

**stichting
mathematisch
centrum**



AFDELING MATHEMATISCHE BESLISKUNDE
(DEPARTMENT OF OPERATIONS RESEARCH)

BW 131/80

DECEMBER

K. NAKAJIMA, S.L. HAKIMI, J.K. LENSTRA

COMPLEXITY RESULTS FOR SCHEDULING TASKS IN FIXED INTERVALS
ON TWO TYPES OF MACHINES

Preprint

kruislaan 413 1098 SJ amsterdam

BIBLIOTHEEK MATHEMATISCH CENTRUM
AMSTERDAM

Printed at the Mathematical Centre, 413 Kruislaan, Amsterdam.

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

COMPLEXITY RESULTS FOR SCHEDULING TASKS IN FIXED INTERVALS
ON TWO TYPES OF MACHINES

K. NAKAJIMA

*Computer Science Division, Department of Electrical Engineering, Texas Tech
University, Lubbock, TX 79409, U.S.A.*

S.L. HAKIMI

*Department of Electrical Engineering and Computer Science, Northwestern
University, Evanston, IL 60201, U.S.A.*

J.K. LENSTRA

Mathematisch Centrum, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Abstract

Suppose that n independent tasks are to be scheduled without preemption on an unlimited number of parallel machines of two types: inexpensive slow machines and expensive fast machines. Each task requires a given processing time on a slow machine or a given smaller processing time on a fast machine. We make two different feasibility assumptions: (a) each task has a specified processing interval, the length of which is equal to the processing time on a slow machine; (b) each task has a specified starting time. For either problem type, we wish to find a feasible schedule of minimum total machine costs. It is shown that both problems are NP-hard in the strong sense. These results are complemented by polynomial algorithms for some special cases.

Key words & phrases: *parallel machines, tasks, release dates, deadlines, computational complexity, NP-hardness, polynomial algorithm.*

Note: This report has been submitted for publication in a journal.

1. Introduction

We begin by considering the following problem. Suppose there are n tasks T_1, \dots, T_n and an unlimited number of *identical parallel machines*. Each task T_j requires a given *processing time* p_j and is to be executed without interruption between a given *release date* r_j and a given *deadline* $d_j = r_j + p_j$. The tasks are *independent* in the sense that there are no precedence constraints between them. Each machine can execute any task, but no more than one at a time. The problem is to find the minimum number of machines needed to execute all tasks as well as a corresponding schedule of the tasks on the machines.

This problem is known as the "fixed job schedule problem" [6] and as the "channel assignment problem" [8;9;10]. It has applications in such diverse areas as vehicle scheduling [2;15], machine scheduling [6;8], and computer wiring [8;9;10]. As a special case of Dilworth's chain decomposition problem, it is solvable in $O(n^2)$ time by the staircase rule of Ford and Fulkerson [3,p.65] and by the step-function method of Gertsbakh and Stern [6]. Hashimoto and Stevens [9;10] presented some interesting graph theoretical approaches to the problem and proposed an $O(n^2)$ algorithm, for which Kernighan, Schweikert and Persky [12] gave an $O(n \log n)$ implementation. Recently, Gupta, Lee and Leung [8] independently developed a different $O(n \log n)$ algorithm and also showed that any solution method for the problem requires $\Omega(n \log n)$ time.

In this paper we will consider a natural generalization of this problem which has potential applications in the scheduling areas mentioned above. Again, there are n independent tasks T_1, \dots, T_n , but there are two types of machines: *slow machines* of cost C^S and *fast machines* of cost $C^F > C^S$. Each task T_j requires a processing time p_j on a slow machine or $q_j < p_j$ on a fast machine and is to be executed without interruption between its release date

r_j and its deadline $d_j = r_j + p_j$. It is assumed that all numerical problem data are integers. In a feasible schedule, the tasks assigned to slow machines have to start at their release dates in order to meet their deadlines. For the tasks T_j assigned to fast machines, we make two different feasibility assumptions:

- (a) VST (*variable starting times*): T_j may start at any time in the interval $[r_j, d_j - q_j]$;
- (b) FST (*fixed starting times*): T_j has to start at time r_j .

A schedule using m^s slow machines and m^f fast machines has total costs $m^s C^s + m^f C^f$. For either problem type, we wish to find a feasible schedule of minimum total costs.

In Section 2 we show that the VST problem is NP-hard [1;4;5;11], even if all release dates are equal. In Section 3 we extend our techniques to prove that the FST problem is NP-hard in the case of arbitrary release dates; the case of equal release dates is trivially solvable in $O(n)$ time. The NP-hardness results are "strong" [4;5] in the sense that they hold even with respect to a unary encoding of the data; this implies that there exists no pseudopolynomial algorithm for these problems unless $\mathcal{P} = \mathcal{NP}$.

