Discrete Optimization 6 (2009) 446–460

Contents lists available at ScienceDirect

Discrete Optimization

journal homepage: www.elsevier.com/locate/disopt

Preemptive scheduling of equal length jobs with release dates on two uniform parallel machines

Irina N. Lushchakova [*]

Belarusian State University of Informatics and Radioelectronics, 6, P. Brovka street, Minsk 220013, Belarus

ARTICLE INFO

Article history:
Received 30 June 2005
Received in revised form 16 April 2009
Accepted 31 May 2009
Available online 27 June 2009

Keywords:
Scheduling
Uniform machines
Equal length jobs
Polynomial algorithm

ABSTRACT

We consider a problem of scheduling n jobs on two uniform parallel machines. For each job we are given its release date when the job becomes available for processing. All jobs have equal processing requirements. Preemptions are allowed. The objective is to find a schedule minimizing total completion time. We suggest an $O(n^3)$ algorithm to solve this problem.

1. Introduction

We consider the following scheduling problem.

There are $M = 2$ uniform parallel machines and a set $N = \{1, 2, \dots, n\}$ of jobs. For each job $i \in N$ we are given its release date $r_i \geq 0$ and the processing requirement p_i. We suppose that all jobs have equal processing requirements, i.e. $p_i = p$ for all i. Each job i has to be processed on any of two machines. Machine L, $1 \leq L \leq 2$, processes any job with the same speed v_L. This means that the processing time of any job on machine L is equal to p/v_L. Preemptions are allowed. After interruption of the processing of job i, it is possible either to resume its processing on the same machine later on or to process job i on the other machine. Each machine can process at most one job at a time and each job can be processed on at most one machine at a time.

For a feasible schedule s, let $C_i(s)$ be the completion time of job i. The objective is to find a schedule s^* minimizing total completion time $\sum_{i \in N} C_i(s)$.

Following the notation system introduced by Graham et al. [1], we denote the described problem by $Q2|r_i, p_i = p, pmtn|\sum C_i$.

This problem is indicated by Brucker and Knust [2] as a minimal open one. We suggest an $O(n^3)$ algorithm to solve it.

It should be mentioned, that Herrbach and Leung [3] solved the $P2|r_i, p_i = p, pmtn|\sum C_i$ problem with two identical parallel machines (the case $v_L = v$, $1 \leq L \leq 2$) in $O(n \log n)$ time. Recently Baptiste et al. [4] showed that the problem $P|r_i, p_i = p, pmtn|\sum C_i$ with an arbitrary number of parallel identical machines can be solved in polynomial time using linear programming.

On the other hand, Du, Leung and Young [5] proved that the $P2|r_i, pmtn|\sum C_i$ problem is NP-hard. Taking into account the elementary reductions for the objective functions [2], we conclude, that the $Q2|r_i, pmtn|\sum C_i$ problem with arbitrary processing requirements is also NP-hard. However, when all jobs are available simultaneously, the $Q|pmtn|\sum C_i$ problem with the variable number M of machines can be solved in $O(n \log n + Mn)$ time [6,1,7].

[*] Fax: +375 172 932333.
E-mail address: IrinaLushchakova@yandex.ru.

1572-5286/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.disopt.2009.05.005

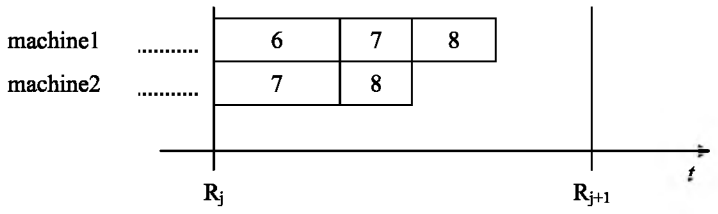

Fig. 1. Case 1. $N_j = \{6, 7, 8\}$.

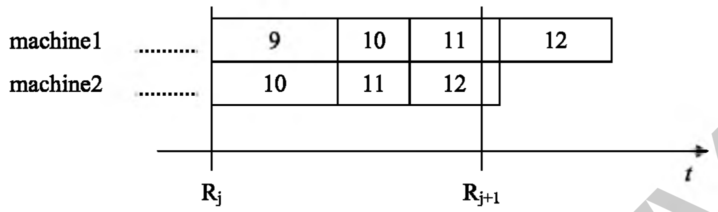

Fig. 2. Case 2. $N_j = \{9, 10, 11, 12\}$.

Notice that the problem $Q|r_i, p_i = p, pmtn|\sum C_i$ with an arbitrary number of parallel uniform machines remains an interesting open problem for further research.

The paper is organized as follows. In Section 2 we give the general Algorithm G for solving the $Q2|r_i, p_i = p, pmtn|\sum C_i$ problem. Algorithm G uses the transformation of partial schedules. This transformation process is described in Section 3 whereas its details are discussed in Sections 4 and 5. In Section 6 we prove that Algorithm G constructs an optimal schedule for the problem under consideration.

2. The general algorithm

Suppose that $v_1 \geq v_2$. Define $q = v_1/v_2$. Without loss of generality we assume that $v_2 = 1$. Therefore, $v_1 = q \geq 1$.

We shall use the variant of the Shortest Processing Time (SPT) rule for the preemptive scheduling of uniform machines [6,1,7] which in the case of two uniform machines may be described in the following way.

Suppose that the processing requirements of jobs are arbitrary and all jobs are available simultaneously.

The variant of the SPT rule:

Order the jobs according to the nondecreasing order of their processing requirements.

Schedule job 1 on machine 1. Having scheduled jobs $1, 2, \ldots, i$, schedule job $i+1$ on machine 2 until machine 1 becomes available, then interrupt the processing of job $i+1$ on machine 2 and resume its processing on machine 1, thereby completing job $i+1$ as soon as possible. □

The above variant of the SPT rule constructs the schedule minimizing total completion time [6,1,7]. We emphasize that the SPT rule is the optimal strategy when all jobs are available simultaneously. However, in this paper we investigate the situation when the release dates of jobs are not the same.

We shall call our general algorithm Algorithm G.

Suppose that the set N of jobs is ordered according to nondecreasing order of their release dates. Let there be z distinct release dates $R_1 < R_2 < \cdots < R_z$. Set $R_{z+1} = \infty$. For each j, $1 \leq j \leq z$, we define block as the time interval $(R_j, R_{j+1}]$.

Algorithm G generates the schedule block by block in increasing order of index j. We shall denote s_j the partial schedule constructed for the time interval $(R_1, R_{j+1}]$, $1 \leq j \leq z$.

Suppose that we are going to schedule jobs in the time interval $(R_j, R_{j+1}]$, $1 \leq j \leq z$. Let N_j denote the set of unscheduled jobs which are available at the time moment R_j. Starting from the time moment R_j and using the SPT rule, we schedule jobs of the set N_j. Let σ_j denote the schedule produced by the SPT rule for the set N_j of jobs. The following cases may occur.

1. All jobs of the set N_j are completed by the time moment R_{j+1}. (see Fig. 1). In this case σ_j is the desired schedule for the time interval $(R_j, R_{j+1}]$.
2. In the schedule σ_j there is no idle machine during the time interval $(R_j, R_{j+1}]$ (see Fig. 2). Then we interrupt the processing of the job(s) at the time moment R_{j+1} and pass all uncompleted jobs of the set N_j to the next block. The subschedule of σ_j in the time interval $(R_j, R_{j+1}]$ is the desired schedule for this block.
3. In the block $(R_j, R_{j+1}]$ there is an idle interval on machine 2 and there is a job from the set N_j, say job k, which is not completed by the time moment R_{j+1} (see Fig. 3). Then we transform the schedule σ_j into the new schedule $\tilde{\sigma}_j$ by using the procedure $SIGMA(\sigma_j)$ described below. If in the schedule $\tilde{\sigma}_j$ job k is not completed by the time moment R_{j+1}, we interrupt its processing at the time R_{j+1} and pass the uncompleted part of k to the next block. The subschedule of $\tilde{\sigma}_j$ in the time interval $(R_j, R_{j+1}]$ is the desired schedule for this block.

