provide justification for the intransigence of the more complicated flowshop and jobshop problems by proving that they belong to the class of "*NP*-complete" problems. For two reasons, it is widely believed that all problems in this class are computationally intractable. First, no "efficient" algorithm is known for *any* problem in this class. Second, if any single problem in this class could be solved with an efficient algorithm, then every other problem in the class would also be solvable with an efficient algorithm. Scheduling problems already known to be "*NP*-complete" can be found in [3], [6], [11], [12], [14], [23], [24]. Further members of this class, including such diverse problems as the traveling salesman problem, the knapsack problem, and the graph chromatic number problem, can be found in [1], [7], [11], [12], [18], [20].

We shall not recapitulate here all the technical details involved in demonstrating



$C = \{C_1, C_2, C_3, C_4, C_5\}$

	C_1	C_2	C_3	C_4	C_5
$\tau(\cdot)$	⟨0,2⟩	⟨4,2⟩	⟨4,6⟩	⟨2,3⟩	⟨3,2⟩
$S(\cdot)$	⟨0,0⟩	⟨0,4⟩	⟨4,8⟩	⟨8,16⟩	⟨10,14⟩
$f_S(\cdot)$	⟨0,2⟩	⟨4,6⟩	⟨8,14⟩	⟨10,19⟩	⟨13,16⟩

FIGURE 1. A flowshop example.

NP-completeness [1], [11], [24], but shall informally specify just those ideas needed to follow our proofs. One of these ideas is the precise meaning of "efficient," as used above. An algorithm will be called "efficient" if it operates in time bounded by a polynomial in the length of its input. In other words, there is a constant c such that the algorithm never takes more time than $O(n^c)$ on an input of length n.

Efficiency consideration leads us to a discussion of what we mean by "length of input." For our flowshop problem, the technical definition would say "the number of bits required to completely specify **C**." Let us call this measure $m_1(\mathbf{C})$. A second measure, which will be of central importance in this paper, is the *sum* of all the task lengths, $m_2(\mathbf{C}) = \sum_{c \in \mathbf{C}} \sum_{i=1}^{m} \tau(C[i])$. Note that in general we have $m_1(\mathbf{C}) < m_2(\mathbf{C})$.

REMARK. The running times of algorithms are often expressed in terms of a third parameter of **C** the number n of chains contained in **C**. A statement that an algorithm takes time $f(n)$, however, contains the implicit assumption that numbers, such as task lengths, can be read and operated on in fixed times, independent of the size of the number. The assumption is only realistic for relatively small numbers (the actual size depending on the particular computer word length used), and in such cases n will be proportional to $m_1(\mathbf{C})$.

It is usual in proving *NP*-completeness of a problem to measure input length according to m_1. For scheduling problems, however, we have reservations about this approach. It is possible for a problem to be *NP*-complete when input is measured by m_1, and yet still be solvable in polynomial time when input is measured by m_2.

For instance, the problem of determining whether a collection of independent tasks can be scheduled to meet a given deadline on two identical processors is *NP*-complete with input measure m_1. However, the same problem can be solved in time proportional to the sum of the task lengths times the number of tasks, using dynamic programming. (See for example [9], [19].) These remarks do not violate our claim that no efficient algorithm is known for any *NP*-complete problem. Since the integer n can be represented using only $\log n$ bits, the sum of the task lengths can itself be an exponential function of the number of bits required to represent task lengths. Thus the above mentioned algorithm, while polynomial in measure m_2, can take exponential time (hence time not bounded by any polynomial) if input length is measured by m_1.

On the other hand, the algorithm may *still* be usable. We might well be willing to spend time a low-order polynomial in the length of the desired schedule in order to find a schedule of minimum length. Thus an *NP*-completeness result using measure m_1 may still leave room for useful algorithms if m_2 is the relevant measure of input length. To essentially foreclose even this possibility, we have to prove that the problem under consideration is *NP*-complete, even when input length is measured by m_2. *NP*-completeness under measure m_1 follows as a consequence. Results using measure m_2 first appeared in [6] and it is this measure we will use in the sequel.

In §2, after first presenting a brief review of how *NP*-completeness is demonstrated, we show that the problem of determining minimum length schedules on a 3-machine flowshop is *NP*-complete. This result is true even if input length is measured by m_2, and is to be contrasted to the fact that the 2-machine flowshop schedule-length minimization problem can be solved quite efficiently using S. M. Johnson's algorithm.

In §3, we prove the *NP*-completeness of the problem of determining minimum mean-flow-time schedules for a 2-machine flowshop, even with input length measured by m_2. In §4 we consider related problems, in particular showing that the jobshop schedule-length minimization problem is *NP*-complete for 2 or more machines.

2. The length minimization problem. In this section we will briefly describe the process of proving that a problem is *NP*-complete, and then illustrate the technique on the 3-machine flowshop length problem.