In Sections 4 and 5 we consider the special case that $q_j = 1$ ($j = 1, \dots, n$). We present $O(n \log n)$ algorithms for the VST problem with equal release dates and for the FST problem with arbitrary release dates, respectively.

p_j arbitrary	VST		FST	
	r_j arbitrary	r_j equal	r_j arbitrary	r_j equal
q_j arbitrary	NP-hard (§2)	NP-hard (§2)	NP-hard (§3)	$O(n)$ (§3)
$q_j = 1$	open	$O(n \log n)$ (§4)	$O(n \log n)$ (§5)	$O(n)$ (§3)

TABLE 1. Summary of complexity results.

These results represent an almost complete complexity classification of the problem class under consideration, as demonstrated by Table 1.

2. NP-hardness of the VST problem

THEOREM 1. *The VST problem is NP-hard in the strong sense, even if all release dates are equal.*

Our proof holds for the case that $C^f/C^s = 3$ and $p_j/q_j = 3$ ($j = 1, \dots, n$).

Theorem 1 dominates a previous result, stating that the VST problem is NP-hard in the strong sense if the release dates are arbitrary, C^f/C^s is an arbitrary constant between 1 and 7, and $p_j/q_j = 4$ ($j = 1, \dots, n$) [17].

Proof of Theorem 1. We have to show that a problem which is known to be NP-complete in the strong sense is reducible to the VST problem. Our starting point will be the following problem [5,p.224,[SP15]]:

3-PARTITION: Given a set $S = \{1, \dots, 3t\}$ and positive integers a_1, \dots, a_{3t}, b with $\frac{1}{4}b < a_j < \frac{1}{2}b$ ($j \in S$) and $\sum_{j \in S} a_j = tb$, does there exist a partition of S into t disjoint 3-element subsets S_1, \dots, S_t such that $\sum_{j \in S_i} a_j = b$ ($i = 1, \dots, t$)?

Given any instance of 3-PARTITION, we construct a corresponding instance of the VST problem with equal release dates as follows:

- the cost coefficients are defined by $C^s = 1$, $C^f = 3$;
- there are $4t$ tasks:

a-tasks T_j^a ($j \in S$) with $r_j^a = 0$, $p_j^a = 6a_j$, $q_j^a = 2a_j$;
 b-tasks T_i^b ($i \in \{1, \dots, t\}$) with $r_i^b = 0$, $p_i^b = 3b$, $q_i^b = b$.

We claim that 3-PARTITION has a solution if and only if there exists a feasible schedule with total costs at most $C^* = 3t$.

Suppose that 3-PARTITION has a solution $\{S_1, \dots, S_t\}$. It is possible to construct a feasible schedule for all tasks on t fast machines M_1^f, \dots, M_t^f as follows (cf. Figure 1): for each $i \in \{1, \dots, t\}$, machine M_i^f processes the three tasks T_j^a ($j \in S_i$) in nondecreasing order of q_j^a value in the interval $[0, 2b]$, and the task T_i^b in $[2b, 3b]$; note that the starting time of each task falls within the required interval. The total costs of this schedule are equal to $tC^f = C^*$.

Conversely, suppose that there exists a feasible schedule with total costs at most C^* . Let there be:

- m_0^s slow machines processing one a-task and no b-tasks,
- m_1^s slow machines processing no a-tasks and one b-task,
- m_0^f fast machines processing four a-tasks and no b-tasks,
- m_1^f fast machines processing three a-tasks and one b-task.

It is easily argued that no other possibilities need be considered. We have:

Instance of 3-PARTITION:

$t = 3; b = 25;$	j	1	2	3	4	5	6	7	8	9
	a_j	7	7	7	8	8	8	9	10	11

Solution: $\{\{1,2,9\}, \{3,4,8\}, \{5,6,7\}\}$

Corresponding VST schedule on t fast machines:

	0	14	28	50	75
M_1^f	$T_1^a:14$	$T_2^a:14$	$T_9^a:22$	$T_1^b:25$	
	0	14	30	50	75
M_2^f	$T_3^a:14$	$T_4^a:16$	$T_8^a:20$	$T_2^b:25$	
	0	16	32	50	75
M_3^f	$T_5^a:16$	$T_6^a:16$	$T_7^a:18$	$T_3^b:25$	

FIGURE 1. Illustration of the transformation in Theorem 1.