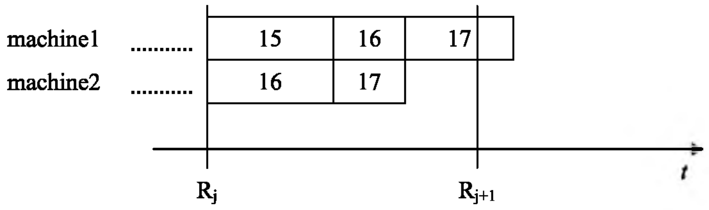

Fig. 3. Case 3. $N_j = \{15, 16, 17\}$.

Later on for convenience we shall say that a block is scheduled by Rule 1, 2 or 3, if during the construction of the desired schedule for this block we meet case 1, 2 or 3, respectively.

Below we show that the procedure $SIGMA(\sigma_j)$ can be done in $O(n^2)$ time. Thus, we construct the schedule for each block in no more than $O(n^2)$ time. Since there are $O(n)$ blocks, the running time of the algorithm is $O(n^3)$.

Now let us describe the transformation process of the schedule σ_j which is used when case 3 occurs.

3. The transformation of the schedule σ_j

Recall that in the schedule σ_j there is the only one job (namely, job k) which is not completed by the time moment R_{j+1}. Therefore, after the time moment R_{j+1} job k is processed on machine 1. Besides, in the schedule σ_j only machine 2 has an idle interval before the time moment R_{j+1}. Notice, that for any part of a job with a p' unit processing requirement its processing time on machine 1 is equal to p'/q time units, $q > 1$, while its processing time on machine 2 is equal to p' time units. So it is impossible to increase the amount of processing done in the block $(R_j, R_{j+1}]$ without increasing the partial objective function for the subset of jobs available before the time moment R_{j+1}. On the other hand, if we increase the amount of processing done in the block $(R_j, R_{j+1}]$, job k will complete earlier. As a result, we can decrease the sum of completion times for the subset of jobs available not earlier than at the time moment R_{j+1}. So the problem is to find the optimal value by which we should increase the amount of processing done in the block $(R_j, R_{j+1}]$. In other words, we should determine the optimal completion time of job k.

Consider the schedule σ_j. Suppose that in the schedule σ_j job k completes the processing at the time moment C_k. As it was mentioned, in the time interval $(R_{j+1}, C_k]$ job k is processed on machine 1. Set $\lambda = C_k - R_{j+1}$. In the block $(R_j, R_{j+1}]$ only machine 2 has an idle interval and let μ be its length. This means that the amount of processing done in the block $(R_j, R_{j+1}]$ can be increased by no more than μ units. As a result, job k can complete the processing on machine 1 earlier by no more than $\frac{\mu}{q}$ units. The length of the interval of processing for job k cannot be less than $\frac{p}{q}$ time units, that is $C_k - R_j \geq \frac{p}{q}$. It follows, that the completion time of job k can be diminished by no more than $\rho = C_k - (R_j + \frac{p}{q}) \geq 0$ time units. Summing up, if we diminish the completion time of job k by means of increasing the amount of processing done in the block $(R_j, R_{j+1}]$, we cannot complete job k earlier by more than $\delta = \min\{\mu/q, \lambda, \rho\}$ time units. Notice that in case 3 of Algorithm G we can have $\rho = 0$ (and therefore, $\delta = 0$) if and only if $N_j = \{k\}$. In this situation we do not transform the schedule σ_j.

Let $x \in [\lambda - \delta, \lambda]$, where $\delta > 0$. Suppose that we want to complete the processing of job k by the time moment $r_{k+1} + x$, where $r_{k+1} = R_{j+1}$. If $x \in [\lambda - \delta; \lambda)$, let us transform the schedule σ_j. For each job i, $i \in N_j$, we shall denote its completion time in the new schedule by $f_i(x)$. Besides, we define $f_i(\lambda) = C_i(\sigma_j)$, $i \in N_j$, and $f_0(\lambda) = 0$. If $j > 1$ and $x \in [\lambda - \delta, \lambda]$, we extend this notation, setting $f_i(x) = C_i(s_{j-1})$ for each job i completed in the time interval $(R_1; R_j]$. This means that during the transformation process we shall not change the partial schedule s_{j-1} constructed by Algorithm G for the time interval $(R_1; R_j]$.

Below in Section 4 we describe the procedure $TRANS(\sigma_j, x)$. This procedure transforms the schedule σ_j into the new schedule $\tilde{\sigma_j}$ that is the best one among all schedules for the set N_j such that job k completes the processing at the time moment $r_{k+1} + x$.

Let us introduce the function $F_k(x) = \sum_{i=1}^{k-1} f_i(x)$, where $x \in [\lambda - \delta; \lambda]$, and consider its increment $\Delta F_k(x) = F_k(x) - F_k(\lambda)$. Since the transformation of the schedule σ_j by means of increasing the amount of processing done in the block $(R_j; R_{j+1}]$ also increase the partial objective function for the subset of jobs $\{1, 2, \ldots, k-1\}$, we conclude that $\Delta F_k(x) > 0$ for $x \in [\lambda - \delta; \lambda)$. In Section 4 we show that the increment $\Delta F_k(x)$ is a linear function or a piecewise linear function with a unique breakpoint.

Now let us consider jobs available not earlier than the time moment R_{j+1}. As above, we suppose that job k completes the processing at the time moment $f_k(x) = r_{k+1} + x$, where $r_{k+1} = R_{j+1}$, $x \in [\lambda - \delta, \lambda]$. Schedule all unscheduled jobs using the strategy to complete each job as soon as possible.

Start the processing of job $k + 1$ on machine 2 at the time moment r_{k+1}. At the time moment $r_{k+1} + x$, when machine 1 completes the processing of job k, interrupt the processing of job $k + 1$ on machine 2 and resume its processing on machine 1 for $\frac{p-x}{q}$ time units. Thus, the completion time of job $k + 1$ will be the following:

$$f_{k+1}(x) = r_{k+1} + x + \frac{p - x}{q}.$$

Consider job i, where $k+2 \le i \le n$. At first suppose that at the time moment r_i machine 1 is available, i.e. $r_i \ge f_{i-1}(x)$. In this case job i is assigned to machine 1. Machine 1 completes the processing of job i at the time moment $f_i(x) = r_i + \frac{p}{q}$. Now suppose that at the time moment r_i machine 1 is busy, i.e. $r_i < f_{i-1}(x)$. In this case job i starts the processing on machine 2 at the time moment $\max\{f_{i-2}(x), r_i\}$ and is processed on this machine for $f_{i-1}(x) - \max\{f_{i-2}(x), r_i\}$ time units, until machine 1 becomes available. At the time moment $f_{i-1}(x)$ we interrupt the processing of job i on machine 2 and resume its processing on machine 1. Machine 1 processes job i for $\frac{p-(f_{i-1}(x)-\max\{f_{i-2}(x),r_i\})}{q}$ time units and completes its processing at the time moment $f_i(x) = f_{i-1}(x) + \frac{p-(f_{i-1}(x)-\max\{f_{i-2}(x),r_i\})}{q}$. Notice, that $f_{i-2}(x) < f_{i-1}(x)$ for $i = k+2, k+3, \dots, n$. Thus, for the both cases the completion time of job i can be expressed by the formula

$$f_i(x) = \max\{f_{i-1}(x), r_i\} + \frac{p - (\max\{f_{i-1}(x), r_i\} - \max\{f_{i-2}(x), r_i\})}{q},$$

where $k+2 \le i \le n$.