The first point to note is that, technically, the only problems which can be *NP*-complete are problems for which the answer is either yes or no. Thus an optimization problem must be rephrased as a feasibility problem in order to put it into the desired form. For instance, the problem under consideration in this section becomes:

3-MACHINE FLOWSHOP LENGTH. Given a collection $\mathbf{C}$ of chains for a 3-machine flowshop, with task lengths given by $\tau : \{C[i] \mid C \in \mathbf{C},\ 1 \leqslant i \leqslant 3\} \rightarrow Z^+$, and a deadline D, does there exist a schedule S for $\mathbf{C}$ whose length does not exceed D? ∎

A proof of *NP*-completeness then consists of two parts. The first part, showing that the problem can be solved in polynomial time by a nondeterministic Turing Machine, is rather technical, but fortunately trivial in most cases. The interested reader is referred to [1], [24] for details. In the current paper we shall omit this part of the proof.

The main part of a proof of *NP*-completeness is the "reduction." We must show that a known *NP*-complete problem X can be *reduced*, or *transformed*, into our problem Y, in the following sense: given a specific input x for problem X, we must show how to construct a corresponding input y to problem Y, such that the answer for y is "yes" if and only if the answer to x is "yes." Moreover, the input length of y, as well as the time taken to construct y, must each be bounded by a polynomial function of $m(x)$, where m is the input length measure for problem X. See [1], [24] for a more detailed explanation of this proof method.

In all the proofs of this paper, the "known *NP*-complete problem" comes from the following number theoretic problem:

PARTITION INTO TRIPLES. Given positive integers n, B, and a set[1] of integers $\mathbf{A} = \{a_1, a_2, \ldots, a_{3n}\}$ with $0 < a_i < B$ for $1 \leqslant i \leqslant 3n$, does there exist a partition $\langle A_1, A_2, \ldots, A_n \rangle$ of $\mathbf{A}$ into 3-element sets such that for each i, $1 \leqslant i \leqslant n$, $\sum_{a \in A_i} a = B$? ∎

A careful examination of the proof of Theorem 3.5 in [6] shows that this problem is *NP*-complete, even if input length is measured by $\sum_{i=1}^{3n} a_i$. We shall add a few additional restrictions to the problem which will not affect its *NP*-completeness. First, we may assume that $\sum_{i=1}^{3n} a_i = nB$, for otherwise no such partition can exist. Second, we may assume that each a_i satisfies $B/4 < a_i < B/2$ and hence any set totaling B must contain exactly 3 elements. (Any problem of the original form can be simply transformed to obey this restriction by setting $a_i' = a_i + B$, $1 \leqslant i \leqslant 3n$, and setting $B' = 4B$.) Thus the *NP*-complete problem which we shall use is:

3-PARTITION. Given positive integers n, B, and a set of integers $\mathbf{A} = \{a_1, a_2, \ldots, a_{3n}\}$ with $\sum_{i=1}^{3n} a_i = nB$ and $B/4 < a_i < B/2$ for $1 \leqslant i \leqslant 3n$, does there exist a partition $\langle A_1, A_2, \ldots, A_n \rangle$ of $\mathbf{A}$ into 3-element sets such that, for each i, $\sum_{a \in A_i} a = B$? ∎

THEOREM 1. *The 3-machine flowshop length problem is NP-complete, even if input length is measured by the sum of the task lengths.*

PROOF. We show that 3-partition reduces to the given flowshop problem. Suppose we are given n, B, and $\mathbf{A} = \{a_1, a_2, \ldots, a_{3n}\}$ as specified for the 3-partition problem. The corresponding input to the 3-machine flowshop length problem will be a collection $\mathbf{C} = \{C_i \mid 0 \leqslant i \leqslant n\} \cup \{E_j \mid 1 \leqslant j \leqslant 3n\}$ of chains, with task lengths specified as

[1] $\mathbf{A}$ is actually a *multiset* since we do not require that all the numbers be distinct. The convention of treating two copies of the same integer as being distinct elements will be used throughout the paper and should cause no confusion.

follows:

$$\tau(C_0) = \langle 0, B, 2B \rangle,$$

$$\tau(C_i) = \langle 2B, B, 2B \rangle, \qquad 1 \leqslant i \leqslant n-1,$$

$$\tau(C_n) = \langle 2B, B, 0 \rangle,$$

$$\tau(E_j) = \langle 0, a_j, 0 \rangle, \qquad 1 \leqslant j \leqslant 3n.$$

The deadline D will be $(2n+1)B$.

Note that $m_2(\mathbf{C})$, the sum of the task lengths, is simply $(6n+1)B$, which is clearly bounded by a polynomial in $\sum_{i=1}^{3n} a_i = nB$, as is the time necessary to construct a description of this input. All that remains is to show that the desired partition of $\mathbf{A}$ exists if and only if there is a schedule for $\mathbf{C}$ of length less than or equal to D.

First, suppose a partition $\langle A_1, A_2, \ldots, A_n \rangle$ exists which has the desired form. That is, each set A_i consists of three elements $a_{g(i,1)}$, $a_{g(i,2)}$, and $a_{g(i,3)}$ such that for all i, $1 \leqslant i \leqslant n$, $\sum_{j=1}^{3} a_{g(i,j)} = B$. Then the following schedule S has length $D = (2n+1)B$.