$$m_0^s + m_1^s + 3m_0^f + 3m_1^f \leq 3t \quad (\text{since } C^* = 3t),$$

$$m_0^s + 4m_0^f + 3m_1^f = 3t \quad (\text{since there are } 3t \text{ a-tasks}),$$

$$m_1^s + m_1^f = t \quad (\text{since there are } t \text{ b-tasks}).$$

These relations imply that $m_0^s = m_1^s = m_0^f = 0$, $m_1^f = t$. Since $\sum_{j \in S} q_j^a + \sum_{j=1}^t q_i^b = 3tb = t \cdot \max\{\max_j\{p_j^a\}, \max_i\{p_i^b\}\}$, the t fast machines are busy in the interval $[0, 3b]$. It follows that the i -th one of them processes three tasks T_j^a ($j \in S_i$) with $\sum_{j \in S_i} q_j^a = 2b$. The collection $\{S_1, \dots, S_t\}$ constitutes a solution to 3-PARTITION. \square

3. NP-hardness of the FST problem

THEOREM 2. *The FST problem is NP-hard in the strong sense.*

THEOREM 3. *The FST problem is solvable in $O(n)$ time if all release dates are equal.*

Our NP-hardness proof holds for the case that $C^f/C^s = (t+2)/(t+1)$ and $p_j/q_j = z$ ($j = 1, \dots, n$), where t and z are input variables. Theorem 2 is still true if C^f/C^s is an arbitrary constant between 2 and 3 and $p_j/q_j = 2$ ($j = 1, \dots, n$) [16]; the proof of this further refinement is quite involved. Theorem 3 shows that the NP-hardness result cannot be extended to the case of equal release dates, unless $\mathcal{P} = \mathcal{NP}$.

Proof of Theorem 2. We will start from the following strongly NP-complete problem [5, p.224, [SP17]]:

NUMERICAL MATCHING WITH TARGET SUMS: Given a set $S = \{1, \dots, t\}$ and positive integers $a_1, \dots, a_t, b_1, \dots, b_t, c_1, \dots, c_t$ with $\sum_{i \in S} (a_i + b_i) = \sum_{i \in S} c_i$, do there exist permutations α and β of S such that $a_{\alpha(i)} + b_{\beta(i)} = c_i$ ($i \in S$)?

We may assume without loss of generality that $a_1 < \dots < a_t$, $b_1 < \dots < b_t$ and $c_1 < \dots < c_t$. Further, we will assume that for any instance of this problem there exists a positive integer z such that

$$z < a_1 < \dots < a_t < 2z < b_1 < \dots < b_t < 3z < c_1 < \dots < c_t < 5z.$$

(If this does not hold, then define $z = \max\{a_t+1, b_t+1\}$ and set $a_i \leftarrow a_i+z$, $b_i \leftarrow b_i+2z$, $c_i \leftarrow c_i+3z$ ($i \in S$).) We will use the notation $S' = \{1, \dots, t-1\}$.

Given any instance of NUMERICAL MATCHING WITH TARGET SUMS, we construct a corresponding instance of the FST problem as follows:

- the cost coefficients are defined by $C^S = t+1$, $C^F = t+2$;
- there are $2t^2+t$ tasks:

$$\begin{aligned} \text{a-tasks } T_i^a \quad (i \in S) & \quad \text{with } r_i^a = 0, \quad p_i^a = za_i, \quad q_i^a = a_i, \\ \text{b-tasks } T_{hi}^b \quad (h \in S', i \in S) & \quad \text{with } r_{hi}^b = a_h, \quad p_{hi}^b = zb_i, \quad q_{hi}^b = b_i, \\ \text{c-tasks } T_i^c \quad (i \in S) & \quad \text{with } r_i^c = c_i, \quad p_i^c = 3z^3, \quad q_i^c = 3z^2, \\ \text{d-tasks } T_{hi}^d \quad (h \in S', i \in S) & \quad \text{with } r_{hi}^d = 2z+zb_i, \quad p_{hi}^d = z^3, \quad q_{hi}^d = z^2. \end{aligned}$$

We claim that NUMERICAL MATCHING WITH TARGET SUMS has a solution if and only if there exists a feasible schedule with total costs at most $C^* = t^3+t^2+t$.