The above formulas for the completion times of the jobs can be written in the following form:

$$f_{k+1}(x) = \left(1 - \frac{1}{q}\right)x + r_{k+1} + \frac{p}{q},$$
$$f_i(x) = \left(1 - \frac{1}{q}\right)\max\{f_{i-1}(x), r_i\} + \frac{1}{q}\max\{f_{i-2}(x), r_i\} + \frac{p}{q}, \tag{1}$$

where $k+2 \le i \le n$.

Notice, that $f_k(x)$ and $f_{k+1}(x)$ are nondecreasing linear functions. Denote $\phi_i(x) = \max\{f_{i-1}(x), r_i\}$, $\psi_i(x) = \max\{f_{i-2}(x), r_i\}$, $k+2 \le i \le n$. It is clear that for any i, $k+2 \le i \le n$, the functions $\phi_i(x)$, $\psi_i(x)$ and, therefore, the function

$$f_i(x) = \left(1 - \frac{1}{q}\right)\phi_i(x) + \frac{1}{q}\psi_i(x) + \frac{p}{q} \tag{2}$$

are nondecreasing continuous piecewise linear convex functions. Each of the functions $\phi_i(x)$, $\psi_i(x)$, $f_i(x)$ can be described by using the list of its breakpoints and the corresponding list of linear functions.

Let us introduce the function $F_k^*(x) = \sum_{i=k}^{n} f_i(x)$, $x \in [\lambda - \delta, \lambda]$. Below in Section 5 we describe how to construct the function $F_k^*(x)$ in $O(n^2)$ time. There we also show that $F_k^*(x)$ is a nondecreasing piecewise linear function with no more than $O(n)$ breakpoints.

If we complete job k at the time moment $f_k(x) = r_{k+1} + x$, where $x \in [\lambda - \delta; \lambda)$, instead of the time moment $f_k(\lambda) = r_{k+1} + \lambda$, the function $F_k^*(x)$ decreases by the value $\Delta F_k^*(x) = F_k^*(\lambda) - F_k^*(x) > 0$. On the other hand, in this case the function $F_k(x) = \sum_{i=1}^{k-1} f_i(x)$, where $x \in [\lambda - \delta, \lambda)$, increases by the value $\Delta F_k(x) = F_k(x) - F_k(\lambda) > 0$.

Thus, the problem is to find the value x^0 maximizing the function $\Delta F_k^*(x) - \Delta F_k(x)$, $x \in [\lambda - \delta, \lambda]$. From the above consideration it follows that $\Delta F_k^*(x) - \Delta F_k(x)$ is a piecewise linear function. So this function can have the maximum value only in the breakpoints or in the endpoints of the interval $[\lambda - \delta, \lambda]$. Therefore, one should choose the maximal value among the values of the function $\Delta F_k^*(x) - \Delta F_k(x)$ in these points. Notice that the function $\Delta F_k^*(x) - \Delta F_k(x)$ has $O(n)$ breakpoints.

When the optimal value x^0 is found, we transform the schedule σ_j into the schedule $\tilde{\sigma}_j$.

The following procedure describes all these actions more formally.

```
SIGMA($\sigma_j$)
1. For the schedule $\sigma_j$ find the value $\delta = \min\{\frac{\mu}{q}, \lambda, \rho\}$.
2. If $\delta = 0$, set $\tilde{\sigma}_j := \sigma_j$ and go to step 7.
3. For the subset of jobs $\{1, 2, \dots, k-1\}$ construct the function $\Delta F_k(x) = F_k(x) - F_k(\lambda)$, $x \in [\lambda - \delta; \lambda]$ (see Section 4).
4. For the subset of jobs $\{k, k+1, \dots, n\}$ construct the function $\Delta F_k^*(x) = F_k^*(\lambda) - F_k^*(x)$, $x \in [\lambda - \delta; \lambda]$ (see Section 5).
5. Find the value $x^0$ maximizing the function $\Delta F_k^*(x) - \Delta F_k(x)$, $x \in [\lambda - \delta; \lambda]$.
6. TRANS($\sigma_j$, $x^0$) (see Section 4).
7. Stop.
```

Step 4 of the procedure *SIGMA*(σ_j) is the most time-consuming and requires $O(n^2)$ time (see Section 5). Recall that the function $\Delta F_k^*(x) - \Delta F_k(x)$ has $O(n)$ breakpoints. Therefore, Step 5 requires $O(n)$ time. Step 6 requires the constant time (see Section 4). Therefore, the procedure *SIGMA*(σ_j) can be done in $O(n^2)$ time.

4. The procedure *TRANS*(σ_j, x) and its justification

Let us describe the procedure *TRANS*(σ_j, x) that transforms the schedule σ_j into the schedule $\tilde{\sigma}_j$ such that job k completes the processing at the time moment $r_{k+1} + x$, $x \in [\lambda - \delta; \lambda]$. Three cases may occur, each of them being handled separately.

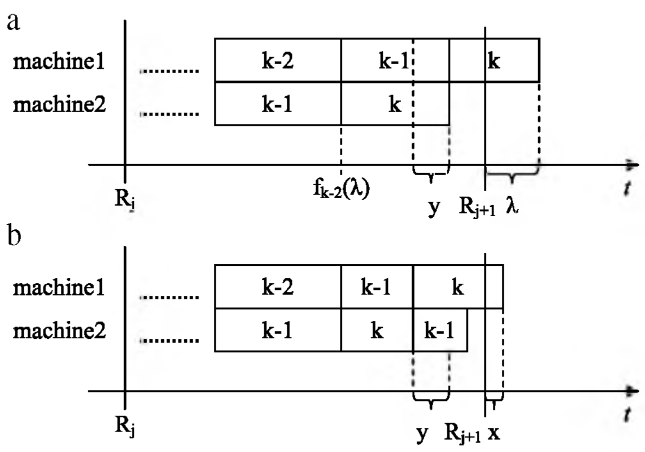

Fig. 4. The transformation of the schedule σ_j. Case 3a.

The motivation to each case and the justification of the correctness are done after the procedure.

TRANS(σ_j, x)

(a) If

$$\max\{r_k, f_{k-2}(\lambda)\} + \frac{p}{q} \le f_k(x) = r_{k+1} + x$$

then do;

a1. Find the value y from the equation

$$\lambda - x + \frac{y}{q} = y. \tag{3}$$

a2. Transform the schedule σ_j in the following way. At the time moment $f_{k-1}(\lambda) - y$ interrupt the processing of jobs $k - 1$ and k on machines 1 and 2, respectively, and resume their processing on the opposite machines. Denote the obtained schedule by $\tilde{\sigma}_j$ (see Fig. 4(a), (b)). Set $C_i(\tilde{\sigma}_j) = f_i(x)$, $i \in N_j$.

end;

(b) If

$$f_{k-2}(\lambda) + \frac{p - (f_{k-2}(\lambda) - \max\{r_k, f_{k-3}(\lambda)\})}{q} \le f_k(x) = r_{k+1} + x < f_{k-2}(\lambda) + \frac{p}{q}$$

then do;

b1. Find the value y from the equation:

$$f_{k-2}(\lambda) + \frac{p - y}{q} = r_{k+1} + x. \tag{4}$$

b2. Transform the schedule σ_j in the following way. At the time moment $f_{k-2}(\lambda) - y$ interrupt the processing of job $k - 1$ on machine 2 and start the processing of job k on this machine. At the time moment $f_{k-2}(\lambda)$ interrupt the processing of job k on machine 2 and resume its processing on machine 1. At the same moment resume the processing of job $k - 1$ on machine 2. Denote the obtained schedule by $\tilde{\sigma}_j$. (See Fig. 5(a) and (b)). Set $C_i(\tilde{\sigma}_j) = f_i(x)$, $i \in N_j$.

end;