First we describe how the C_i chains are scheduled.

$$S(C_0) = \langle 0, 0, B \rangle.$$

$$S(C_i) = \langle 2B(i-1), 2Bi, 2Bi + B \rangle, \qquad 1 \leqslant i \leqslant n.$$

See Figure 2. Note that this basic framework leaves a series of n "time slots" open on machine P_2, each of length exactly B. These are precisely tailored so that we can fit in the E_j chains as follows. For each i, $1 \leqslant i \leqslant n$, set

$$S(E_{g(i,1)}) = \langle 0, 2Bi - B, 2Bn + B \rangle,$$

$$S(E_{g(i,2)}) = \langle 0, 2Bi - B + a_{g(i,1)}, 2Bn + B \rangle,$$

$$S(E_{g(i,3)}) = \langle 0, 2Bi - B + a_{g(i,1)} + a_{g(i,2)}, 2Bn + B \rangle.$$

Since $\sum_{j=1}^{3} a_{g(i,j)} = B$, $1 \leqslant i \leqslant n$, this yields a valid schedule with length$(S) = D$.

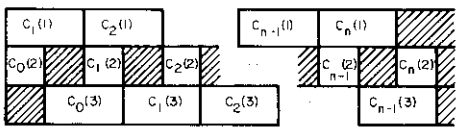


FIGURE 2. Proof of Theorem 1. The C chains leave "slots" of length B on processor P_2. The "slots" can be filled exactly with $E[2]$ tasks when the 3-partition problem has a solution.

Conversely, suppose a schedule S with length$(S) \leqslant D$ does exist. It is easy to see that we must have length$(S) = D = (2n+1)B$, and that the C_i chains must be scheduled the same way as they are in Figure 2, except that C_1 through C_{n-1}, being identical, may be permuted. Thus there are again n slots of length B into which the machine P_2 tasks of the E_j chains must be placed.

Since the total length of the $E_j[2]$ tasks is $\sum_{i=1}^{3n} a_i = nB$, every one of these n slots must be filled completely, and hence must contain a set of tasks whose total length is exactly B. Now since every $a_i > B/4$, no such set can contain more than three tasks. Similarly, since every $a_i < B/2$, no such set can contain less than three tasks. Thus each set contains *exactly* three tasks $E_j[2]$. Hence, by setting $A_i = \{a_j \mid 2B(i-1) < S(E_j[2]) < 2Bi\}$, $1 \leqslant i \leqslant n$, we obtain our desired partition. ∎

3. The mean-flow-time minimization problem. In this section we prove that the 2-machine flowshop mean-flow-time problem is *NP*-complete. As in the previous section we must first cast the problem in the form of a feasibility question:

2-MACHINE FLOWSHOP FLOW PROBLEM. Given a collection $\mathbf{C}$ of chains for a 2-machine flowshop, with task lengths given by $\tau : \{C[i] \mid C \in \mathbf{C}, 1 \leqslant i \leqslant 2\} \to Z^+$, and a bound D, does there exist a schedule S for $\mathbf{C}$ whose mean flow time $MFT(S)$ does not exceed D? ∎

As in the last section we will reduce 3-partition to the flowshop problem under consideration, but this time the reduction and its proof are much more complicated. We will start as in Figure 3(a) by defining a set of chains $T_0, T_1, T_2, \ldots, T_n$, where $\tau(T_0) = \langle 0, 1 \rangle$ and for all i, $1 \leqslant i \leqslant n$, $\tau(T_i) = \langle t, 1 \rangle$. If these are the only chains to be scheduled, then on the second processor we will have n slots of length $t - 1$. These slots will be used to test if the 3-partition problem has a solution. We will add chains to the system which do not require any processing on P_1, and exactly fill the slots iff the 3-partition problem has a solution.

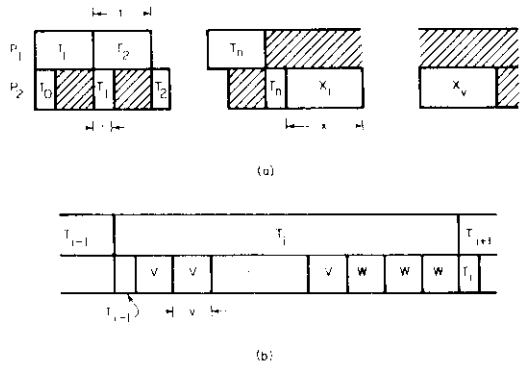


FIGURE 3. The T-chains in (a) (subscripts [i] are omitted for clarity) create "slots" on the second processor. The X-tasks ensure that no idle time occurs before X_1 starts. The W tasks in (b) do the actual checking for the 3-partition solution. There are enough V tasks to ensure that if $T_{i-1}[2]$ is delayed, then the mean flow time becomes too high.

But if the slots are to work we must ensure that for all i, $1 \leqslant i \leqslant n$, $T_i[2]$ is executed as soon as it is ready. We will add a large number of long tasks $X_1, X_2, \ldots, X_v$ of length x, which because of their length will be the last tasks to be executed on processor P_2. The number of these tasks, v, will be large enough that if $T_n[2]$ is delayed for even one unit, the mean flow time of the system exceeds the bound D. As a consequence, no schedule meeting the mean-flow-time bound will have any idle time on processor P_2.

To ensure that the $T_i[2]$, $0 \leqslant i \leqslant n - 1$, are executed as soon as they are ready, we will have a number of tasks of intermediate length, the V's and the W's. Their quantity and size will be such that exactly u of them must be executed between $T_i[2]$ and $T_{i+1}[2]$, and such that if these u tasks are delayed as much as one time unit, the mean flow time will again exceed D.

