Theoretical Computer Science 75 (1990) 347-355
North-Holland

NOTE

MINIMIZING MEAN FLOW TIME WITH RELEASE TIME CONSTRAINT*

Jianzhong DU, Joseph Y.-T. LEUNG and Gilbert H. YOUNG
Computer Science Program, University of Texas at Dallas, Richardson, TX 75083, U.S.A.

Communicated by F. Yao
Received September 1988
Revised September 1989

Abstract. We consider the problem of preemptively scheduling a set of n independent tasks with release times on m identical processors with the objective of minimizing the mean flow time. For one processor, Baker gives an $O(n \log n)$ time algorithm to find an optimal schedule. Lawler asks the question whether the problem can be shown to be solvable in polynomial time or shown to be NP-hard for $m \geq 2$. In this paper we answer this question by showing that it is NP-hard for each fixed $m \geq 2$.

1. Introduction

We consider the problem of preemptively scheduling a set $\{T_1, T_2, \ldots, T_n\}$ of n *independent tasks* on $m \geq 1$ *identical processors* with the objective of minimizing the *mean flow time.* Each task T_i has associated with it a *release time* $r(T_i)$ and an *execution time* $e(T_i)$. The tasks are to be preemptively scheduled on the processors under the constraint that no task can start before its release time. If S is a schedule of the n tasks on the m processors, then the finishing time of T_i in S is denoted by $f(S, T_i)$, and the mean flow time of S, denoted by MFT(S), is defined to be $\mathrm{MFT}(S) = \sum_{i=1}^{n} f(S, T_i)$. Our goal is to find a schedule S_O such that $\mathrm{MFT}(S_O) \leq \mathrm{MFT}(S)$ for all schedules S. Such a schedule will be called an *optimal schedule.*

For nonpreemptive scheduling, Lenstra [7] has shown that finding an optimal schedule is NP-hard even for one processor. Thus, unless P = NP, there is no hope of solving this problem efficiently. The problem appears to be easier for preemptive scheduling. Baker [1] has given an $O(n \log n)$ time algorithm to find an optimal schedule for one processor. Herrbach and Leung [4] have given an $O(n \log n)$ time algorithm to solve the special case of two processors and identical execution times. In [5] Lawler asks the question whether the problem can be shown to be solvable in polynomial time or shown to be NP-hard for $m \geq 2$. In this paper we answer this question by showing that it is NP-hard for each fixed $m \geq 2$.

* Research supported in part by the ONR Grant N00014-87-K0833.

A more general problem is the one when each task T_i has an additional *deadline* $d(T_i)$ associated with it, and it is required that each task T_i be scheduled within the interval $[r(T_i), d(T_i)]$. For preemptive scheduling on one processor, Smith [8] has given an $O(n \log n)$ time algorithm for the special case of identical release times, while Baker's algorithm [1] solves the special case of identical deadlines. The complexities of these two special cases are not known for $m \geq 2$ processors. By choosing a large common deadline, our NP-hardness proof shows that the special case studied by Baker is NP-hard for each fixed $m \geq 2$. For arbitrary release times, deadlines and execution times, Du and Leung [2] have recently shown that the problem is NP-hard for each fixed $m \geq 1$. Furthermore, they give a polynomial-time algorithm to solve a large class of task systems that includes the special cases studied by Smith and Baker as well as the class of equal-execution-time task systems. For nonpreemptive scheduling on one processor, Smith's algorithm [8] solves the special case of identical release times, while Lenstra's NP-hardness proof [7] shows that the special case of identical deadlines is NP-hard. For a survey on this and related problems, the readers are referred to the survey papers by Lawler [5] and Lawler et al. [6].

2. NP-hardness proof

In this section we show that the problem of finding an optimal schedule is NP-hard by showing the decision version of the problem to be NP-complete. The decision problem with parameter m is defined as follows.

MFTRTP(m): Given an integer ω, m identical processors, and a set $T = \{T_i\}$ of n independent tasks with integer release times $\{r(T_i)\}$ and integer execution times $\{e(T_i)\}$, is there a preemptive schedule S of T on m processors such that $\text{MFT}(S) \leq \omega$?