(c) If

$$r_k + \frac{p}{q} \le f_k(x) = r_{k+1} + x < f_{k-2}(\lambda) + \frac{p - (f_{k-2}(\lambda) - r_k)}{q}$$

then do;

c1. Transform the schedule σ_j as it was done at Step b2, setting $y = f_{k-2}(\lambda) - r_k$. Denote the obtained schedule by $\hat{\sigma}_j$ (see Fig. 6(a) and (b)).

c2. Find the value x^* from the Eq. (4) with the fixed value $y = f_{k-2}(\lambda) - r_k$. In other words, find x^* from the equation:

$$r_{k+1} + x^* = f_{k-2}(\lambda) + \frac{p - (f_{k-2}(\lambda) - r_k)}{q}. \tag{5}$$

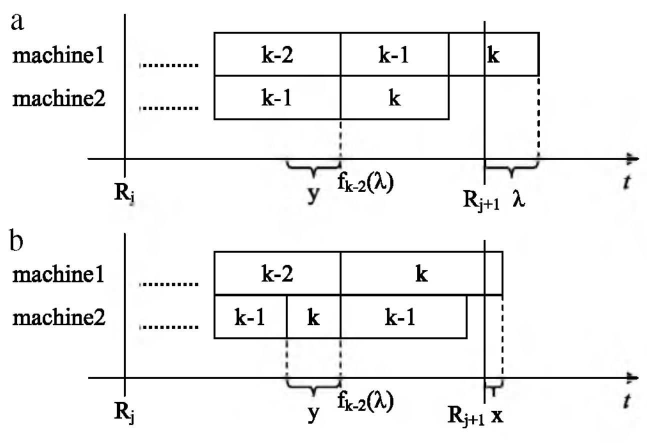

Fig. 5. The transformation of the schedule σ_j. Case 3b.

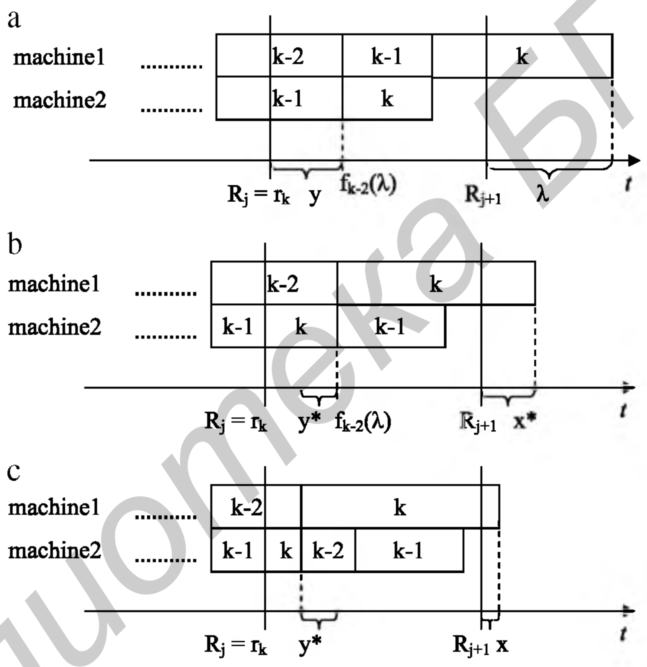

Fig. 6. The transformation of the schedule σ_j. Case 3c.

c3. Find the value y^* from the equation

$$x^* - x + \frac{y^*}{q} = y^*. \tag{6}$$

c4. Transform the schedule $\hat{\sigma}_j$ in the following way. At the time moment $f_{k-2}(\lambda) - y^*$ interrupt the processing of jobs $k-2$ and k on machines 1 and 2, respectively, and resume their processing on the opposite machines. After the completion of job $k-2$ on machine 2, resume the processing of job $k-1$ on this machine. Denote the obtained schedule by $\tilde{\sigma}_j$ (see Fig. 6(b) and (c)). Set $C_i(\tilde{\sigma}_j) = f_i(x)$, $i \in N_j$.
end;
Stop.

The condition of case (a) means that if we start the processing of job k at the time moment $\max\{r_k, f_{k-2}(\lambda)\}$, we can complete it by the moment $r_{k+1} + x$. It should be mentioned that $r_k \geq f_{k-2}(\lambda)$ iff $N_j = \{k-1, k\}$. Moreover, we claim that the following Lemma takes place.

Lemma 1. *If $N_j = \{k-1, k\}$, only case* (a) *may occur.*

Proof. Suppose that the inequality

$$r_k + \frac{p}{q} > f_k(x) = r_{k+1} + x \tag{7}$$

holds. By definition $\rho = f_k(\lambda) - r_k - \frac{p}{q}$. Therefore, we get $r_{k+1} < r_k + \frac{p}{q} - x = f_k(\lambda) - \rho - x \le f_k(\lambda) - \delta - x = r_{k+1} + \lambda - \delta - x \le r_{k+1}$, because $\delta \le \rho$ and $x \ge \lambda - \delta$. Thus, we conclude that for the set $N_j = \{k-1, k\}$ the inequality (7) cannot take place. □

The condition of case (b) means the following. If we start the processing of job k at the time moment $f_{k-2}(\lambda)$, we cannot complete it by the time moment $r_{k+1} + x$. However, if we start the processing of job k on machine 2 at the time moment $\max\{r_k, f_{k-3}(\lambda)\}$ and continue its processing on machine 1 from the time moment $f_{k-2}(\lambda)$, we can complete it by the moment $r_{k+1} + x$.

It should be mentioned that $r_k \ge f_{k-3}(\lambda)$ iff $N_j = \{k-2, k-1, k\}$. Furthermore, we claim that the following Lemma takes place.

Lemma 2. *If* $|N_j| > 3$, *only cases* (a) *or* (b) *may occur.*

Proof. Suppose that it is not the case. In other words, suppose that the inequality

$$f_{k-2}(\lambda) + \frac{p - (f_{k-2}(\lambda) - f_{k-3}(\lambda))}{q} > r_{k+1} + x \tag{8}$$

holds.

Notice that $f_{k-1}(\lambda) = f_{k-2}(\lambda) + (p - (f_{k-2}(\lambda) - f_{k-3}(\lambda)))/q$. Therefore, from (8) we obtain the inequality $f_{k-1}(\lambda) > r_{k+1} + x$ which is impossible. So (8) cannot take place if $|N_j| > 3$. □

From Lemmas 1 and 2 one can easy obtain the following Corollary.

Corollary 1. *Case* (c) *may occur only if* $N_j = \{k-2, k-1, k\}$.

The condition of case (c) means the following. If we start the processing of job k on machine 2 at the time moment r_k and continue its processing on machine 1 from the time moment $f_{k-2}(\lambda)$, we will not be able to complete it by the moment $r_{k+1} + x$. However, if job k starts the processing at the time moment r_k, it is possible to complete its processing by the moment $r_{k+1} + x$.

Consider case (a). Let us explain the Eq. (3) which is connected with the transformation of the schedule σ_j done at Step (a2) (see Fig. 4(a), (b)).

In the schedule σ_j during the interval $(f_{k-1}(\lambda) - y; f_{k-1}(\lambda)]$ machine 1 processed job $k-1$, and the length of this interval is equal to y time units. In the schedule $\tilde{\sigma}_j$ during the interval $(f_{k-1}(\lambda) - y; f_{k-1}(\lambda)]$ machine 1 processes job k. The Eq. (3) expresses the structure of this part of job k with qy unit processing requirement which is the following:

- a part of job k with $q(\lambda - x)$ unit processing requirement which was done in the schedule σ_j for $\lambda - x$ time units on machine 1 during the interval $(r_{k+1} + x, r_{k+1} + \lambda]$, and is also done for $\lambda - x$ time units on machine 1 in the schedule $\tilde{\sigma}_j$;
- a part of job k with y unit processing requirement which was done in the schedule σ_j for y time units on machine 2 during the interval $(f_{k-1}(\lambda) - y; f_{k-1}(\lambda)]$, and is done for $\frac{y}{q}$ time units on machine 1 in the schedule $\tilde{\sigma}_j$.