Finally, for u of these tasks to fit exactly between $T_i[2]$ and $T_{i+1}[2]$, $u - 3$ of them will have to be V-type tasks (with length v) and the remaining three must be W-type tasks (with length $v + a_i$ for some a_i) whose sum is exactly $3v + B$.

From the above intuitive description it is probably evident that there will be a delicate balance between the numbers of tasks and their lengths in the reduction. In

the proof of the theorem we will demonstrate that appropriate numbers u, v, x and t can be chosen to make the arguments work.

THEOREM 2. *The 2-machine flowshop flow problem is NP-complete, even if input length is measured by the sum of the task lengths.*

PROOF. Suppose we are given the 3-partition problem n, B, and $\mathbf{A} = \{a_1, a_2, \ldots, a_{3n}\}$. We will first describe the details of the corresponding flowshop problem $\mathbf{C}$.

Let

$$u = 3nB + 1,$$

$$v = u + 3nB + nu + n(n-1)u(B+1)/2.$$

$$t = uv + B + 1, \quad \text{and}$$

$$x = 2(n+2)t + v.$$

There are basically three classes of chains in $\mathbf{C}$, one of which is broken into two subclasses. The first class, $\mathbf{T} = \{T_i \mid 0 \leqslant i \leqslant n\}$ consists of "T-type" chains, where task lengths are specified as follows:

$$\tau(T_0) = \langle 0, 1 \rangle,$$

$$\tau(T_i) = \langle t, 1 \rangle, \qquad 1 \leqslant i \leqslant n.$$

The second class, $\mathbf{X} = \{X_i \mid 1 \leqslant i \leqslant v\}$, consists of "$X$-type" chains, with task lengths specified as follows:

$$\tau(X_i) = \langle 0, x \rangle, \qquad 1 \leqslant i \leqslant v.$$

The final class $\mathbf{U} = \mathbf{V} \cup \mathbf{W}$, is made up of two subclasses $\mathbf{V} = \{V_{i,j} \mid 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant u - 3\}$ and $\mathbf{W} = \{W_i \mid 1 \leqslant i \leqslant 3n\}$, consisting of "$V$-type" and "$W$-type" chains respectively; called "U-type" chains collectively. The corresponding task lengths are as follows:

$$\tau(V_{i,j}) = \langle 0, v \rangle, \qquad 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant u - 3, \quad \text{and}$$

$$\tau(W_i) = \langle 0, v + a_i \rangle, \qquad 1 \leqslant i \leqslant 3n.$$

The bound is given by $D = T_\epsilon + X_\epsilon + U_\epsilon$, where

$$T_\epsilon = \sum_{i=0}^{n} (it + 1) = n + 1 + n(n+1)t/2,$$

$$X_\epsilon = \sum_{i=1}^{v} (nt + 1 + ix) = v(nt + 1) + v(v+1)x/2, \quad \text{and}$$

$$U_\epsilon = 3nB + \sum_{i=0}^{n-1} \left[\sum_{j=1}^{u} (jv + it + 1) \right]$$

$$= 3nB + nu + n(n-1)u(B+1)/2 + nu(nu+1)v/2, \quad \text{by the definition of } t.$$

If we let $s = 3nB + nu + n(n-1)u(B+1)/2$, then $U_\epsilon = s + nu(nu+1)v/2$. In terms of s, $v = s + u$.

Note that the sum of the task lengths, though large, is still bounded by a polynomial in $\sum_{i=1}^{3n} a_i = nB$, as is the time necessary to construct a description of this input. Thus all that is left to show is that the desired partition of **A** exists if and only if there is a schedule S for **C** with $\mathrm{MFT}(S) \leqslant D$.

First suppose there is a partition $\langle A_1, A_2, \ldots, A_n \rangle$ of the desired form, where as before $A_i = \{a_{g(i,1)}, a_{g(i,2)}, a_{g(i,3)}\}$ and $\sum_{j=1}^{3} a_{g(i,j)} = B$, $1 \leqslant i \leqslant n$. Consider the schedule defined as follows:

$$S(T_0) = \langle 0, 0 \rangle,$$

$$S(T_i) = \langle (i-1)t, it \rangle, \qquad 1 \leqslant i \leqslant n,$$

$$S(X_i) = \langle 0, nt + 1 + (i-1)x \rangle, \qquad 1 \leqslant i \leqslant v,$$

$$S(V_{i,j}) = \langle 0, (i-1)t + 1 + (j-1)v \rangle, \qquad 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant u-3,$$

$$S(W_{g(i,1)}) = \langle 0, (i-1)t + 1 + (u-3)v \rangle, \qquad 1 \leqslant i \leqslant n,$$

$$S(W_{g(i,2)}) = \langle 0, (i-1)t + 1 + (u-2)v + a_{g(i,1)} \rangle, \qquad 1 \leqslant i \leqslant n,$$

$$S(W_{g(i,3)}) = \langle 0, (i-1)t + 1 + (u-1)v + a_{g(i,1)} + a_{g(i,2)} \rangle, \qquad 1 \leqslant i \leqslant n.$$

See Figure 3. The central observation in verifying that the above is a valid schedule is that u of the U-type chains have their P_2 task scheduled between the finish of $T_{i-1}[2]$ and the start of $T_i[2]$, $1 \leqslant i \leqslant n$, and that the sum of these u task lengths is exactly $uv + \sum_{j=1}^{3} a_{g(i,j)} = uv + B = t - 1$. Hence the tasks will fit exactly. The remaining details of the verification that S is a valid schedule for **C** are left to the reader.