Suppose that the matching problem has a solution (α, β) . It is possible to construct a feasible schedule for all tasks on t fast machines M_i^F ($i \in S$) and t^2-t slow machines M_{hi}^S ($h \in S', i \in S$) as follows (cf. Figure 2): for each $i \in S$, machine M_i^F processes the tasks $T_{\alpha(i)}^a, T_{\alpha(i)\beta(i)}^b, T_i^c$ in the intervals $[0, a_{\alpha(i)}], [a_{\alpha(i)}, a_{\alpha(i)}+b_{\beta(i)}], [c_i, c_i+3z^2]$ (note that $a_{\alpha(i)}+b_{\beta(i)} = c_i$), and each of the $t-1$ machines M_{hi}^S ($h \in S'$) processes one of the $t-1$ tasks T_{hi}^b ($h \in S - \{\alpha(\beta^{-1}(i))\}$) in $[a_h, a_h+zb_i]$ and one of the $t-1$ tasks T_{hi}^d ($h \in S'$) in $[2z+zb_i, 2z+zb_i+z^3]$ (note that $a_h < 2z$). The total costs of this schedule are equal to $tC^F + (t^2-t)C^S = C^*$.

Instance of NUMERICAL MATCHING WITH TARGET SUMS:

$t = 3; z = 4;$	i	1	2	3
	a_i	5	6	7
	b_i	9	10	11
	c_i	14	16	18
Solution:	$\alpha(i)$	1	2	3
	$\beta(i)$	1	2	3

Corresponding FST schedule on t fast machines and $t^2 - t$ slow machines:

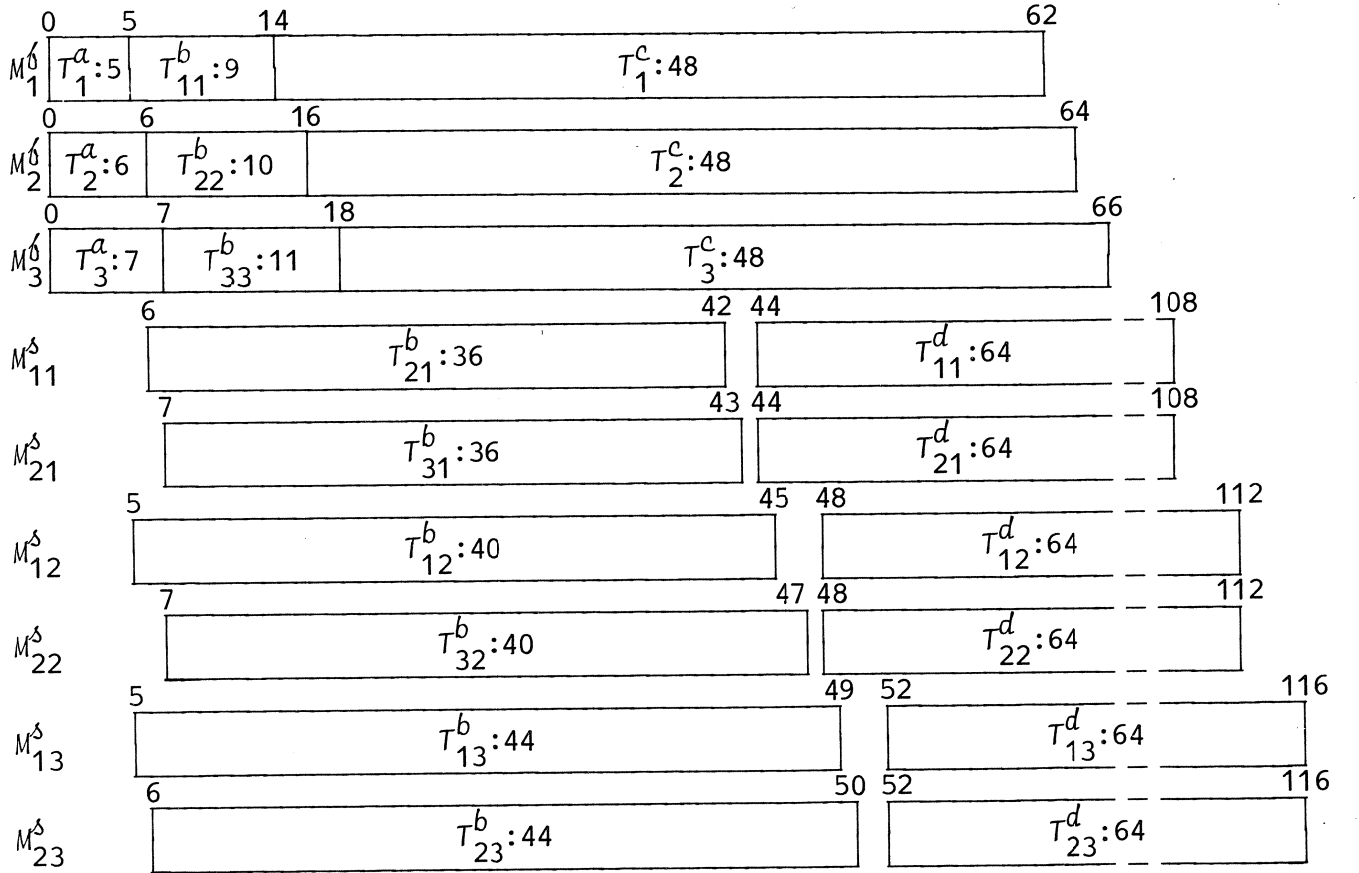


FIGURE 2. Illustration of the transformation in Theorem 2.