As a result of Step (a2) we have $f_{k-1}(x) = f_{k-1}(\lambda) - y + yq = f_{k-1}(\lambda) + q(\lambda - x)$ and $f_k(x) = f_k(\lambda) - y + \frac{y}{q} = f_k(\lambda) - (\lambda - x) = (r_{k+1} + \lambda) - (\lambda - x) = r_{k+1} + x$. The completion times of the other jobs from the set N_j do not change, i.e. $f_i(x) = f_i(\lambda)$ for all $i \in N_j \setminus \{k-1, k\}$. Notice, that in the schedule $\tilde{\sigma}_j$ we have $f_{k-1}(x) = f_{k-1}(\lambda) + q(\lambda - x) \le f_{k-1}(\lambda) + q\delta \le f_{k-1}(\lambda) + q * \frac{\mu}{q} = f_{k-1}(\lambda) + \mu = R_{j+1}$.

Now consider case (b). The Eq. (4) expresses the fact that in the schedule $\tilde{\sigma}_j$ obtained at Step (b2) job k completes the processing at the time moment $r_{k+1} + x$.

It is not difficult to see that in the schedule $\tilde{\sigma}_j$ constructed at Step (b2) machine 1 processes job k for additional $(f_{k-1}(\lambda) - f_{k-2}(\lambda) - y)/q$ time units instead of processing of job $k-1$ for $f_{k-1}(\lambda) - f_{k-2}(\lambda)$ time units. As a result we complete job k earlier by $\lambda - x$ time units. This fact can be expressed by the following equation:

$$f_{k-1}(\lambda) - f_{k-2}(\lambda) - \frac{f_{k-1}(\lambda) - f_{k-2}(\lambda) - y}{q} = \lambda - x. \tag{9}$$

From (9) we easy get

$$(q-1)(f_{k-1}(\lambda) - f_{k-2}(\lambda)) + y = q(\lambda - x). \tag{10}$$

Using (10), we obtain the expression for the completion time of job $k-1$ in the transformed schedule $\tilde{\sigma}_j$: $f_{k-1}(x) = f_{k-2}(\lambda) + y + q(f_{k-1}(\lambda) - f_{k-2}(\lambda)) = f_{k-2}(\lambda) + (f_{k-1}(\lambda) - f_{k-2}(\lambda)) + (q-1)(f_{k-1}(\lambda) - f_{k-2}(\lambda)) + y = f_{k-1}(\lambda) + q(\lambda - x)$. By the construction $f_k(x) = r_{k+1} + x$. The completion times of all other jobs from the set N_j do not change, i.e. $f_i(x) = f_i(\lambda)$, $i \in N_j \setminus \{k-1, k\}$. Besides, by the same reason as in case (a) we have $f_{k-1}(x) \le R_{j+1}$.

Finally, consider case (c).

The Eq. (5) expresses the following. In the schedule $\hat{\sigma}_j$ (see Step c1) job k starts the processing on machine 2 at the time moment r_k. At the time moment $f_{k-2}(\lambda)$ the processing of job k on machine 2 is interrupted and is resumed on machine 1. Job k completes the processing at the time moment $r_{k+1} + x^*$.

Notice, that $x^* > x$. Therefore, the transformation $\sigma_j \to \hat{\sigma}_j$ is not resultant. In the schedule $\hat{\sigma}_j$ we have $f_{k-1}(x^*) = f_{k-1}(\lambda) + q(\lambda - x^*), f_k(x^*) = r_{k+1} + x^*, f_{k-2}(x^*) = f_{k-2}(\lambda)$ (see case b).

The Eq. (6) is connected with the transformation $\hat{\sigma}_j \to \tilde{\sigma}_j$ done at Step c4 (see Fig. 6(b) and (c)).The meaning of the Eq. (6) can be explained analogously as it was done for the Eq. (3).

In the schedule $\tilde{\sigma}_j$ we have $f_k(x) = r_{k+1} + x$ by the construction. Further, taking into account (6), we get $f_{k-2}(x) = f_{k-2}(\lambda) + (q-1)y^* = f_{k-2}(\lambda) + q(x^* - x)$ and $f_{k-1}(x) = f_{k-1}(x^*) + (q-1)y^* = f_{k-1}(\lambda) + q(\lambda - x^*) + q(x^* - x) = f_{k-1}(\lambda) + q(\lambda - x)$.

Now let us prove the following theorem.

Theorem 1. *The schedule $\tilde{\sigma}_j$ constructed by the procedure TRANS(σ_j, x) is the best one among all schedules for the block $(R_j; R_{j+1}]$ provided that job k completes the processing at the time moment $r_{k+1} + x$.*

Proof. Let us consider the increment $\Delta F_k(x) = F_k(x) - F_k(\lambda)$ of the function $F_k(x) = \sum_{i=1}^{k-1} f_i(x)$, where $x \in [\lambda - \delta; \lambda]$.

One can see that $\Delta F_k(x) = q(\lambda - x)$ for cases (a) and (b).

Recall that it is impossible to decrease the completion time of job k by $\lambda - x$ time units without increasing the amount of processing done in the block $(R_j; R_{j+1}]$ at least by $q(\lambda - x)$ units. So we conclude, that for the cases (a) and (b) the schedule $\tilde{\sigma}_j$ has the minimal possible increment $\Delta F_k(x) = q(\lambda - x)$ of the function $F_k(x)$. Therefore, for the cases (a) and (b) we have constructed the best schedule among all schedules in which job k completes the processing by the time moment $r_{k+1} + x$.

Further, it is easy to see that for the case (c) the constructed schedule $\tilde{\sigma}_j$ has the increment $\Delta F_k(x) = q(\lambda - x) + q(x^* - x)$, where x^* is the solution of the Eq. (5).

Let us show that the theorem also takes place for the case (c). Recall that case (c) may occur only if $N_j = \{k-2, k-1, k\}$.

Denote the processing requirements of jobs $k-2$, $k-1$ and k in the schedule σ_j (see Fig. 6(a)) by p'_{k-2}, p'_{k-1} and p_k, respectively. Notice, that $p'_{k-2} < p'_{k-1} < p_k = p$ and $\frac{p'_{k-2}}{q} = f_{k-2}(\lambda) - r_k$.

To prove the theorem, we need to investigate the properties of the best schedule σ^* for jobs $k-2$, $k-1$ and k with the processing requirements p'_{k-2}, p'_{k-1} and p_k, provided that these jobs can start the processing no earlier than at the time moment r_k and the completion time of job k is equal to $r_{k+1} + x$.

Property 1. The schedule σ^* is a dense schedule.

This means that machines work continuously from the time moment r_k until each of them completes the processing. Notice that machine 1 has no idle periods in the interval $(r_k; r_{k+1} + x]$.

Property 2. In the schedule σ^* machine 2 completes the processing at the time moment $f_{k-1}(\lambda) + q(\lambda - x)$.

Indeed, since the completion time of job k is decreased by $\lambda - x$ time units, the amount of processing done in the block $(R_j; R_{j+1}]$ increases by $q(\lambda - x)$ units. This additional amount of processing can be done only on machine 2, because machine 1 has no idle periods during the interval $(R_j; R_{j+1}]$ in the schedule σ_j. Notice, that in the schedule σ_j machine 2 completes the processing at the time moment $f_{k-1}(\lambda)$. Therefore, in the schedule σ^* machine 2 should complete the processing at the time moment $f_{k-1}(\lambda) + q(\lambda - x)$.

Property 3. In the schedule σ^* the total length of the intervals in which jobs $k-2$ and $k-1$ are processed on machine 1 is less than $\frac{p'_{k-2}}{q}$.