Calculation of the mean flow time $\mathrm{MFT}(S)$ is also straightforward. The sum of the finishing times for the T-type chains is clearly T_ϵ, and the sum of the finishing times for the X-type chains is also clearly X_ϵ. For the U-type tasks the sum is not actually U_ϵ but in fact a bit smaller

$$\sum_{U \in \mathbf{U}} f_S(U[2]) = \sum_{i=0}^{n-1} \left[\sum_{j=1}^{u} (jv + it + 1) + 3a_{g(i+1,1)} + 2a_{g(i+1,2)} + a_{g(i+1,3)} \right]$$

$$< U_\epsilon = \sum_{i=0}^{n-1} \left[\sum_{j=1}^{u} (jv + it + 1) \right] + 3 \sum_{i=1}^{3n} a_i.$$

Thus the constructed schedule S has $\mathrm{MFT}(S) < T_\epsilon + X_\epsilon + U_\epsilon = D$, as desired.

Now suppose there *is* a schedule S with $\mathrm{MFT}(S) \leqslant D$ (a "good" schedule). We complete our proof of Theorem 2 by showing that the desired partition of **A** must then exist. The basic idea of our argument is to show that if there is a good schedule, then there must be a good schedule with a form like that indicated in Figure 3(a).

First note that we can make the following restrictions on a good schedule, without loss of generality.

Claim T1. Every task on machine P_1 which has length 0 starts at time 0.

Claim T2. Since the chains T_i, $1 \leqslant i \leqslant n$, all have the same lengths, $S(T_i[1]) \leqslant S(T_{i+1}[1])$, $1 \leqslant i < n$. Since $T_0[1]$ is a 0 length task it must start at time 0 from Claim T1. Moreover, $T_{i+1}[1]$ can start as soon as $T_i[1]$ finishes, since executing a task earlier on P_1 cannot violate the chain constraint. Thus $S(T_i[1]) = (i-1)t$, $0 \leqslant i \leqslant n$.

Claim T3. Having established that $S(T_i[1]) = (i-1)t$, we can now say that $S(T_i[2]) < S(T_{i+1}[2])$ for $0 \leqslant i \leqslant n$.[2] The reason is that the lengths of all $T[2]$ tasks are the same; so if $S(T_j[2]) > S(T_{j+1}[2])$, for some j, then we can interchange $T_j[2]$ and $T_{j+1}[2]$. The chain constraint is not violated since $T_j[1]$ is executed before $T_{j+1}[1]$; so if $T_{j+1}[2]$ is ready, then $T_j[2]$ must also be ready.

Claim X1. Since all the X-type chains are identical and have their machine P_1 tasks starting at time 0, we can order them so that $S(X_i[2]) < S(X_{i+1}[2])$, $1 \leqslant i \leqslant c-1$.

At this point we can begin the main arguments of the proof. These arguments are phrased in a series of claims about the X-type, and then the U-type tasks.

Claim X2. There exists a good schedule S satisfying the previous claims and such that no task $X_i[2]$ starts before $nt+1$, i.e., $S(X_i[2]) \geqslant nt+1$, $1 \leqslant i \leqslant c$.

Let S be a minimum mean flow time schedule satisfying the previous claims. Then S is the desired schedule. For if not, let $X_i[2]$ start before $nt+1$ for some i. Since the length of each $X[2]$ task is greater than $nt+1$, from Claim X1 this task can only be $X_1[2]$.

Since $\tau(X_1[2]) = 2(n+2)t + c$, by the time $X_1[2]$ finishes, all tasks to be executed on P_1 have finished from Claim T2. Thus, tasks executed after $X_1[2]$ can be reordered without violating the chain constraint; in order to minimize mean flow time they must be executed shortest first in S. We will bound the mean flow time of S and show that S cannot be a good schedule.

Let us start with X_f a lower bound on the sum of the finishing times of the X-type chains. From the previous paragraph all $X[2]$ tasks except $X_1[2]$ are the last to be executed on P_2. $X_1[2]$ can finish no earlier than x. Thus

$$X_f = x + \sum_{j=1}^{c-1} (nt + 1 + x + jx) = X_\epsilon - nt - 1.$$

Since $X_1[2]$ starts before $nt+1$, $T_n[2]$ cannot start until $X_1[2]$ has finished. Thus T_f, a lower bound on the sum of the finishing times of the T-type chains, is given by

$$T_f = \sum_{j=0}^{n-1} (jt + 1) + x \geqslant T_\epsilon + (n+2)t + c.$$

A trivial lower bound U_f on the sum of the finishing times of the U-type chains can be determined by treating them as tasks of length c scheduled one after the other. This bound is given by

$$U_f = \sum_{j=1}^{nt} jc = nt(nt+1)c/2.$$

From the definition of U_ϵ, $U_f = U_\epsilon - s$. Thus

$$\mathrm{MFT}(S) \geqslant T_f + X_f + U_f = T_\epsilon + X_\epsilon + U_\epsilon + (n+2)t + c - nt - 1 - s.$$

Since $(n+2)t > nt+1$ and $c > s$, $\mathrm{MFT}(S) > D$, so S cannot be a good schedule. Thus Claim X2 must be true.

