ISSN 0005-1179, Automation and Remote Control, 2010, Vol. 71, No. 10, pp. 2085–2092. © Pleiades Publishing, Ltd., 2010.
Original Russian Text © A.A. Lazarev, A.G. Kvaratskhelia, 2010, published in Avtomatika i Telemekhanika, 2010, No. 10, pp. 80–89.

Properties of Optimal Schedules for the Minimization Total Weighted Completion Time in Preemptive Equal-length Job with Release Dates Scheduling Problem on a Single Machine[1]

A. A. Lazarev and A. G. Kvaratskhelia

Trapeznikov Institute of Control Sciences, Russian Academy of Sciences, Moscow, Russia

Received January 12, 2010

Abstract—In this paper, we consider the minimizing total weighted completion time in preemptive equal-length job with release dates scheduling problem on a single machine. This problem is known to be open. Here, we give some properties of optimal schedules for the problem and its special cases.

DOI: 10.1134/S0005117910100073

1. INTRODUCTION

The *minimizing total weighted completion time in preemptive equal-length job with release dates* scheduling problem on a single machine is formulated as follows.

There are n jobs that have to be processed on a single machine with preemptions. Let $N = \{1, \dots, n\}$ be a set of jobs. For each job $j \in N$ the following parameters are given: release date $r_j \geqslant 0$, identical processing time $p_j = p > 0$, and weight $w_j \geqslant 0$. All parameters are integer values. Without loss of generality, we suppose that $w_1 \geqslant \dots \geqslant w_n$ and $\min\limits_{j \in N} r_j = 0$.

The schedule $\mathbb{S}$ sets the order in which jobs are executed on the machine. Basically, we consider the schedule as a piecewise constant left-continuous function $\mathbb{S}(t)$ that is acting from $\mathbb{R}$ to $\{0, \dots, n\}$, i.e., $\mathbb{S} : \mathbb{R} \to N \bigcup \{0\}$. If $\mathbb{S}(t)$ equals $j \in N$ then we say the job j is executed on the machine. Otherwise (when $\mathbb{S}(t) = 0$), we say the machine is idle. As all parameters of jobs are integer values, at the rest of the paper we consider only integer time points t. A part of a job with length 1 we call as *unit*.

Given a schedule $\mathbb{S}$, each job j has the *start time* S_j and the *completion time* C_j when execution of this job is finished. The completion time of job j is defined as the largest integer t such that $\mathbb{S}(t) = j$, the start time is defined as the smallest integer t such that $\mathbb{S}(t+1) = j$.

Let $F(\mathbb{S})$ denote the value of the *total weighted completion time* $\sum\limits_{j=1}^{n} w_j C_j$ for schedule $\mathbb{S}$. The objective is to find an optimal schedule $\mathbb{S}^*$ that minimizes weighted completion time. In the notation of Graham et al. [1] the problem is denoted as $1 \mid r_j, p_j = p, pmnt \mid \sum w_i C_j$.

There exists a polynomial time algorithm for non-preemptive case of the problem [2] with complexity $O(n^7)$ operations.

[1] This work was supported by the program of Presidium of the Russian Academy of Sciences no. 29 "The Mathematical Theory of Control."

2. PROPERTIES OF OPTIMAL SCHEDULES

In this section we give some properties of optimal schedules for the general case of the problem.

Theorem 1. *For every optimal schedule, if $S_i < S_j$ then either $C_i \leqslant S_j$, or $C_i > C_j$ for all $i, j \in N$.*

Proof. Let's consider an optimal schedule $\mathbb{S}$ where for two jobs i and j we have $S_i < S_j$, $C_i > S_j$, and $C_i < C_j$ (see Fig. 1).

Here, we can swap the first unit of job j with the last unit of job i and improve $\mathbb{S}$ at least by value w_i. ■

Theorem 1 states that in every optimal schedule units of two arbitrary jobs cannot be mixed in the time-line, i.e., there are two major possibilities of relative execution for two jobs:

- one job is processed completely before the second job;
- one job is "embedded into" the second jobs such that the execution period of the embedded job does not contain units of the second job.

This feature will be later discussed in Section 4.

If job i is executed completely before job j, we use notation $(i \to j)$. If job j is embedded into job i, we use notation $(i \rightsquigarrow j \rightsquigarrow i)$.