Let us show that Property 3 takes place. If job k is processed continuously from the time moment r_k and the total length of the intervals of its processing on machine 2 is equal to $\frac{p'_{k-2}}{q} = f_{k-2}(\lambda) - r_k$, it cannot complete the processing earlier than at the time moment $f_{k-2}(\lambda) + \frac{p-(f_{k-2}(\lambda)-r_k)}{q}$. However, we have $r_{k+1} + x < f_{k-2}(\lambda) + \frac{p-(f_{k-2}(\lambda)-r_k)}{q}$. So we conclude that in the schedule σ^* the total length of the intervals of processing job k on machine 2 is less than $\frac{p'_{k-2}}{q}$. Moreover, in σ^* the total length of the intervals in which job k is not processed on machine 1 is also less than $\frac{p'_{k-2}}{q}$. It follows that the total length of the intervals in which jobs $k-2$ and $k-1$ can be processed on machine 1 is less than $\frac{p'_{k-2}}{q}$.

Property 4. In the schedule σ^* machine 1 at first processes jobs of the set $\{k-2; k-1\}$ and then processes job k.

The schedule σ^* has the minimal total completion time of jobs $k-2$ and $k-1$ among all schedules with the fixed completion time $r_{k+1} + x$ of job k. The speed of machine 1 is greater than the one of machine 2. So it is expedient to process jobs $\{k-2; k-1\}$ on machine 1 at the beginning of the interval $(r_k; r_{k+1} + x]$, because in this case it may be possible to complete these jobs earlier.

For further convenience, we denote the interval of processing of jobs $\{k-2; k-1\}$ on machine 1 in the schedule σ^* by τ.

Property 5. In the schedule σ^* the completion time of one job from the set $\{k-2; k-1\}$ is equal to $f_{k-1}(\lambda) + q(\lambda - x)$.

Let us show that Property 5 takes place. From Property 3 and Property 4 it follows that machine 1 completes the processing of jobs $\{k-2; k-1\}$ earlier than the time moment $r_k + \frac{p'_{k-2}}{q}$. We have $r_k + \frac{p'_{k-2}}{q} = f_{k-2}(\lambda) < f_{k-1}(\lambda) < f_{k-1}(\lambda) + q(\lambda - x)$. Therefore, at the time moment $f_{k-1}(\lambda) + q(\lambda - x)$ machine 1 processes job k. Since according to Property

2 in the schedule σ^* machine 2 completes the processing at the time moment $f_{k-1}(\lambda) + q(\lambda - x)$ and machine 1 is busy at this moment, we conclude that the last job processed by machine 2 is a job from the set $\{k-2; k-1\}$.

Property 6. In the schedule σ^* job k is processed throughout the interval $(r_k; r_{k+1} + x]$.

In other words, we need to show that machine 2 processes only job k in the interval τ. Assume the converse. Then there are some subintervals of τ during which both machines process jobs of the set $\{k-2; k-1\}$. Since machine 2 does not process job k in these subintervals, a greater amount of processing of job k will be done on machine 1. As a result, the length of the interval τ will be shorter. However, it is not reasonable.

Indeed, according to Property 5 the completion time of one job of the set $\{k-2; k-1\}$ is fixed. To complete the other job of the set $\{k-2; k-1\}$ earlier, we should use machine 1 for its processing as much as possible. From Property 3 we obtain that the length of the interval τ is less than $\frac{p'_{k-2}}{q} < \frac{p'_{k-1}}{q}$. So in the interval τ machine 1 is used only for the processing of the job of the set $\{k-2; k-1\}$ with nonfixed completion time, the length of the interval τ being as large as possible. This contradicts our assumption. Thus, in the interval τ machine 2 processes only job k. After the interval τ job k is processed on machine 1. Since the processing requirement and the completion time of job k are known, the length of the interval τ is uniquely determined.

Property 7. In the schedule σ^* jobs $k-2$ and $k-1$ are scheduled in the sequence $(k-2; k-1)$.

Indeed, in the schedule σ^* one job of the set $\{k-2; k-1\}$ is processed on machine 1 during the interval τ and after the completion of this interval continues the processing on machine 2. After the completion of this job machine 2 processes the other job of the set $\{k-2; k-1\}$ and completes its processing at the time moment $f_{k-1}(\lambda) + q(\lambda - x)$. To obtain the minimal total completion time for the set $\{k-2; k-1\}$ we should schedule these jobs in the sequence $(k-2; k-1)$ because $p'_{k-2} < p'_{k-1}$.

One can easily see that Properties 1–7 uniquely determine the schedule σ^* which is the same as the schedule $\tilde{\sigma}_j$ constructed by algorithm for the case (c). This completes the proof of Theorem 1. □

Summing up, we should say the following. If $N_j = \{k-2; k-1; k\}$, the function $\Delta F_k(x)$ can have a breakpoint x^*. To find this breakpoint, one should solve the Eq. (5). In this case we have

$$\Delta F_k(x) = \begin{cases} q(\lambda - x) + q(x^* - x), & \text{if } x \in [\lambda - \delta, x^*), \\ q(\lambda - x), & \text{if } x \in [x^*, \lambda]. \end{cases}$$

For all other cases we have $\Delta F_k(x) = q(\lambda - x)$, where $x \in [\lambda - \delta, \lambda]$.

Thus, the increment $\Delta F_k(x)$ is a linear or a piecewise linear function.

5. Construction of the function $F_k^*(x) = \sum_{i=k}^{n} f_i(x)$

Consider jobs available not earlier than the time moment R_{j+1}. Suppose that job k completes the processing at the time moment $f_k(x) = r_{k+1} + x$, where $r_{k+1} = R_{j+1}$, $x \in [\lambda - \delta; \lambda]$. Recall that we schedule all unscheduled jobs using the strategy to complete each job as soon as possible (see Section 3, formulas (1), (2)).

Let us show how to determine the function $F_k^*(x) = \sum_{i=k}^{n} f_i(x)$, $x \in [\lambda - \delta; \lambda]$, which is used during the transformation of the schedule σ_j (see the procedure *SIGMA*(σ_j) in Section 3).

We shall use the following data structures:

- A sorted list BREAK which contains endpoints of the interval $[\lambda - \delta, \lambda]$ and all breakpoints $x_1, x_2, \ldots, x_{l-1}, x_l, \ldots$ of the function $F_k^*(x)$ ordered in such way that $\lambda - \delta = x_0 < x_1 < \cdots < x_{l-1} < x_l < \cdots < x_{l'} = \lambda$. Initially list BREAK contains only endpoints $\lambda - \delta$ and λ of the interval $[\lambda - \delta, \lambda]$.
- Sorted lists PHI, PSI, GFATHER, FATHER and SON which determine the current functions $\phi_i(x)$, $\psi_i(x)$, $f_{i-2}(x)$, $f_{i-1}(x)$ and $f_i(x)$, respectively. Each of these lists contains linear functions in the order which corresponds to intervals determined by points from list BREAK. Initially lists PHI, PSI and SON are empty, list GFATHER contains function $r_{k+1} + x$, list FATHER contains function $(1 - \frac{1}{q})x + r_{k+1} + \frac{p}{q}$.
- A sorted list TOTAL which determines the current value of the function $F_k^*(x)$. This list contains linear functions in the order which corresponds to intervals determined by list BREAK. Initially list TOTAL contains only one function $(x + r_{k+1}) + ((1 - \frac{1}{q})x + r_{k+1} + \frac{p}{q})$.

Jobs $k+2, k+3, \ldots, n$ are considered consecutively. At each step the current job i, $k+2 \leq i \leq n$, is treated. At first the contents of lists GFATHER and FATHER are assigned to lists PSI and PHI, respectively.