Claim X3. There exists a good schedule S satisfying the previous claims such that $S(X_1[2]) = nt+1$. In addition, P_2 is not idle before $nt+1$, and all $T[2]$- and $U[2]$-type tasks are finished before $nt+1$.

[2] In fact as shown in [5] in a 2-processor flowshop, without loss of generality, chains appear in the same order on both processors.

Again let S be a minimum mean-flow-time schedule satisfying the previous claims. Then S also satisfies this claim. From Claim X2, $S(X_1[2]) \geqslant nt + 1$. So suppose $S(X_1[2]) > nt + 1$. Then the total finishing times of the X-type chains, using Claim X1, must be at least

$$\sum_{i=1}^{t} (nt + 2 + ix) = X_t + r.$$

From Claim T3 the sum of the finishing times of the T-type chains is at least

$$\sum_{i=0}^{n} (it + 1) = T_t.$$

The trivial lower bound $U_t = U_t - s$ from Claim X2 still applies to the U-type chains. Adding these bounds and noting that $r > s$, we get $\mathrm{MFT}(S) > D$, so S cannot be a good schedule. The claim must therefore be true.

In order to see that P_2 may not be idle before $nt + 1$, observe that the sum of the lengths of all $T[2]$, $U[2]$, and $W[2]$ tasks is exactly $nt + 1$. Thus if P_2 *were* idle, one of these tasks would have to be executed after $X_1[2]$, which starts at time $nt + 1$. Since that delayed task would be shorter than $X_1[2]$, we could get a valid schedule with smaller mean flow time by interchanging the two, a contradiction.

Claim U1. There exists a good schedule S satisfying the previous claims such that $T_i[2]$ is preceded by exactly iu $U[2]$-type tasks, $0 \leqslant i \leqslant n$.

Again, if S is a minimum MFT schedule satisfying the previous claims, then it satisfies the current claim. Otherwise, first suppose that $T_i[2]$ is preceded by less than iu $U[2]$-type tasks. Then the total length of tasks preceding $T_i[2]$ on P_2 is no more than $i + (iu - 1)v + nB < it$. (Recall that $t = uv + B + 1$ and $v \geqslant 3nB$.) By the chain constraint and Claim T2, $S(T_i[2]) \geqslant it$, so P_2 must be idle sometime before it, and hence before $nt + 1$, in violation of Claim X3. On the other hand, suppose $T_i[2]$ is preceded by more than iu $U[2]$-type tasks. In this case the total length of the tasks preceding $T_i[2]$ and P_2 must be at least $i + (iu + 1)v = it + r - iB$, so $S(T_i[2]) \geqslant it + r - iB$. The sum of the finishing times of the T-type chains must then be at least

$$\sum_{j=0}^{n} (jt + 1) + r - iB = T_t + r - iB.$$

Since the sum of the finishing times of the X-type tasks must be at least X_t by the preceding claim, and the sum for the U-type chains at least $U_t = U_t - s$ by Claim X2, we get

$$\mathrm{MFT}(S) \geqslant X_t + T_t + U_t + r - iB - s.$$

Since $r = u + s$ and $u > iB$, $\mathrm{MFT}(S) > D$, contradicting the fact that S is a good schedule. Hence the claim is true.

Claim U2. There exists a good schedule S satisfying the previous claims and such that $S(T_i[2]) = it$, $0 \leqslant i \leqslant n$.

Again, let S be a minimum MFT schedule satisfying the previous claims. We show that S must also satisfy the current one. By Claim T2, $S(T_i[2]) \geqslant it$. Suppose for any i, $S(T_i[2]) > it$. By Claim X3, i must be such that $0 \leqslant i \leqslant n - 1$, since $S(X_1[2]) = nt + 1$. By Claim U1 there are u $U[2]$-type tasks starting between $T_i[2]$ and $T_{i+1}[2]$. The sum of the finishing time for these tasks must be at least $\sum_{j=1}^{u} (it + 2 + jv)$, whereas the sum of the finishing times of the $U[2]$-type tasks between $T_k[2]$ and $T_{k+1}[2]$ for any k, $0 \leqslant k < n$, must be at least $\sum_{j=1}^{u} (kt + 1 + jv)$. Thus the sum of the finishing times for all $U[2]$-type tasks is at least

$$\sum_{k=0}^{n-1} \sum_{j=1}^{u} [kt + 1 + jv] + u = U_t - 3nB + u = U_t + 1.$$

Therefore, since the sums of the finishing times of the X-type and T-type chains are at least X_ϵ and T_ϵ respectively, we have

$$\mathrm{MFT}(S) \geqslant X_\epsilon + T_\epsilon + U_\epsilon + 1 = D + 1,$$

a contradiction of the fact that S is a good schedule. Hence S does satisfy the claim.