Theorem 2. *For every optimal schedule, if $(i \rightsquigarrow j \rightsquigarrow i)$ then $w_i \leqslant w_j$.*

Proof. Let's consider an optimal schedule $\mathbb{S}$ where for two jobs i and j, such that $(i \rightsquigarrow j \rightsquigarrow i)$, we have $w_i > w_j$. Let m be the count of units of job i that are processed after C_j, $m < n$ (see Fig. 2).

Since j is embedded into i, we can always swap m last units of job i with m last units of job j and improve $\mathbb{S}$ by value $(w_j - w_i)(C_i - C_j)$. ■

Theorem 3. *All parts of jobs that are processed after* $\max_i r_i$ *in any optimal schedule are sequenced in Smith's order* [3] *(i.e., in order of non-decreasing values p'_j/w_j where p'_j is the length of part of job j).*

Proof. All jobs are available after $\max_i r_i$. This implies the following:

- all parts of jobs are processed after $\max_i r_i$ without preemptions;
- two joint parts do not satisfy Smith's order then we can always interchange them and improve the schedule. ■

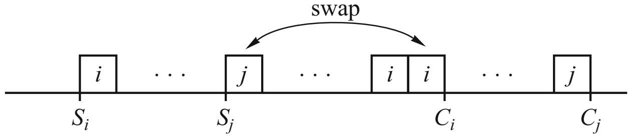

Fig. 1. Improve the schedule $\mathbb{S}$ (Theorem 1).

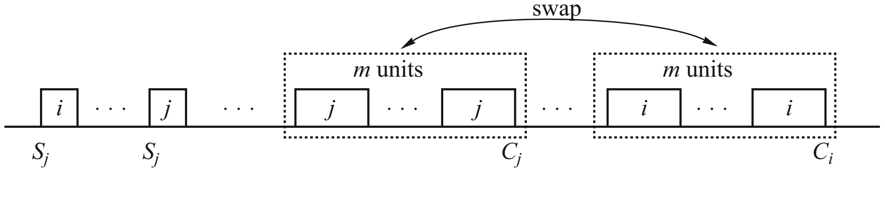

Fig. 2. Improve the schedule $\mathbb{S}$ (Theorem 2).

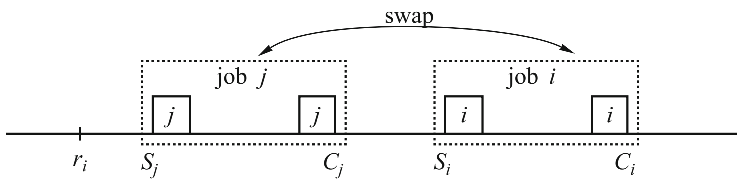

Fig. 3. Improve the schedule in Case 1 (Theorem 4).

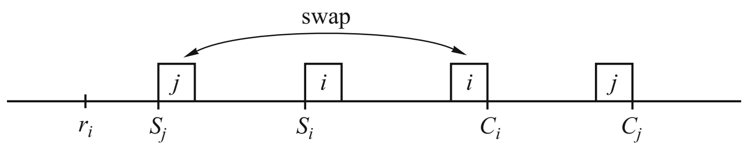

Fig. 4. Improve the schedule in Case 2 (Theorem 4).

Theorem 4. *For every optimal schedule, if $r_i \leqslant S_j$ and $w_i > w_j$ then $(i \to j)$ (i.e., $S_j \geqslant C_i$, job j is started only after i is completed).*

Proof. Let's consider an optimal schedule where $S_j < C_i$. By Theorem 2, this happens in two cases (the case $(i \rightsquigarrow j \rightsquigarrow i)$ is prohibited because $w_i > w_j$):

Case 1 $(j \to i)$, i.e., job i is started after j is finished.
Case 2 $(j \rightsquigarrow i \rightsquigarrow j)$, i.e., job i is embedded into job j.

Both jobs i and j are available at time S_j. Therefore, in Case 1 we can swap all items of these jobs and improve the schedule by value $(w_i - w_j)(C_i - C_j)$ (see Fig. 3). In Case 2 we can swap the last item of job i with the first item of job j and improve the schedule at least by value w_i (see Fig. 4). ∎

Corollary 1. *For every optimal schedule, if $r_i \leqslant r_j$ and $w_i > w_j$ then $(i \to j)$.*

Proof. Inequality $r_i \leqslant r_j$ implies $r_i \leqslant S_j$. ∎

Corollary 2. *There exists an optimal schedule where the conditions of Theorem 4 and Corollary 1 with replacement $w_i > w_j$ by $w_i \geqslant w_j$ hold.*

Proof. In Theorem 4 and Corollary 1 we strictly improve the schedule because $w_i > w_j$. If $w_i = w_j$ then our modifications of the schedule do not increase its cost value. ∎

Theorem 5. *In every optimal schedule jobs can be interrupted only at time points from the set $\{r_2, r_3, \ldots, r_n\}$. Also, if some job i is interrupted at r_k then $S_k = r_k$.*

Prior to the proof of Theorem 5 we give three lemmas.