Conversely, suppose that there exists a feasible schedule with total costs at most C^* . We make the following propositions.

- (1) Two a-tasks are not assigned to the same machine.

Proof. Each a-task is processed during the interval $[0, z]$.

- (2) Two b-tasks are not assigned to the same machine.

Proof. Each b-task is processed during the interval $[2z, 3z]$.

- (3) Two c- or d-tasks are not assigned to the same machine.

Proof. Each c- or d-task is processed during the interval $[3z^2+z, 3z^2+3z]$.

- (4) An a-task and a b-task are not assigned to the same slow machine.

Proof. On a slow machine, each a- or b-task is processed during the interval $[2z-1, z^2+z]$.

- (5) A b-task and a c-task are not assigned to the same slow machine.

Proof. On a slow machine, each b- or c-task is processed during the interval $[5z-1, 2z^2+2z+1]$.

All tasks are assigned to at most t^2 machines, since $(t^2+1)C^S > C^*$. Propositions (1), (2) and (3) imply that there are exactly t^2 machines, each processing at most one a-task, exactly one b-task and exactly one c- or d-task. These machines include at most t fast ones, since $(t^2-t-1)C^S + (t+1)C^F > C^*$. Propositions (4) and (5) imply that there are exactly t fast machines, each processing one a-task, one b-task and one c-task; hence, there are exactly t^2-t slow machines, each processing one b-task and one d-task.

We denote the t fast machines by M_i^f ($i \in S$) and the t^2-t slow machines by M_{hi}^s ($h \in S'$, $i \in S$). It may be assumed that T_i^c is assigned to M_i^f ($i \in S$) and T_{hi}^d to M_{hi}^s ($h \in S'$, $i \in S$). There exists a permutation α of S such that $T_{\alpha(i)}^a$ is assigned to M_i^f ($i \in S$).

Let us define the size of T_{hi}^b as b_i , its processing time on a fast machine. The size of a b-task on M_i^f is at most $c_i - a_{\alpha(i)}$, and the size of a b-task on M_{hi}^s is at most $\lfloor (2z + zb_i - a_1)/z \rfloor = b_i$. The sum of these upper bounds over all machines is equal to $\sum_{i \in S} (c_i - a_{\alpha(i)}) + \sum_{h \in S', i \in S} b_i = t \sum_{i \in S} b_i$, which is the total size of all b-tasks. It follows that all these upper bounds are actually achieved. More explicitly, for each $i \in S$, there exists an index $\beta(i) \in S$ such that $T_{\alpha(i)\beta(i)}^b$ is assigned to M_i^f , and there exists an index $\gamma(i) \in S$ such that the $t-1$ tasks T_{hi}^b ($h \in S - \{\gamma(i)\}$) are assigned to the $t-1$ machines M_{hi}^s ($h \in S'$), while $T_{\gamma(i)i}^b$ is assigned to a fast machine. This implies that the functions β and γ are permutations of S with $\gamma(\beta(i)) = \alpha(i)$ ($i \in S$).

Since $T_{\alpha(i)\beta(i)}^b$ leaves no idle time between $T_{\alpha(i)}^a$ and T_i^c on M_i^f , we have $a_{\alpha(i)} + b_{\beta(i)} = c_i$ ($i \in S$). The pair (α, β) constitutes a solution to 3-PARTITION. \square

Proof of Theorem 3. In the FST problem with equal release dates, each task has to start at the same time and therefore each machine can process at most one task. It follows that an optimal schedule uses n slow machines and has total costs nC^S . It is constructed in $O(n)$ time. \square

4. A well-solvable case of the VST problem

THEOREM 4. *In the case that $q_j = 1$ ($j = 1, \dots, n$), the VST problem is solvable in $O(n \log n)$ time if all release dates are equal.*

The complexity of the VST problem with all $q_j = 1$ and arbitrary release dates remains unresolved (cf. Table 1).