We first show that MFTRTP(2) is NP-complete. The proof can readily be generalized to $m > 2$. To show MFTRTP(2) to be NP-complete, we reduce to it the following NP-complete problem [3].

Partition: Given a set of z positive integers $A = \{a_1, a_2, \ldots, a_z\}$, is there a partition of A into two subsets A_1 and A_2 such that $\sum_{a_i \in A_1} a_i = \sum_{a_i \in A_2} a_i$?

We begin by giving a reduction from PARTITION to MFTRTP(2). Let $A = \{a_1, a_2, \ldots, a_z\}$ be an instance of PARTITION, and let $B = \sum_{i=1}^{z} a_i$. Without loss of generality, we may assume that each a_i is an integral multiple of 2 and less than $\frac{1}{2}B$. We construct a set T of $3(z+1)$ independent tasks as follows. $T = \{U_i, V_i \mid 1 \leq i \leq z\} \cup \{W_i \mid 0 \leq i \leq z+2\}$. The first set is the set of *partition tasks*, and the release times and execution times of the tasks are given as follows. For each

$1 \le i \le z$, $r(U_i) = r(V_i) = 2(i-1)B^2$ and $e(U_i) = e(V_i) = a_iB$. Note that U_i and V_i have the same release times and the same execution times. The second set is the set of *penalty tasks*, and the release times and execution times of the tasks are given as follows. For each $1 \le i \le z$, $r(W_i) = r(U_i) + e(U_i)$ and $e(W_i) = a_i$. $r(W_0) = 0$, $r(W_{z+1}) = r(W_{z+2}) = 2zB^2 + \frac{1}{2}B^2$, and $e(W_0) = e(W_{z+1}) = e(W_{z+2}) = 2zB^2$. Note that W_i, $1 \le i \le z$, are small tasks, and W_0, W_{z+1} and W_{z+2} are large tasks. Furthermore, W_{z+1} and W_{z+2} can both start at their release times only if W_0 finishes by that time. Also, the earliest time W_0 can finish is $2zB^2 = r(W_{z+1}) - \frac{1}{2}B^2$. Figure 1 shows the release time pattern of the tasks in T.

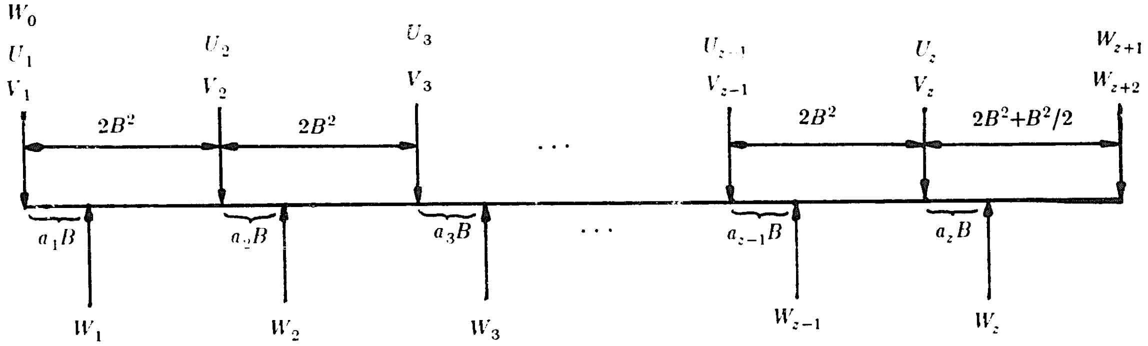

Fig. 1. Release time pattern of the tasks in T.

Let S_O be an optimal schedule of T on two identical processors. In the following we will characterize the structure of S_O. First, we need to introduce the following notations. Let S be a schedule of T on two processors. A task X is said to be *delayed* in S if $f(S, X) > r(X) + e(X)$. For each $1 \le i \le z$, the partial schedule of S restricted to the time interval $[2(i-1)B^2, 2iB^2]$ is called the ith block of S, denoted by $\mathrm{BLK}_i(S)$. Let L_i denote the total amount of W_0 executed in $\mathrm{BLK}_i(S_O)$ for each $1 \le i \le z$. The next three lemmas characterize the structure of $\mathrm{BLK}_i(S_O)$.