Let $\mathbb{I}_j$ denote a set of time points when job j is interrupted in a schedule $\mathbb{S}$, i.e., for each $t \in \mathbb{I}_j$ we have $\mathbb{S}(t) = j$, $\mathbb{S}(t+1) \neq j$, and $t \neq C_j$.

Denote by $\mathbb{I} = \bigcup\limits_{j \in N} \mathbb{I}_j$ and $\mathbb{T}_k$ the interval $(r_k, r_{k+1}]$, $k = 1, \ldots, n$, where $r_{n+1} = np$.

Lemma 1. *For every optimal schedule, each job is interrupted no more than once on $\mathbb{T}_k$, $\forall k$.*

Proof. Consider an optimal schedule $\mathbb{S}$ where for some k job i is interrupted at least twice on T_k (see Fig. 5).

Between interruption points of i there exists some job j that is finished here. Otherwise, by Theorem 1 the schedule is not optimal.

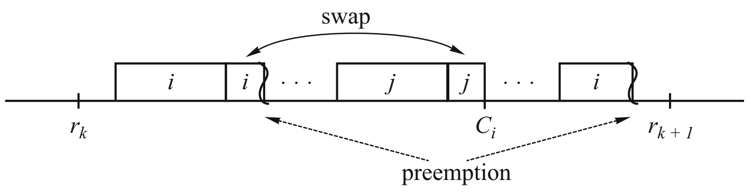

Fig. 5. Improve the schedule $\mathbb{S}$ (Lemma 1).

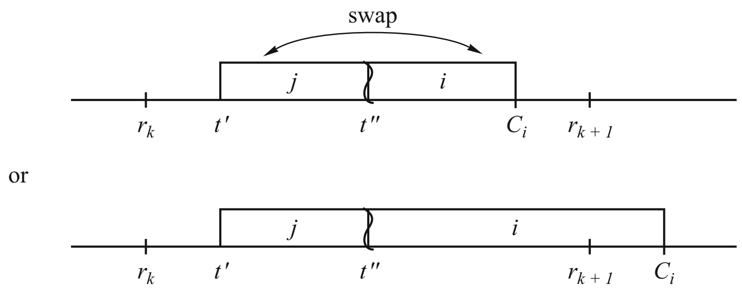

Fig. 6. Arrangement of units of jobs i and j at the schedule $\mathbb{S}$ in the assumption of the Lemma 1.

We can swap the last unit of the first part of job i with the last unit of job j (as shown on Fig. 5) and strongly improve the schedule by value w_j. ■

Lemma 2. *In each optimal schedule all interrupted jobs are processed at the end of* $\mathbb{T}_k$*,* $\forall k < n$*.*

Proof. Consider an optimal schedule $\mathbb{S}$ where for some $k \in N$ there exist two jobs (see Fig. 6):

(1) job j that is interrupted on $\mathbb{T}_k$;

(2) job i starts immediately after j was interrupted and completes either on $\mathbb{T}_k$, or somewhere afterward.

The point is that both jobs are available on $\mathbb{T}_k$ and, from Theorem 1, job i is finished before job j. Therefore, we can swap units of these jobs in such way that interrupted unit of j will be processed after i is completed. Moving units of job j at right does not change the cost function of the schedule, whereas moving units of job i at left strongly improves the schedule. ■

Lemma 3. *For each optimal schedule, at most one job can be interrupted on* $\mathbb{T}_k$*.*

Proof. Consider an optimal schedule where for some k we have two interrupted jobs i and j on T_k (see Fig. 7). By Lemma 2, these jobs are executed at the end of T_k.

In this case, we can swap units of jobs as shown on Fig. 7 and strongly improve the schedule by value w_j. ■

Proof of Theorem 5. The first part of the theorem directly follows from Lemmas 1–3. Now we show that if some job i is interrupted at r_k then $S_k = r_k$.