Proof of Theorem 4. In the VST problem with equal release dates, a slow machine can process at most one task but a fast machine may be able to process more than one.

Let us assume that there are m fast machines, with $0 \leq m \leq n$, and let X_m denote the maximum number out of the n unit-time tasks that can be completed in time on these machines. A schedule using m fast machines has to use $n - X_m$ slow machines; its total costs are equal to $C_m = mC^f + (n - X_m)C^s$. It follows that an optimal schedule has total costs $\min_{0 \leq m \leq n} \{C_m\}$.

For each given value of m , the number X_m and a corresponding schedule on m fast machines can be found by an $O(n \log n)$ algorithm due to Lawler [14;7,p.295]. Straightforward application of this algorithm for $m = 0, \dots, n$ would yield an overall optimal schedule in $O(n^2 \log n)$ time.

However, all X_0, \dots, X_n together can be determined by an $O(n \log n)$ algorithm, which constructs a schedule on n fast machines with the property that, for any value of m , the partial schedule on the first m machines is an optimal schedule on m machines [13]. This algorithm considers the tasks in order of nondecreasing deadlines and assigns each task to the machine with lowest index on which it can be completed in time. A formal statement is as follows.

VST ALGORITHM (*only fast machines, all $q_j = 1$, all $r_j = 0$*)

Initialize. Reorder the tasks in such a way that $d_1 \leq \dots \leq d_n$; set $d_0 \leftarrow -\infty$.

Introduce an array x of size n and set $x_m \leftarrow 0$ ($m = 1, \dots, n$) [x_m tasks have been assigned to machine M_m^f].

Introduce an array μ of size n [T_j will be assigned to $M_{\mu_j}^f$].

Set $m \leftarrow 1$.

Iterate. for $j \leftarrow 1$ to n do
 begin
 set $m \leftarrow$ if $d_{j-1} < d_j$ then 1 else if $x_m < d_j$ then m else $m+1$;
 set $\mu_j \leftarrow m$, $x_m \leftarrow x_m + 1$
 end.
Finalize. Set $X_0 \leftarrow 0$; for $m \leftarrow 1$ to n do set $X_m \leftarrow X_{m-1} + x_m$.

It can be shown that X_m is the maximum number of tasks that can be completed in time on m fast machines, for $m = 0, \dots, n$ [13]. The algorithm requires $O(n \log n)$ time to order the tasks, and $O(n)$ time to construct the schedule and to determine the values X_0, \dots, X_n . It follows that an overall optimal schedule is obtained in $O(n \log n)$ time. \square

Note. Since $x_m \geq x_{m+1}$ ($m = 1, \dots, n-1$), X_m is a concave function of m , so that C_m is convex. A similar observation will be exploited in the next section. \square

5. A well-solvable case of the FST problem

THEOREM 5. *In the case that $q_j = 1$ ($j = 1, \dots, n$), the FST problem is solvable in $O(n \log n)$ time.*

The assumption that all $q_j = 1$ is too strong: an analysis of the proof below shows that our algorithm is applicable in the more general situation that the q_j are bounded from above by the minimum length of the interval between two different adjacent release dates. Although this restriction still limits the practical value of our result, we feel that the insight gained might be useful in the design of approximation algorithms for the general FST problem.

Proof of Theorem 5. The development of our algorithm will proceed along the same lines as in the previous section. First, we will assume that there are m fast machines and we will determine an optimal set of tasks to be scheduled on these machines. Next, we will compute the minimum number of slow machines needed to execute the remaining tasks. Finally, we will describe an efficient method to find the optimal value of m .

We start by representing the problem data in a convenient way. Suppose that the release dates assume k different values $\bar{r}_1, \dots, \bar{r}_k$ with $\bar{r}_1 < \dots < \bar{r}_k$. For $j = 1, \dots, k$, there are n_j tasks $T_{1j}, \dots, T_{n_j j}$ with release dates $r_{1j} = \dots = r_{n_j j} = \bar{r}_j$ and deadlines $d_{1j} \geq \dots \geq d_{n_j j}$. We have $n = \sum_{j=1}^k n_j$ and define $n' = \max_{1 \leq j \leq k} \{n_j\}$. This representation can be obtained by sorting the release dates and the deadlines in $O(n \log n)$ time and applying a bucket sort [1] to order the tasks with the same release date according to deadlines in $O(n)$ time.