Lemma 2.1. *For each $1 \le i \le z$, U_i, V_i and W_i must be finished in* $\mathrm{BLK}_i(S_O)$. *Furthermore, U_i is not delayed in S_O.*

Proof. For each $1 \le i \le z$, we can execute a total of $4B^2$ amount of tasks in $\mathrm{BLK}_i(S_O)$, and the maximum amount of W_0 that can be executed in $\mathrm{BLK}_i(S_O)$ is $2B^2$. Therefore, we can execute a total of at least $2B^2$ amount of the tasks U_i, V_i and W_i in $\mathrm{BLK}_i(S_O)$. Since the total execution time of these three tasks is less than $2B^2$, they must be finished in $\mathrm{BLK}_i(S_O)$. This proves the first part of the lemma. For the second part, we observe that the maximum amount of W_0 that can be executed in the time interval $[r(U_i), r(U_i) + e(U_i)]$ of $\mathrm{BLK}_i(S_O)$ is a_iB, and hence we can execute a total of at least a_iB amount of U_i and V_i in the same interval. Since U_i and V_i are identical tasks, we can transform S_O without increasing the mean flow time such that U_i finishes at time $r(U_i) + e(U_i)$. The transformation can be done as follows. By renaming the two tasks if necessary, we may assume that $f(S_O, U_i) \le f(S_O, V_i)$.

Suppose there is an interval $IX=[t, t+\alpha]$ within the time interval $[r(U_i), r(U_i)+e(U_i)]$ such that U_i is not executing in IX, and there is an interval $IY=[t', t'+\alpha]$ within the time interval $[r(U_i)+e(U_i), f(S_O, U_i)]$ such that U_i is executing in IY. Then, we interchange the tasks executing in IX with those executing in IY. This is always possible unless W_i is executing in IY. In this case, we simply interchange U_i in IY with one of the tasks in IX. It is easy to see that the transformation cannot increase the mean flow time of the schedule. Thus, U_i is not delayed in S_O. □

Lemma 2.2. *For each $1 \leq i \leq z$, task W_0 must execute continuously in the time interval $[2iB^2 - L_i, 2iB^2]$ in* $\mathrm{BLK}_i(S_O)$.

Proof. By Lemma 2.1, U_i executes continuously in the time interval $[2(i-1)B^2, 2(i-1)B^2+a_iB]$ in $\mathrm{BLK}_i(S_O)$. By interchanging processors if necessary, we may assume that U_i executes continuously on processor P_2. Thus, V_i and W_0 are the only tasks that can possibly execute on processor P_1 in the same time interval. We now concentrate on the interval $[2(i-1)B^2+a_iB, 2iB^2]$ in $\mathrm{BLK}_i(S_O)$. Again, by interchanging processors if necessary, we may assume that W_0 executes only on P_1 in this interval. The tasks that can possibly execute in this interval in $\mathrm{BLK}_i(S_O)$ are W_0, V_i and W_i. Thus, if there is a time slice in this interval during which P_1 is not executing W_0, then P_1 must be either idle, or it is executing V_i or W_i. By interchanging processors if necessary, we may assume that P_1 is either idle, or it is executing V_i during the time slice. Therefore, during the interval $[2(i-1)B^2, 2iB^2]$ in $\mathrm{BLK}_i(S_O)$, P_1 is either idle, or it is executing V_i or W_0. If we now right justify the execution of W_0 on P_1 and left justify the execution of V_i on P_1 in this interval, then the mean flow time of the schedule cannot be increased. This proves the lemma. □

Lemma 2.3. *For each $1 \leq i \leq z$, we have $L_i \geq 2B^2 - a_iB$. Furthermore,* $\mathrm{BLK}_i(S_O)$ *must be one of the three schedules shown in Fig.* 2.