Let us describe how to obtain the current function $\psi_i(x) = \max\{f_{i-2}(x), r_i\}$ and fulfil the appropriate updating of lists BREAK, PSI, PHI and TOTAL. Since $f_{i-2}(x)$ is a nondecreasing convex function, the following cases may occur:

1. For the branch $y = a_1 x + b_1$, $x \in [x_0; x_1)$ of the function $f_{i-2}(x)$ determined by the first elements of lists BREAK and PSI the inequality $f_{i-2}(x_0) = a_1 x_0 + b_1 \geq r_i$ holds.
   We conclude that $f_{i-2}(x) \geq f_{i-2}(x_0) \geq r_i$ and $\psi_i(x) = f_{i-2}(x)$ for $x \in [\lambda - \delta; \lambda]$. In this case, the updating of lists is not carried out.

2. For the branch $y = a_{l'}x + b_{l'}, x \in [x_{l'-1}, x_{l'}]$, of the function $f_{i-2}(x)$ determined by the last elements of lists BREAK and PSI the inequality $f_{i-2}(\lambda) = f_{i-2}(x_{l'}) = a_{l'}x_{l'} + b_{l'} \le r_i$ holds.
   We conclude that $f_{i-2}(x) \le f_{i-2}(\lambda) \le r_i$ and $\psi_i(x) = r_i$ for $x \in [\lambda - \delta; \lambda]$. In this case in list PSI we replace each element (i.e. function) by r_i.
3. The inequalities $f_{i-2}(x_0) < r_i$ and $f_{i-2}(\lambda) > r_i$ hold.
   In this case the system

$$\begin{cases} y = f_{i-2}(x), \\ y = r_i, \\ x \in [\lambda - \delta; \lambda) \end{cases} \tag{11}$$

has the unique solution, because $f_{i-2}(x)$ is a nondecreasing convex function.
   For each branch $y = a_l x + b_l, x \in [x_{l-1}, x_l), 1 \le l \le l'$, of the function $f_{i-2}(x)$ determined by lists BREAK and PSI we consider the following system of linear equations and inequalities:

$$\begin{cases} y = a_l x + b_l, \\ y = r_i, \\ x_{l-1} \le x < x_l. \end{cases} \tag{12}$$

The system (11) is equivalent to the set of systems of the form (12). It follows, that only one system of the form (12) has the unique solution, while the other systems of this set have no solution. The following subcases may occur.
   (i) For some function $a_l x + b_l$ from list PSI the system (12) has the unique solution $x^* \in (x_{l-1}; x_l)$. This means that x^* is a breakpoint for the function $\psi_i(x) = \max\{f_{i-2}(x), r_i\}$. Furthermore, x^* will be a new breakpoint for the function $F_k^*(x)$. In this case let us do the following.
   Insert the point x^* in list BREAK after the point x_{l-1}. In list PSI we delete the first $l - 1$ elements (i.e. functions) and then before the function $a_l x + b_l$ insert l copies of r_i. In lists PHI and TOTAL after the l-th element we insert one additional element which is a copy of the l-th element.
   (ii) For some function $a_l x + b_l$ from list PSI the system (12) has the unique solution $x^* = x_{l-1}$.
   In this case in list PSI we replace the first $l - 1$ elements (functions) by $l - 1$ copies of r_i.

Analogously we obtain the current function $\phi_i(x) = \max\{f_{i-1}(x), r_i\}$ and fulfil the appropriate updating of lists BREAK, PHI, PSI and TOTAL.

Now form list SON which determines the function $f_i(x)$. Consider lists PHI and PSI from the beginning to the end. Each element of list SON (i.e. each branch of the function $f_i(x)$) is obtained by applying the formula (2) to the corresponding elements of lists PHI and PSI (i.e. to the corresponding branches of the functions $\phi_i(x)$ and $\psi_i(x)$).

Finally we update list TOTAL. To each element from list TOTAL we add the element with the same number from list SON.

To conclude the step, we delete the contents of list GFATHER and assign to this list the contents of list FATHER. Then replace the contents of list FATHER by contents of list SON and delete the contents of lists PHI, PSI and SON.

Repeat the described updating of lists for job $i + 1$. When the updating of lists for job n has been completed, stop.

At each step list BREAK is supplemented by no more than one breakpoint. Therefore, by construction $F_k^*(x) = \sum_{i=k}^{n} f_i(x)$, $x \in [\lambda - \delta; \lambda]$, is a nondecreasing piecewise linear function with no more than $O(n)$ breakpoints.

Notice, that at each step the updating of lists is done in $O(n)$ time. Since no more than n jobs are considered (i.e. no more than n steps are fulfilled), the construction of the function $F_k^*(x)$ can be done in $O(n^2)$ time.

6. The justification of the solution correctness

In this section we prove the theorem that justifies Algorithm G.

Theorem 2. *Algorithm G generates an optimal schedule for the $Q2|r_i, p_i = p, pmtn|\sum C_i$ problem.*

Proof. We shall prove this theorem by contradiction. The proof is partially based on the scheme developed by Herrbach and Leung [3].

Consider an instance I with the smallest (in terms of number of distinct release dates) set $N = \{1, 2, \dots, n\}$ of jobs that violates the theorem. Suppose that there are z distinct release dates $R_1 < R_2 < \dots < R_z$ for jobs of the set N. It is clear that $z \ge 2$. The set N is ordered in nondecreasing order of release dates of jobs.

Let s^* be the schedule produced by Algorithm G for the instance I. We have supposed that s^* is not optimal schedule.

We claim that none of the blocks of s^*, except the last one, is scheduled by Rule 1 of Algorithm G. Indeed, if in s^* some block is scheduled by Rule 1, we can easy construct an instance I' with a smaller set $N' \subset N$ of jobs that violates the theorem. This contradicts the assumption that I has the smallest set of jobs violating the theorem.

We also claim that the block $(R_{z-1}, R_z]$ of s^* is not scheduled by Rule 2 of Algorithm G. Suppose that it is not the fact. Since the last block of s^* has not been transformed, we can construct an instance I' with a set $N' = N$ of jobs by letting the jobs released at time R_z be released at time R_{z-1}. The set N' of jobs is smaller (in terms of number of distinct release dates) than the set N. For I' Algorithm G produces the same schedule s^*, while the optimal schedule for I' must have the value of the objective function no larger than that for I. Therefore, I' has a smaller set of jobs violating the theorem, contradicting the assumption that I has the smallest such set of jobs.

Thus, there is at least one block in s^*, namely the block $(R_{z-1}, R_z]$, that is scheduled by Rule 3.

Consider the structure of the blocks scheduled by Rule 3 in s^*. Let Algorithm G construct the subschedule of s^* for some block $(R_j, R_{j+1}]$ by using Rule 3 (see the procedure $SIGMA(\sigma_j)$), jobs $k+1, k+2, \dots, n$ being available not earlier than the time moment R_{j+1}. Recall that job k completes the processing at the time moment $f_k(x) = R_{j+1} + x$ and the completion times of all jobs $k+1, k+2, \dots, n$ are determined by formulas (1).

The following cases may occur:

A. The function $\Delta F_k^*(x) - \Delta F_k(x)$, $x \in [\lambda - \delta; \lambda]$, achieves its maximal value in the endpoint $x = \lambda - \delta$. Therefore, the function $F_k(x) + F_k^*(x) = \sum_{i=1}^{k-1} f_i(x) + \sum_{i=k}^{n} f_i(x)$ is increasing on the interval $[\lambda - \delta; \lambda]$, because $\Delta F_k^*(x) - \Delta F_k(x) = F_k^*(\lambda) - F_k^*(x) - (F_k(x) - F_k(\lambda)) = (F_k(\lambda) + F_k^*(\lambda)) - (F_k(x) + F_k^*(x))$.
B. The function $\Delta F_k^*(x) - \Delta F_k(x)$, $x \in [\lambda - \delta; \lambda]$, achieves its maximal value in the endpoint $x = \lambda$. In this case the function $F_k(x) + F_k^*(x)$ is decreasing on the interval $[\lambda - \delta; \lambda]$.
C. The function $\Delta F_k^*(x) - \Delta F_k(x)$, $x \in [\lambda - \delta; \lambda]$, achieves its maximal value in a breakpoint $x' \in (\lambda - \delta; \lambda)$. Then the function $F_k(x) + F_k^*(x)$ is decreasing on the interval $[\lambda - \delta; x']$ and is increasing on the interval $[x'; \lambda]$.