We are now ready to complete the proof of Theorem 2. Let S be a good schedule satisfying all the previous claims. Then our desired partition of $\mathbf{A}$ is given by the following sets A_i, $1 \leqslant i \leqslant n$,

$$A_i = \{ a_k \mid S(T_{i-1}[2]) < S(W_k[2]) < S(T_i[2]) \}.$$

By the preceding claims we know that

$$\mathbf{U}_i = \{ U \in \mathbf{U} \mid S(T_{i-1}[2]) < S(U[2]) < S(T_i[2]) \}$$

has cardinality u and $\sum_{U \in \mathbf{U}_i} \tau(U[2]) = t - 1 = ut + B$.

Since every $V[2]$-type task in $\mathbf{U}_i$ has length v, and every $W[2]$-type task in $\mathbf{U}_i$ has length between $v + B/4$ and $v + B/2$, this means that $\mathbf{U}_i$ must contain exactly three $W[2]$-type tasks, and that $\sum_{W_k \in \mathbf{U}_i} \tau(W_k[2]) = 3v + B$, and hence $\sum_{W_k \in \mathbf{U}_i} a_k = B$. Thus the A_i are disjoint 3-element sets each of which has sum exactly B and $\langle A_1, A_2, \ldots, A_n \rangle$ is our desired partition.

Hence the collection $\mathbf{C}$ of chains has a schedule with mean flow time less than or equal to the deadline D if and only if $\mathbf{A}$ has the desired partition, and so the proof of NP-completeness is complete. ∎

4. Some related problems. The results in the preceding sections show how few machines are sufficient for each of the problems to be NP-complete. The 3-machine flowshop length problem is NP-complete, whereas the corresponding 2-machine problem can be solved in time proportional to $n \cdot \log n$ for n chains. Similarly, the 2-machine flowshop flow problem is NP-complete, whereas the corresponding 1-machine problem can be solved simply by scheduling tasks in order of nondecreasing length [22]. In both cases, the problem remains NP-complete when the number of machines is increased.

A number of generalizations of the basic flowshop problem also give rise to NP-complete problems. In [2], [21] a generalization is considered where, instead of just one machine, P_i, being available for processing the ith task of each chain, there are m_i such machines which can be used interchangeably for those tasks. Our results carry over directly to this generalized model. The problem of schedule-length minimization is NP-complete if there are three types of machines and the mean-flow-time minimization problem is NP-complete for two types of machines.

Notice that, if there is only one machine type, this generalized flowshop model becomes the well-studied model for scheduling independent tasks on identical processors. Here, in contrast to our results for standard flowshop problems, devising minimum mean-flow-time schedules is easier than finding minimum length schedules. In particular, if input length is measured by the number of bits required to describe the task lengths, the problem of finding a minimum length schedule is NP-complete even for two machines [11]. On the other hand, for any $m \geqslant 1$, a minimum flow time schedule can be found in polynomial time, even if the length of a task depends on which machine processes it [3].

Another much-studied generalization of the basic flowshop model is the *jobshop* [5]. In this model each chain $C = \langle C[1], C[2], \ldots, C[k] \rangle$ is made up of tasks $C[i]$ having

length $\tau(C[i])$ and machine assignment $m(C[i])$, which specifies the machine that must process $C[i]$. The chains need not all contain the same number of tasks, but the constraint that $C[i]$ must be completed before starting $C[i+1]$ is retained. Since the basic flowshop problems are special cases of the corresponding jobshop problems, our results carry over. The 3-machine jobshop length problem and the 2-machine jobshop flow problem are *NP*-complete. However, S. M. Johnson's algorithm [10] does not extend to solving the 2-machine jobshop length problem, and here we have the following stronger result:

THEOREM 3. *The 2-machine jobshop length problem is NP-complete, even if input length is measured by the sum of the task lengths.*

PROOF. The reduction is once again from 3-partition. Suppose we are given n, B, and $\mathbf{A} = \{a_1, a_2, \ldots, a_{3n}\}$. The corresponding jobshop input is given as follows. The collection $\mathbf{C}$ of chains is made up of C_0, which has $2n$ tasks, and C_j, $1 \leq j \leq 3n$, each with two tasks. The machine assignments and lengths are given as follows.

$$m(C_0[i]) = \begin{cases} P_1 : 2 \leq i \leq 2n \quad \text{and } i \text{ even}, \\ P_2 : 1 \leq i \leq 2n-1 \quad \text{and } i \text{ odd}. \end{cases}$$

$$m(C_j[1]) = P_1, \qquad 1 \leq j \leq 3n,$$

$$m(C_j[2]) = P_2, \qquad 1 \leq j \leq 3n,$$

$$\tau(C_0[i]) = B, \qquad 1 \leq i \leq 2n,$$

$$\tau(C_j[1]) = 0, \qquad 1 \leq j \leq 3n,$$

$$\tau(C_j[2]) = a_j, \qquad 1 \leq j \leq 3n.$$

The deadline is given by $D = 2nB$. We leave to the reader the straightforward task of verifying that $\mathbf{C}$ can be scheduled to meet the deadline if and only if $\mathbf{A}$ has the desired partition. ∎

As a final example of a flowshop-like problem we mention the special type of flowshop in which there can be no waiting time between the finish of one task in a chain and the start of the next. This is sometimes referred to as the "hot ingot" problem (the ingot cannot be allowed to cool off in the middle of processing) and has been studied in [15], [16]. Although S. M. Johnson's algorithm [10] does not work for the 2-machine length minimization problem for this type of a flowshop, the problem can be solved in polynomial time using a special-case traveling salesman algorithm of Gilmore and Gomory [8], as pointed out in [16]. It would be interesting to determine whether analogues of our *NP*-completeness results hold for this special problem.