Proof. By Lemma 2.1, U_i executes continuously from time $2(i-1)B^2$ until $2(i-1)B^2+a_iB$ in $\mathrm{BLK}_i(S_O)$. Without loss of generality, we may assume that U_i executes continuously on processor P_2 in $\mathrm{BLK}_i(S_O)$. By Lemma 2.2, W_0 executes continuously in the time interval $[2iB^2 - L_i, 2iB^2]$ in $\mathrm{BLK}_i(S_O)$. We may assume that W_0 executes continuously on processor P_1 in this interval in $\mathrm{BLK}_i(S_O)$. If $L_i < 2B^2 - a_iB$, then V_i must be executing continuously in the time interval $[2(i-1)B^2, 2(i-1)B^2+a_iB]$ in $\mathrm{BLK}_i(S_O)$. For otherwise, we can reschedule V_i so that this condition is satisfied, and the mean flow time of the schedule will be decreased, contradicting the fact that S_O is optimal. Consequently, W_i is the only task that can be executing in the time interval $[2(i-1)B^2+a_iB, 2iB^2-L_i]$ in $\mathrm{BLK}_i(S_O)$. Since there are two processors and W_i is the only task executing in this interval, we can reassign the last portion of W_0 (i.e. the portion of W_0 executing in the interval $[f(S_O, W_0)-(2B^2-L_i-a_iB), f(S_O, W_0)]$ in S_O) to this interval with a

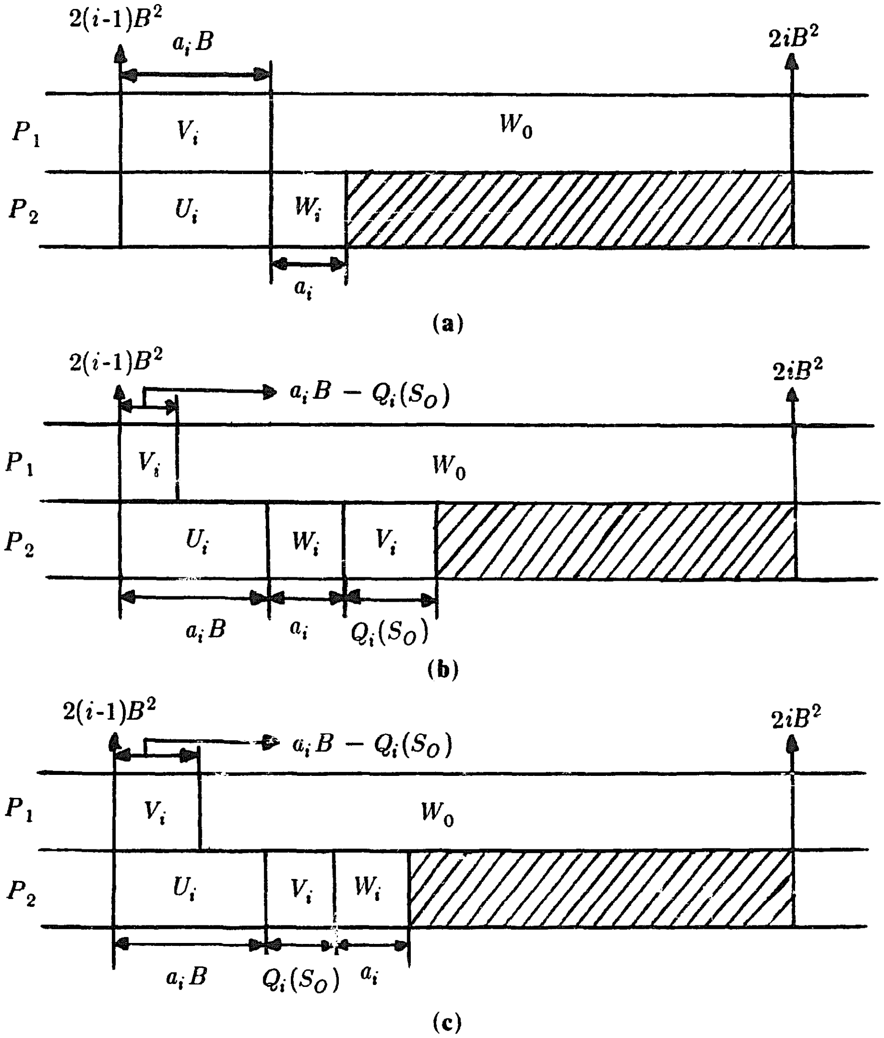

Fig. 2. Three schedules of $\mathrm{BLK}_i(S_\mathrm{O})$. (a) Normal form; (b) normal form ($a_i \leq Q_i(S_\mathrm{O}) \leq a_iB$); (c) offset form ($0 < Q_i(S_\mathrm{O}) < a_i$).

net decrease in mean flow time. This contradicts the fact that S_O is optimal. Thus, it is impossible to have $L_i < 2B^2 - a_iB$, and hence the first part of the lemma is proved.