Consider an optimal schedule $\mathbb{S}$ where job i is interrupted at r_k, but then job j is started (not k), $r_j < r_k$. By Theorem 1 we have $C_j < C_i$. Therefore, we can swap the last unit of job j with any unit from preempted part of job i and strongly improve the schedule by value w_j (see Fig. 8). ■

3. PERMUTATION SCHEDULES

As we mentioned in Introduction, the general way to define a schedule is to use a function acting from $\mathbb{R}$ to $\{0, 1, \ldots, n\}$. For a variety of scheduling problems (typically, without preemptions) one

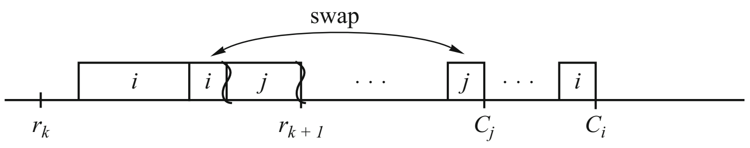

Fig. 7. Improve the schedule $\mathbb{S}$ (Lemma 3).

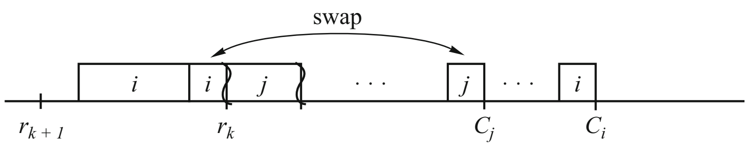

Fig. 8. Improve the schedule $\mathbb{S}$ (Theorem 5).

can consider a schedule as a permutation of job's indices. In this section we show that in spite of preemptions we can find an optimal solution of the problem in permutation schedules.

By π we denote a permutation schedule, i.e., $\pi = (j_1, j_2, \ldots, j_n)$. We say that job's order $(j_1, j_2, \ldots, j_n)$ means $C_{j_1} < C_{j_2} < \cdots < C_{j_n}$.

Now we give the procedure that constructs a time schedule $\mathbb{S}$ from the permutation schedule $\pi = (j_1, j_2, \ldots, j_n)$.

Procedure 1 (*construct time schedule from permutation*)

```
1: job $j_1$ starts at $r_{j_1}$ and ends at $r_{j_1} + p$ (i.e., $S_{j_1} := r_{j_1}$, $C_{j_1} := r_{j_1} + p$);
2: mark interval $[r_{j_1}, r_{j_1} + p)$ as busy;
3: for $k = 2, 3, \ldots, n$ do
4:     if $r_{j_k}$ is marked then
5:         let $s$ be the nearest value that is unmarked and greater than $r_{j_k}$;
6:     else
7:         $s := r_{j_k}$;
8:     end if
9:     calculate the width of unmarked intervals between $s$ and $C_{j_{k-1}} + 1$. Let this value be $d$
       ($d$ can also be negative);
10:    if $d \leqslant p$ then
11:        job $j_k$ starts at $s$ and fills unmarked intervals to the right until finished;
12:    else
13:        job $j_k$ ends at $C_{j_{k-1}} + 1$ and fills unmarked intervals to the left;
14:    end of
15: end for
```

On each step of the Procedure 1 we need to perform the following operations:

- realize whether r_{j_k} is in unmarked interval or not. This can be done by binary search among start and end times of jobs in $O(\log n)$ operations;
- calculate the width of unmarked intervals between r_{j_k} and $C_{j_{k-1}} + 1$. This can be done in $O(n)$ operations.

Therefore, each step is executed in $O(n)$ operations. Since Procedure 1 has n steps, the overall complexity of the procedure is $O(n^2)$.

By notation $\mathbb{S} = \text{Proc1}(\pi)$ we say that the time-schedule $\mathbb{S}$ is constructed from the permutation π using Procedure 1.

Lemma 4. *If the permutation π conforms to some optimal time schedule then Line 13 of Procedure 1 is never executed.*

Proof. Indeed, Line 13 means that between the possible start time s of the current job j_k and the minimal completion time $C_{j_{k-1}} + 1$ there are:

(1) either units of job j_k and units of some job i (job i is completed after j_k);
(2) or units of job j_k and empty time slots;

In Case 1, we can always exchange the last unit of j_k with the mentioned unit of i and strongly improve the schedule by value w_{j_k}.