Let us now assume that there are m fast machines M_1^f, \dots, M_m^f , with $0 \leq m \leq n'$. For $j = 1, \dots, k$, each of these machines can process exactly one of the tasks $T_{1j}, \dots, T_{n_j j}$. It is obviously optimal to assign T_{ij} to M_i^f for $j = 1, \dots, k$ and $i = 1, \dots, \min\{n_j, m\}$, so that the remaining tasks will be as short as possible. Let \mathcal{T}_m denote the set of tasks that are not assigned to the m fast machines, where $\mathcal{T}_0 = \{T_1, \dots, T_n\}$ and $\mathcal{T}_{n'} = \emptyset$, and let ℓ_m denote the minimum number of slow machines needed to execute these tasks. A schedule using m fast machines has total costs $C_m = mC^f + \ell_m C^s$. It follows that an optimal schedule uses m^* fast machines, where $C_m^* = \min_{0 \leq m \leq n'} \{C_m\}$.

For each given value of m , the number ℓ_m and a corresponding schedule of the tasks in \mathcal{T}_m on ℓ_m slow machines can be found in $O(n \log n)$ time. This problem has already been discussed in the first two paragraphs of Section 1. The following algorithm is a slight modification of the channel assignment

algorithm of Gupta, Lee and Leung [8]; for simplicity, it is stated for the case that $m = 0$.

FST ALGORITHM (*only slow machines*)

Initialize. Reorder the tasks in such a way that $r_1 \leq \dots \leq r_n$; determine a permutation δ of $\{1, \dots, n\}$ such that $d_{\delta(1)} \leq \dots \leq d_{\delta(n)}$.

Introduce a stack S of size n and push machine indices $1, \dots, n$ onto S in such a way that m is on top of $m+1$ ($m = 1, \dots, n-1$).

Introduce an array λ of size n [T_j will be assigned to $M_{\lambda_j}^S$].

Set $j \leftarrow 1, i \leftarrow 1$.

Iterate. while $j \leq n$ do

if $r_j < d_{\delta(i)}$

then begin set $\lambda_j \leftarrow$ top element of S ; pop S ; set $j \leftarrow j+1$ end

else begin push $\lambda_{\delta(i)}$ onto S ; set $i \leftarrow i+1$ end.

Finalize. Set $\ell_0 \leftarrow \max_{1 \leq j \leq n} \{\lambda_j\}$.

It can be shown that ℓ_0 is the minimum number of slow machines needed to execute all tasks. The algorithm requires $O(n \log n)$ time to order the tasks, and $O(n)$ time to construct the schedule and to compute the value ℓ_0 . Since the release dates and the deadlines have already been sorted, each application of this algorithm requires only $O(n)$ time. Straightforward computation of ℓ_m for $m = 0, \dots, n'$ would yield an overall optimal schedule in $O(n \log n + n'n) = O(n^2)$ time.

However, it will be shown below that C_m is a convex function of m , and this property can be exploited to arrive at an $O(n \log n)$ algorithm. The convexity of C_m implies that, if $C_m < C_{m+1}$, then $m^* \in \{0, \dots, m\}$, and else $m^* \in \{m+1, \dots, n'\}$. Thus, m^* can be found by a bisection search as follows: for $m = \lfloor \frac{1}{2}n' \rfloor$, compute C_m and C_{m+1} , reduce the domain of m^* by a factor of

two by eliminating either $[0, m]$ or $[m+1, n']$, and repeat the procedure on the remaining interval. The optimal value of m is found in at most $\lceil \log_2(n'+1) \rceil$ iterations.

The entire algorithm requires $O(n \log n)$ time to sort the release dates and the deadlines and, for each of $O(\log n')$ values of m , $O(n)$ time to compute C_m . It follows that an overall optimal schedule is obtained in $O(n \log n)$ time.

It remains to be shown that C_m is a convex function of m . Since $C_m = mC^f + \ell_m C^s$, we have to prove that ℓ_m is convex, or equivalently that

$$(1) \quad \ell_{m-1} - \ell_m \geq \ell_m - \ell_{m+1} \quad (m = 1, \dots, n'-1).$$

We define the *degree of overlap* of the set \mathcal{S} at time t as the number of tasks $T_j \in \mathcal{S}$ such that $t \in [r_j, d_j]$. Let $X_m(t)$ denote the degree of overlap of \mathcal{T}_m at t and $x_{m-1}(t)$ the degree of overlap of $\mathcal{T}_{m-1} - \mathcal{T}_m$ at t , i.e., $x_{m-1}(t) = X_{m-1}(t) - X_m(t)$. It is known [9] that

$$(2) \quad \ell_m = \max_t \{X_m(t)\} \quad (m = 0, \dots, n').$$

Since the number of tasks $T_j \in \mathcal{T}_{m-1} - \mathcal{T}_m$ and the lengths of their intervals $[r_j, d_j]$ do not increase as m increases, it is also true that