From the previous discussions, we know that U_i executes continuously on P_2 in the interval $[r(U_i), r(U_i) + e(U_i)]$ and W_0 executes continuously on P_1 in the interval $[2iB^2 - L_i, 2iB^2]$ in $\mathrm{BLK}_i(S_\mathrm{O})$. If $L_i = 2B^2 - a_iB$, then $\mathrm{BLK}_i(S_\mathrm{O})$ must be the schedule shown in Fig. 2a. Otherwise, we have $L_i > 2B^2 - a_iB$ and $\mathrm{BLK}_i(S_\mathrm{O})$ must be one of the schedules shown in Figs. 2b, c, depending on whether the portion of V_i executed on P_2 is larger than a_i or not. □

By Lemma 2.3, we know that each $\mathrm{BLK}_i(S_\mathrm{O})$ is one of the three schedules shown in Fig. 2. Let S be a schedule of T on two processors such that each $\mathrm{BLK}_i(S)$ is one of the three schedules shown in Fig. 2. We say that $\mathrm{BLK}_i(S)$ is in *normal form* if it is one of the schedules shown in Figs. 2a, b, and $\mathrm{BLK}_i(S)$ is in *offset form* if it is the schedule shown in Fig. 2c. Let $Q_i(S)$ denote the total amount of V_i executed on processor P_2 in $\mathrm{BLK}_i(S)$ for each $1 \leq i \leq z$. Clearly, we have $0 \leq Q_i(S) \leq a_iB$. Note that if $\mathrm{BLK}_i(S)$ is in normal form, then W_i is not delayed in S. If $\mathrm{BLK}_i(S)$

is in offset form, then W_i is delayed in S and $Q_i(S) < a_i$. The next lemma shows that S_O can be transformed into another schedule such that each block is in normal form and W_0 finishes at $2zB^2 + \frac{1}{2}B^2$.

Lemma 2.4. *There is a schedule S' such that* $\mathrm{MFT}(S') \leqslant \mathrm{MFT}(S_O)$, $f(S', W_0) = 2zB^2 + B^2/2$, *and* $\mathrm{BLK}_i(S')$ *is in normal form for each* $1 \leqslant i \leqslant z$.

Proof. S' is obtained from S_O by the algorithm given below. The idea is that if $f(S_O, W_0) > 2zB^2 + \frac{1}{2}B^2$, then we transfer the portions of W_0 executed after $2zB^2 + \frac{1}{2}B^2$ to earlier blocks. This is made possible by moving the portions of V_i executed on processor P_1 to processor P_2. The transformation will decrease the finishing times of W_0 and W_{z+1}. (Since W_{z+1} and W_{z+2} are identical tasks. we may assume that W_{z+1} starts immediately after W_0 finishes in S_O.) Furthermore, it will increase the finishing times of at most two tasks (namely, V_i and W_i) by the same amount. Thus, the mean flow time of the schedule cannot be increased. On the other hand, if $f(S_O, W_0) < 2zB^2 + \frac{1}{2}B^2$, then we transfer the portions of W_0 executed in earlier blocks to the end and we move the portions of V_i executed on P_2 to the intervals vacated by W_0. This transformation will increase the finishing time of W_0, and decrease the finishing time of V_i, and possibly W_i, by the same amount. Again, the mean flow time of the schedule cannot be increased. After this, we simply transform the offset blocks to normal blocks.

Case 1: $f(S_O, W_0) > 2zB^2 + \frac{1}{2}B^2$. Let $Z = f(S_O, W_0) - (2zB^2 + \frac{1}{2}B^2)$. The transformation is described as follows. In the following, we assume that if the last portion of W_0 is moved to an earlier block, then the task that follows it will be shifted left by the same amount.

(1) Set i to be 1.