In Case 2, we can move the last unit of j_k to the empty time slot (job j_k is available at this time) and also strongly improve the schedule by value w_{j_k}. ■

Now, let's consider an optimal schedule $\mathbb{S}$ with completion times $C_1, \ldots, C_n$ and its respective permutation $\pi = (j_1, j_2, \ldots, j_n)$. Let apply Procedure 1 to this permutation and obtain schedule $\mathbb{S}'$ with completion times $C'_1, \ldots, C'_n$.

Theorem 6. *For each optimal schedule $\mathbb{S}$ we have $C_j = C'_j$, $j \in N$.*

Proof. In the schedule $\mathbb{S}'$, the first job j_1 is completed at the least possible time $C'_{j_1} = r_{j_1} + p$. Therefore, $C_{j_1} \geqslant C'_{j_1}$.

If we have $C_{j_1} > C'_{j_1}$ then for some $t \in [r_{j_1}, C_{j_1})$ we have:

(1) either $\mathbb{S}(t) = i \neq j_1$ (i.e., there exists a unit of some job $i \neq j_1$ that is processed after r_{j_1} and before C_{j_1}),
(2) or $\mathbb{S}(t) = 0$ (i.e., the machine is idle at time t).

In Case 1, we can swap the last unit of job j_1 and the unit of job i. In Case 2, we can put the last unit of job j_1 at time slot t. In both cases, we strongly improve the schedule $\mathbb{S}$ by value w_{j_1}. Therefore, $C_{j_1} = C'_{j_1}$.

Now suppose that for $i < k$ the equalities $C_{j_i} = C'_{j_i}$ hold. We show $C_{j_k} = C'_{j_k}$. Note that job j_k cannot be completed earlier than C'_{j_k} in $\mathbb{S}$ because either the restriction $C_{j_k} \geqslant r_{j_k} + p$, or $C_{j_k} > C_{j_{k-1}}$ will be violated.

If $C_{j_k} > C'_{j_k}$ then for some $t \in [r_{j_k}, C_{j_k})$:

(1) either $\mathbb{S}(t) = i \neq j_1$ (i.e., there exists a unit of some job $i \neq j_k$ that is processed after r_{j_k} and before C_{j_k}),
(2) or $\mathbb{S}(t) = 0$ (i.e., the machine is idle at time t).

In Case 1, we can swap the last unit of job j_k and the unit of job i. In Case 2, we can put the last unit of job j_k at time slot t. In both cases, we strongly improve the schedule $\mathbb{S}$ by value w_{j_k}. Therefore, $C_{j_k} = C'_{j_k}$. ■

Theorem 6 states that we can find an optimal solution among permutation schedules. We use this property in our algorithm.

Theorem 7. *In every optimal schedule, each part of each job starts and ends at time points from set $T = \{r_j + lp, j = \overline{1,n}, l = \overline{0,n}\}$.*

Proof. Consider an optimal schedule $\mathbb{S}$. From Lemma 4 we have that Line 13 of Procedure 1 is never executed for this schedule. Other lines of the procedure insure that the completion times of jobs for the optimal schedule $\mathbb{S}$ will be from set T.

The start time of every job j is either its release time r_j, or the completion time of the previous job in $\mathbb{S}$. Both these time points are in set T. ■

Note, that the number of elements in the set T is $O(n^2)$.

Theorem 8. *Let π be an optimal permutation and $\mathbb{S} = Proc1(\pi)$. Then for each t and j such that $\mathbb{S}(t) = 0$ and $S_j \geqslant t$ we have $r_j \geqslant t$.*

Proof. Consider an optimal schedule $\mathbb{S}$ where $r_j < t$ for t and j such that $\mathbb{S}(t) = 0$ and $S_j \geqslant t$. In this schedule, j is available at time t and we can put the last unit of j at this empty time slot. Such transformation strongly improves $\mathbb{S}$ by value w_j. ■

4. GROUPS IN OPTIMAL SCHEDULES

From Theorem 1 we have the following two major possibilities of relative execution for two jobs in any optimal schedule:

- one job is processed completely before the second job starts;
- one job is "embedded into" the second jobs so that the execution period of the embedded job does not contain units of the second job.

Now we give definition of *a group* in an optimal schedule.

Consider an optimal permutation $\pi = (j_1, j_2, \ldots, j_n)$. We say that jobs $j_i, j_{i+1}, \ldots, j_m$ organize *a group* if:

- execution of jobs in the group starts with the first unit of j_m and ends with the last unit of j_m;
- all jobs $j_1, \ldots, j_{i-1}$ are executed completely before jobs from the group;
- all jobs $j_{m+1}, \ldots, j_n$ are executed completely after jobs from the group.