$$(3) \quad x_{m-1}(t) \geq x_m(t) \quad (\text{all } t; m = 0, \dots, n'-1).$$

Defining t_m such that $X_m(t_m) = \max_t \{X_m(t)\}$ ($m = 0, \dots, n'$) and applying (2), we rewrite (1) as

$$X_{m-1}(t_{m-1}) - X_m(t_m) \geq X_m(t_m) - X_{m+1}(t_{m+1}).$$

We have for the left-hand side that

$$X_{m-1}(t_{m-1}) - X_m(t_m) = X_{m-1}(t_{m-1}) - X_{m-1}(t_m) + x_{m-1}(t_m) \geq x_{m-1}(t_m).$$

Similarly, we have for the right-hand side that

$$X_m(t_m) - X_{m+1}(t_{m+1}) = X_{m+1}(t_m) + x_m(t_m) - X_{m+1}(t_{m+1}) \leq x_m(t_m).$$

Application of (3) for $t = t_m$ now implies the validity of (1). This completes the proof of Theorem 5. \square

Note. By means of ingenious counting techniques, the above algorithm for computing a single value ℓ_m can be extended to an $O(n \log n)$ algorithm for computing all ℓ_0, \dots, ℓ_n , together [13]; when the data have already been sorted, it requires only $O(n)$ time, as before. A similar result has been used in the previous section. \square

Acknowledgments

This research was in part supported by the National Science Foundation Grant ENG79-09724. The authors gratefully acknowledge constructive suggestions by B.J. Lageweg.

References

1. A.V. AHO, J.E. HOPCROFT, J.D. ULLMAN (1974) *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA.
2. G.B. DANTZIG, D.R. FULKERSON (1954) Minimizing the number of tankers to meet a fixed schedule. *Naval Res. Logist. Quart.* 1, 217-222.
3. L.R. FORD, JR., D.R. FULKERSON (1962) *Flows in Networks*, Princeton University Press, Princeton, NJ.
4. M.R. GAREY, D.S. JOHNSON (1978) "Strong" NP-completeness results: motivation, examples and implications. *J. Assoc. Comput. Mach.* 25, 499-508.
5. M.R. GAREY, D.S. JOHNSON (1979) *Computers and Intractability: a Guide to the Theory of NP-Completeness*, Freeman, San Francisco.
6. I. GERTSBAKH, H.I. STERN (1978) Minimal resources for fixed and variable job schedules. *Oper. Res.* 26, 68-85.
7. R.L. GRAHAM, E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN (1979) Optimization and approximation in deterministic sequencing and scheduling:

- a survey. *Ann. Discrete Math.* 5, 287-326.
8. U.I. GUPTA, D.T. LEE, J.Y.-T. LEUNG (1979) An optimal solution for the channel-assignment problem. *IEEE Trans. Comput.* C-28, 807-810.
 9. A. HASHIMOTO, J.E. STEVENS (1970) Path cover of acyclic graphs. ILLIAC IV, Document 239, University of Illinois, Urbana, IL.
 10. A. HASHIMOTO, J.E. STEVENS (1971) Wire routing by optimizing channel assignment within large apertures. *Proc. 8th Design Automation Workshop*, 155-169.
 11. R.M. KARP (1972) Reducibility among combinatorial problems. In: R.E. MILLER, J.W. THATCHER (eds.) (1972) *Complexity of Computer Computations*, Plenum Press, New York, 85-103.
 12. B.W. KERNIGHAN, D.G. SCHWEIKERT, G. PERSKY (1973) An optimum channel-routing algorithm for polycell layouts of integrated circuits. *Proc. 10th Design Automation Workshop*, 50-59.
 13. B.J. LAGEWEG (1980) Personal communication.
 14. E.L. LAWLER (1976) Sequencing to minimize the weighted number of tardy jobs. *RAIRO Inform.* 10.5 Suppl. 27-33.
 15. J.K. LENSTRA, A.H.G. RINNOOY KAN (1979) Complexity of vehicle routing and scheduling problems. Report BW 111, Mathematisch Centrum, Amsterdam; to appear in *Networks*.
 16. K. NAKAJIMA (1980) *On Nonpreemptive Multiprocessor Scheduling with Discrete Starting Times*, Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL.
 17. K. NAKAJIMA, S.L. HAKIMI (1979) On the NP-completeness of a real-time scheduling problem with two types of machines. *Proc. 17th Allerton Conf. Communication, Control, and Computing*, University of Illinois, Urbana, IL, 652-658.