(2) If $a_iB - Q_i(S_O) \leqslant Z$, then move all of V_i executed on P_1 to P_2. If $\mathrm{BLK}_i(S_O)$ is the schedule shown in Fig. 2a, then insert V_i in front of W_i; otherwise, expand the execution of V_i on P_2 by the amount $a_iB - Q_i(S_O)$ and push any tasks that follow it to the right. Move the last portion of W_0 to fill up the interval vacated by V_i. Decrement Z by $a_iB - Q_i(S_O)$ and set $Q_i(S')$ to be a_iB. This transformation will decrease the finishing times of two tasks and increase the finishing times of at most two tasks by the same amount. Therefore, the mean flow time of the schedule cannot be increased. GOTO (5).

(3) If $a_iB - Q_i(S_O) > Z$ and $Q_i(S_O) = 0$, then move the rightmost Z amount of V_i executed on P_1 to P_2, inserting it in front of W_i. Move the last portion of W_0 to fill up the interval vacated by V_i. Set Z to be 0 and set $Q_i(S')$ to be Z. It is easy to see that the mean flow time of the schedule cannot be increased. GOTO (5).

(4) If $a_iB - Q_i(S_O) > Z$ and $Q_i(S_O) > 0$, then move the rightmost Z amount of V_i executed on P_1 to P_2. The execution of V_i on P_2 will be expanded by Z amount, pushing any tasks that follow it to the right. Set Z to be 0 and set $Q_i(S')$ to be $Q_i(S_O) + Z$. Again, the mean flow time of the schedule cannot be increased.

(5) If $Z > 0$, then increment i by 1 and GOTO (2).

(6) The remaining blocks of S' are the same as those of S_O. Now, for each $1 \leq i \leq z$, if $\mathrm{BLK}_i(S')$ is in offset form and $Q_i(S') \geq a_i$, then convert it to a normal form by interchanging V_i with W_i on P_2. This interchange cannot increase the mean flow time of the schedule.

Case 2. $f(S_O, W_0) < 2zB^2 + \frac{1}{2}B^2$. Let $Z = 2zB^2 + \frac{1}{2}B^2 - f(S_O, W_0)$. The transformation is given below. In the following, we assume that if the last portion of V_i is moved from P_2 to P_1, then the tasks that follow it will be shifted left by the same amount.

(1) Set i to be 1.

(2) Let $Y = \min\{Z, Q_i(S_O)\}$. Move the leftmost Y amount of W_0 to the end, and move the rightmost Y amount of V_i executed on P_2 to the interval vacated by W_0. Decrement Z by Y and set $Q_i(S')$ to be $Q_i(S_O) - Y$. This transformation will increase the finishing time of W_0 by Y, and decrease the finishing time of V_i, and possibly W_i, by at least the same amount. Thus, the mean flow time of the schedule cannot be increased.

(3) If $Z > 0$, then increment i by 1 and GOTO (2).

(4) The remaining blocks of S' are the same as those of S_O. Now, for each $1 \leq i \leq z$, if $\mathrm{BLK}_i(S')$ is in offset form and $Q_i(S') \geq a_i$, then convert it to a normal form by interchanging V_i with W_i on P_2. This interchange cannot increase the mean flow time of the schedule.

We now show that $\mathrm{BLK}_i(S')$ is in normal form for each $1 \leq i \leq z$. Let $E = \{i \mid \mathrm{BLK}_i(S') \text{ is in offset form}\}$ and $F = \{i \mid \mathrm{BLK}_i(S') \text{ is in normal form and } Q_i(S') > 0\}$. We claim that if E is not empty, then $Q_k(S') = a_k B$ for each $k \in F$. Suppose not. Then there exist a $j \in E$ such that $0 < Q_j(S') < a_j$, and a $l \in F$ such that $a_l \leq Q_l(S') < a_l B$. We can transform $\mathrm{BLK}_j(S')$ into a new block (which is still in offset form) such that V_j executes α amount less on P_2, and transform $\mathrm{BLK}_l(S')$ into a new block (which is still in normal form) such that V_l executes α amount more on P_2, where α is an arbitrary small positive number. The resulting schedule has a net decrease of α in mean flow time, since the finishing times of V_j and W_j have been decreased by α while the finishing time of V_l has been increased by the same amount. This contradicts the fact that S' is optimal, and hence proving our claim. Now, $\sum_{i \in E} Q_i(S') < \sum_{i \in E} a_i \leq B$. Furthermore, $\sum_{i \in F} Q_i(S')$ must be an integral multiple of B since $Q_i(S') = a_i B$ for each $i \in F$. Therefore, $\sum_{i \in E} Q_i(S') + \sum_{i \in F} Q_i(S')$ cannot be an integral multiple of B. This contradicts the fact that $\sum_{i \in E} Q_i(S') + \sum_{i \in F} Q_i(S') = \frac{1}{2}B^2$ is an integral multiple of B, since B is an integral multiple of 2. Thus, E must be empty. □