To denote a group we use symbol G. The same symbol is used to denote the set of jobs that are executed in this group, i.e., $G = \{j_1, \ldots, j_m\}$.

The example of a group is shown on Fig. 9.

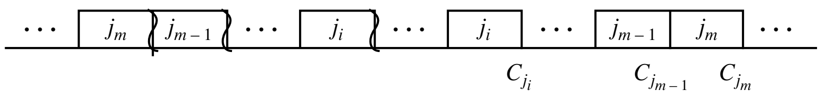

Fig. 9. The example of a group for jobs $\{j_1, \ldots, j_m\}$.

Theorem 9. *For every optimal schedule, if jobs $\{j_1, \ldots, j_m\}$ organize a group then w_{j_m} is the minimum weight among jobs from the group.*

Proof. The proof directly follows from Theorem 2. ■

5. SPECIAL CASE OF THE PROBLEM

In this section we study the special case of the problem $1 \mid r_j, p_j = p, pmnt \mid \sum w_j c_j$, when $p = 2$, $r_j = j - 1$, $j = 1, \ldots, n$, and

$$w_1 \leqslant w_2 \leqslant \ldots \leqslant w_n. \tag{1}$$

Theorem 10. *There exists an optimal schedule where for every job i in $\mathbb{S}$ we have either $S_j = r_i$, or $S_j > r_n$.*

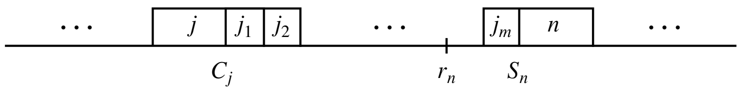

Fig. 10. The schedule in Case 1 (Theorem 11).

Proof. Let's consider an optimal schedule where $S_j > r_j$ and $S_j \leqslant r_n$. Therefore, $S_j = r_i$ for some job i. Since (1), we have $w_i \geqslant w_j$. By Theorem 4, we can always modify the schedule (as shown in the proof of Theorem 4) without increasing its cost value and move job j after i. By series of such modifications, we put job j after r_n. ■

Theorem 11. *There exists an optimal schedule where either $S_n = r_n$, or $S_n = r_n + 1$ holds.*

Proof. Let's consider an optimal schedule where $S_n > r_n + 1$. Let j be the nearest job that is processed without preemptions and $S_j < S_n$. Such job j always exists in each optimal schedule, otherwise either $S_n > r_n$, or Theorem 1 is failed.

Case 1. $C_j < S_n$, i.e., between C_j and S_n units of jobs $j_1, \ldots, j_m$ are processed (see Fig. 10).

All these units are the last units of jobs, otherwise the schedule is not optimal by Theorem 1. So, each such unit can be interchanged with its neighbor. This implies the following Smith's order for jobs j, $j_1, \ldots, j_m$, n:

$$\frac{2}{w_j} \leqslant \frac{1}{w_{j_1}} \leqslant \ldots \leqslant \frac{1}{w_{j_m}} \leqslant \frac{2}{w_n}.$$

If one of these inequalities is strong then we have $w_j > w_n$. This contradicts to 1. Otherwise, we can swap job n with the unit of job j_m, the unit of job j_{m-1} and so on until we (a) obtain the condition of the theorem or (b) put job m before job j. In case (b) we repeat reasoning of the theorem. These modifications do not increase the cost value of the schedule.

Case 2. $C_j < S_n$. Since $w_j \leqslant w_n$ then we can swap these jobs without increasing the cost value of the schedule. Next, we repeat reasoning of the theorem.

Modifications of the schedule in Cases 1 and 2 cannot be performed if conditions of the theorem are hold. ■

REFERENCES

1. Graham, R.L., Lawler, E.L., Lenstra, J.K., et. al., Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey, *Ann. Discrete Math.*, 1979, no. 5, pp. 287–326.
2. Baptiste, Ph., Scheduling Equal-Length Jobs on Identical Parallel Machines, *Discrete Appl. Math.*, 2000, no. 103, pp. 21–32.
3. Smith, W.E., Various Optimizers for Single-stage Production, *Naval Res. Logistics Quart.*, 1956, no. 3, pp. 59–66.

This paper was recommended for publication by P.Yu. Chebotarev, a member of the Editorial Board