We conclude with a comment about what *NP*-completeness results do and do not mean. They do not mean that particular simple instances of the problem cannot be solved in practice. It is quite possible that an algorithm can be developed for certain restricted cases which will rapidly find optimal schedules. However, *NP*-completeness does mean that any algorithm purporting to solve such a problem in its full generality is likely to require exponential time in the worst case and thus be impractical except for relatively small inputs. For larger, more complicated inputs, the best approach may well be to seek heuristics which will guarantee near-optimal results, as in [13].

References

[1] Aho, A. V., Hopcroft, J. E. and Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms*. Addison Wesley, Reading, Mass.

[2] Arthanari, T. S. (1974). On Some Problems of Sequencing and Grouping. Ph.D. Thesis, Indian Statistical Institute, Calcutta.
[3] Bruno, J. L., Coffman, E. G., Jr. and Sethi, R. (1974). Scheduling Independent Tasks to Reduce Mean Finishing Time. *CACM* **17** 382–387.
[4] Coffman, E. G., Jr. (ed.) (1976). *Computer and Jobshop Scheduling Theory*. John Wiley, New York, N. Y.
[5] Conway, R. W., Maxwell, W. L. and Miller, L. W. (1967). *Theory of Scheduling*. Addison Wesley, Reading, Mass.
[6] Garey, M. R. and Johnson, D. S. (1975). Complexity Results for Multiprocessor Scheduling under Resource Constraints. *SIAM J. Computing* **4** 397–411.
[7] ———, Johnson, D. S. and Stockmeyer, L. (1974). Some Simplified NP-Complete Problems. *Sixth Annual ACM Symposium on Theory of Computing* 47–63.
[8] Gilmore, P. C. and Gomory, R. E. (1964). Sequencing a One State-Variable Machine: A Solvable Case of the Traveling Salesman Problem. *Operations Res.* **12** 655–679.
[9] Horowitz, E. and Sahni, S. (1976). Exact and Approximate Algorithms for Scheduling Nonidentical Processors. *J. Assoc. Comput. Mach.* **23** 317–327.
[10] Johnson, S. M. (1954). Optimal Two- and Three-Stage Production Schedules with Setup Times Included. *Nav. Res. Logist. Quart.* **1** 61–68.
[11] Karp, R. M. (1972). Reducibility among Combinatorial Problems. In R. E. Miller and J. W. Thatcher (eds.) *Complexity of Computer Computations*. Plenum Press, New York 85–104.
[12] ———, (1975). On the Computational Complexity of Combinatorial Problems. *Networks* **5** 45–68.
[13] Kohler, W. H. and Steiglitz, K. (1975). Exact, Approximate and Guaranteed Accuracy Algorithms for the Flowshop Problem $n/2/F/\bar{F}$. *J. Assoc. Comput. Mach.* **22** 106–114.
[14] Lenstra, J. K., Rinnooy Kan, A. H. G. and Brucker, P. (to appear). Complexity of Machine Scheduling Problems. *Ann. Discrete Math.*
[15] Piehler, J. (1960). Ein Beitrag zum Reihenfolgeproblem. *Unternehmenforschung* **4** 138–142.
[16] Reddi, S. S. and Ramamoorthy, C. V. (1972). On the Flowshop Sequencing Problem with No Wait in Process. *Operational Research Quarterly* **23** 323–331.
[17] Rinnooy Kan, A. H. G. (1976). *Machine Scheduling Problems*. H. E. Stenfert Kroese B. V., Leiden, Netherlands.
[18] Sahni, S. (1974). Computationally Related Problems. *SIAM J. Computing* **3** 262–279.
[19] ——, (1976). Algorithms for Scheduling Independent Tasks. *J. Assoc. Comput. Mach.* **23** 116–127.
[20] Sethi, R. (1975). Complete Register Allocation Problems. *SIAM J. Computing* **4** 226–248.
[21] Shen, V. Y. and Chen, Y. E. (1972). A Scheduling Strategy for the Flowshop Problem in a System with Two Classes of Processors. *Sixth Annual Conference on Information Sciences and Systems*. Princeton, N. J.
[22] Smith, W. E. (1956). Various Optimizers for Single Stage Production. *Nav. Res. Logist. Quart.* **3** 59–66.
[23] Ullman, J. D. (1975). NP-Complete Scheduling Problems. *J. Comput. System Sci.* **10** 384–393.
[24] ———, (1976). Complexity of Sequencing Problems. In E. G. Coffman, Jr. (ed.) *Computer and Job-Shop Scheduling Theory*. John Wiley, New York.

BELL LABORATORIES, MURRAY HILL, NEW JERSEY 07974
COMPUTER SCIENCE DEPARTMENT, THE PENNSYLVANIA STATE UNIVERSITY, UNIVERSITY PARK, PENNSYLVANIA 16802