By Lemma 2.4, we may assume that S_O satisfies the properties of S'. The next lemma gives a lower bound for the mean flow time of S_O.

Lemma 2.5. *Let* $D = \{i \mid V_i$ *is delayed in* $\mathrm{BLK}_i(S_O)\}$. *Then, we have*

$$\mathrm{MFT}(S_O) = (3z^2 + 7z + 5)B^2 + B + \sum_{i \in D} a_i \geq (3z^2 + 7z + 5)B^2 + \tfrac{3}{2}B.$$

Moreover, the lower bound is attained only if each $\mathrm{BLK}_i(S_O)$ *is one of the schedules shown in Fig. 3.*

Proof. It is easy to compute $\mathrm{MFT}(S_O)$ by observing that $f(S_O, W_0) = 2zB^2 + \frac{1}{2}B^2$, $f(S_O, W_{z+1}) = f(S_O, W_{z+2}) = 4zB^2 + \frac{1}{2}B^2$, and each $\mathrm{BLK}_i(S_O)$ is one of the schedules shown in Figs. 2a, b. We leave the routine calculation to the readers. The lower bound of $\mathrm{MFT}(S_O)$ depends only on the lower bound of $\sum_{i \in D} a_i$. Let DT denote $\sum_{i \in D} a_i$ and QT denote $\sum_{i \in D} Q_i(S_O)$. Clearly, we have $QT = \frac{1}{2}B^2$. For each $i \in D$, $Q_i(S_O)$ will be added to QT while a_i will be added to DT. Since $Q_i(S_O)$ is at most a_iB, it is easy to see that every B units of QT contribute at least one unit of DT. Thus, the lower bound of DT is $\frac{1}{2}B$, and it can be attained only if $Q_i(S_O) = a_iB$ for each $i \in D$. Therefore, each block in S_O is one of the schedules shown in Fig. 3 □

Using Lemma 2.5, we can prove that MFTRTP(2) is NP-complete.

Theorem 2.6. MFTRTP(2) *is* NP-*complete.*

Proof. MFTRTP(2) is clearly in NP. To complete the proof, we reduce PARTITION to MFTRTP(2) as given in the beginning of the section, and choose ω to be $(3z^2+7z+5)B^2 + \frac{3}{2}B$. It is clear that the reduction can be done in polynomial time. Suppose the given instance of the PARTITION problem has a solution. Let A_1 and A_2 be a solution to the PARTITION problem. We can construct a schedule S with $f(S, W_0) = 2zB^2 + \frac{1}{2}B^2$ and $f(S, W_{z+1}) = f(S, W_{z+2}) = 4zB^2 + \frac{1}{2}B^2$ by executing V_i as

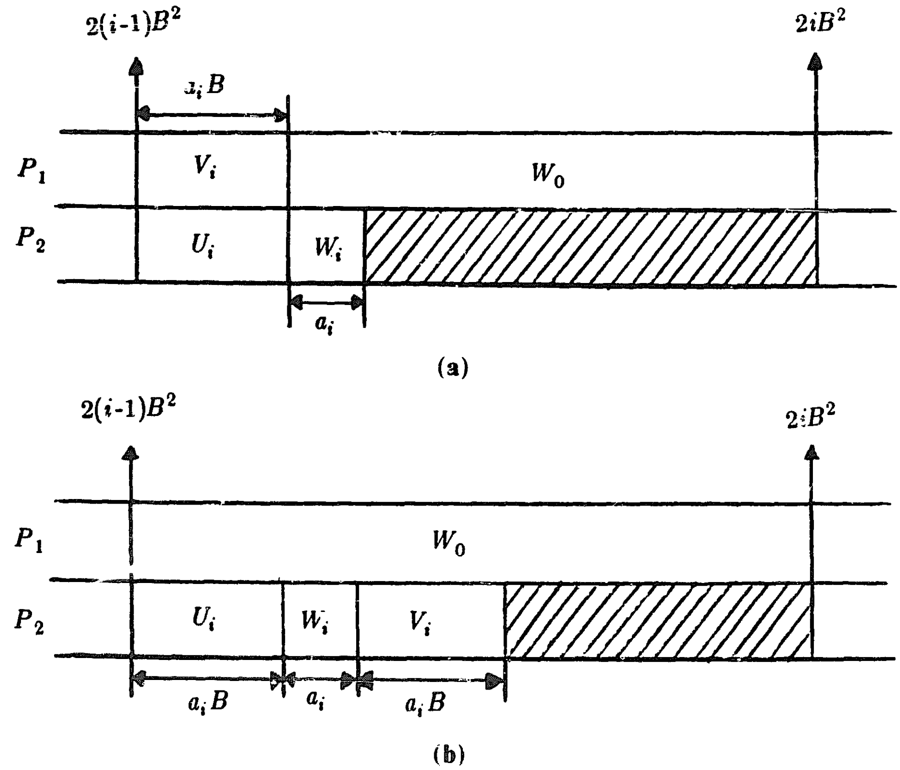

Fig. 3. $\mathrm{BLK}_i(S_O)$ when $\mathrm{MFT}(S_O)$ attains the lower bound. (a) V_i is not delayed; (b) V_i is delayed.

in Fig. 3a if $a_i \in A_1$; otherwise, we execute V_i as in Fig. 3b. It is easy to verify that the constructed schedule has mean flow time ω. Conversely, if the constructed instance of MFTRTP(2) has an optimal schedule S_O such that $\mathrm{MFT}(S_O) \leq \omega$, then the instance of the PARTITION problem must have a solution by Lemma 2.5. □

Corollary 2.7. MFTRTP(m) *is* NP-*complete for each* $m \geq 2$.

Proof. For $m \geq 3$, we simply add $m-2$ copies of U_i and W_i for each $1 \leq i \leq z$, and $m-2$ copies of W_{z+1} in the reduction. □

3. Conclusions

In this paper we have shown that finding a minimum mean flow time schedule for a set of independent tasks with release times is NP-hard for each fixed $m \geq 2$. As noted in Section 1, it also implies that the problem is NP-hard when the tasks have a common deadline to meet. For future research, it will be interesting to determine the complexity of the case of identical release times, arbitrary deadlines, arbitrary execution times, and $m \geq 2$. This problem is posed as an open problem in the survey paper by Lawler [5].

References

[1] K.R. Baker, *Introduction to Sequencing and Scheduling* (Wiley, New York, 1974).

[2] J. Du and J.Y.-T. Leung, Minimizing mean flow time with release time and deadline constraints, Technical Report UTDCS-2-88, Computer Science Program, University of Texas at Dallas, Richardson, TX 75083, 1988.

[3] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, CA, 1979).

[4] L.A. Herrbach and J.Y.-T. Leung, Preemptive scheduling of equal-length jobs on two machines to minimize mean flow time, *Oper. Res.* to appear.

[5] E.L. Lawler, Recent results in the theory of machine scheduling, in: A. Backem, Grotschel and B. Korte, eds., *Mathematical Programming: The State of the Art* (Springer, Berlin, 1982).

[6] E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Recent development in deterministic sequencing and scheduling: a survey, in: M.A.H. Dempster et al., *Deterministic and Stochastic Scheduling* (D. Reidel, Dordrecht, 1982) 35–73.

[7] J.K. Lenstra, *Sequencing by Enumerative Methods*, Mathematical Centre Tracts **69** (Mathematisch Centrum, Amsterdam, 1977).

[8] W.E. Smith, Various optimizers for single-stage production, *Naval Res. Logist. Quart.* 3 (1956) 59–66.