At each step of Algorithm G the number of unscheduled jobs becomes smaller. So the structure of the blocks scheduled by Rule 3 will be the following. At first, blocks are situated in which the amount of processing was increased by the maximal value $q\delta$ (the value δ was determined individually for each of these blocks). We shall call these blocks with the maximal possible amount of processing blocks of the first type. Then these blocks may be followed by a block in which the amount of processing was increased by the value $q(\lambda - x')$, where $x' \in (\lambda - \delta; \lambda)$ is a breakpoint of the function $\Delta F_k^*(x) - \Delta F_k(x)$. Finally, there may situate blocks in which the SPT-subschedules were not transformed, because the number of the remaining unscheduled jobs was rather small for obtaining a profitable transformation. We shall call the blocks in which the assigned amount of processing is not the maximal possible amount blocks of the second type.

Let s^0 denote an optimal schedule for the instance I. Since s^* is not optimal, we have

$$\sum_{i=1}^{n} C_i(s^0) < \sum_{i=1}^{n} C_i(s^*). \tag{13}$$

Let $l+1, l+2, \dots, n$ $(1 \le l < n)$ be the jobs with the release date R_z. Consider the new instance $\tilde{I}$ with the set $\tilde{N} = \{1, 2, \dots, l\}$ of jobs. Denote $\tilde{s}$ the schedule produced by Algorithm G for the instance $\tilde{I}$. The structure of the schedule $\tilde{s}$ is analogous to the structure of the schedule s^*. However, block $z-1$ of s^* is scheduled by Rule 3 while block $z-1$ of $\tilde{s}$ is scheduled by Rule 1. Besides, for instance I Algorithm G at each step analyzes additional $n-l$ jobs (i.e. jobs $l+1, l+2, \dots, n$). So it may occur that for I it is more profitable to do more amount of processing in some block $(R_j; R_{j+1}]$, $1 \le j \le z-1$, than for $\tilde{I}$. Thus, in $\tilde{s}$ the number of the blocks of the first type is no greater than the number of the same blocks in s^*. Algorithm G increases the amount of processing done in each block by means of increasing the loading of machine 2. This leads to the increasing of the partial objective function. So we have

$$\sum_{i=1}^{l} C_i(\tilde{s}) \le \sum_{i=1}^{l} C_i(s^*). \tag{14}$$

1. Now assume for the moment that

$$\sum_{i=1}^{l} C_i(s^0) < \sum_{i=1}^{l} C_i(s^*). \tag{15}$$

Let $\tilde{s}^0$ denote an optimal schedule for the instance $\tilde{I}$. From (15) we have

$$\sum_{i=1}^{l} C_i(\tilde{s}^0) \le \sum_{i=1}^{l} C_i(s^0) < \sum_{i=1}^{l} C_i(s^*). \tag{16}$$

Taking into account (14) and (16), we consider the following cases.

(a) Suppose that we have

$$\sum_{i=1}^{l} C_i(s^0) < \sum_{i=1}^{l} C_i(\tilde{s}) \le \sum_{i=1}^{l} C_i(s^*). \tag{17}$$

Then from (16) we obtain

$$\sum_{i=1}^{l} C_i(\tilde{s}^0) \le \sum_{i=1}^{l} C_i(s^0) < \sum_{i=1}^{l} C_i(\tilde{s}). \tag{18}$$

However, (18) shows that the instance $\tilde{I}$ which has a smaller set of jobs violates the theorem, contradicting the assumption that I has the smallest such set of jobs. Therefore, the inequality (17) cannot hold.

(b) Suppose that we have

$$\sum_{i=1}^{l} C_i(\tilde{s}) \le \sum_{i=1}^{l} C_i(s^0) < \sum_{i=1}^{l} C_i(s^*). \tag{19}$$

For a schedule s let us denote by $a(s)$ the total time of processing the jobs of the set $\tilde{N} = \{1, 2, \ldots, l\}$ on machine 1 after the time moment R_z. Since in the schedule s^* block $z-1$ was constructed by Rule 3, in s^* machine 2 does not process the jobs of the set $\tilde{N}$ after the time moment R_z. Notice, that in $\tilde{s}$ machine 2 is idle after the time moment R_z. Besides, $a(\tilde{s}) > a(s^*)$, because in $\tilde{s}$ the block $z-1$ was scheduled by Rule 1. The inequality (15) means that in the blocks 1 to $z-1$ the loading of machine 2 in the schedule s^0 is less than its loading in s^*. Therefore, in the schedule s^0 machine 2 does not process the jobs of the set $\tilde{N} = \{1, 2, \ldots, l\}$ after the time moment R_z, otherwise one can show that the schedule s^0 is not optimal. Thus, we have

$$a(\tilde{s}) \ge a(s^0) > a(s^*), \tag{20}$$

because otherwise the inequality (19) is impossible.

i. Suppose that $a(\tilde{s}) = a(s^0)$. Consider the schedule $\tilde{s}$. Taking into account that machine 1 is busy in the interval $(R_z; R_z + a(\tilde{s})]$, let us assign jobs $l+1, l+2, \ldots, n$ for processing from the time moment R_z according to the SPT-rule. Denote the obtained schedule for the instance I by $\tilde{s}_1$. In fact, the schedule $\tilde{s}_1$ can be obtained during the construction of the schedule s^* by Algorithm G, because the number of blocks of the first type in $\tilde{s}$ (and, therefore, in $\tilde{s}_1$) is less than the number of blocks of the same type in s^*. However, Algorithm G constructed the schedule s^*.Thus, by construction we have

$$\sum_{i=1}^{n} C_i(\tilde{s}_1) > \sum_{i=1}^{n} C_i(s^*). \tag{21}$$

Now consider the schedule s^0. We claim that the schedule s^0 passes exactly one job to block z. Indeed, suppose that the schedule s^0 passes several jobs to block z. Without loss of generality, we may assume that these jobs were immediately scheduled in block z by the SPT-rule. In this case machine 2 processes these jobs in block z. However, as it was discussed above, it is impossible.

Thus, in the schedule s^0 the job passed from block $z-1$ was immediately scheduled in block z on machine 1, followed by jobs $l+1, l+2, \ldots, n$ in a SPT-fashion. Therefore, we obtain

$$\sum_{i=l+1}^{n} C_i(s^0) = \sum_{i=l+1}^{n} C_i(\tilde{s}_1). \tag{22}$$

Taking into account (21), (19) and (22), we get

$$\sum_{i=1}^{n} C_i(s^*) < \sum_{i=1}^{l} C_i(\tilde{s}_1) + \sum_{i=l+1}^{n} C_i(\tilde{s}_1) = \sum_{i=1}^{l} C_i(\tilde{s}) + \sum_{i=l+1}^{n} C_i(\tilde{s}_1)$$
$$\le \sum_{i=1}^{l} C_i(s^0) + \sum_{i=l+1}^{n} C_i(s^0) = \sum_{i=1}^{n} C_i(s^0),$$

which contradicts (13).

ii. Now suppose that

$$a(\tilde{s}) > a(s^0). \tag{23}$$

Denote u the smallest number of such block, that its amount of processing in the schedule $\tilde{s}$ differs from the amount of processing in s^0. Beginning from block u, consider how Algorithm G constructs the schedule s^*. At each step j of Algorithm G, $j \ge u$, the partial schedule s_j for the set N_j of jobs is constructed. Now let us take this partial schedule s_j and assign for processing all unscheduled jobs of the set N using the strategy to complete each job as soon as possible (see formulas (1)). Denote the obtained schedule by $\tilde{s}_j$. Thus, we get a finite sequence of schedules $\tilde{s}_u$, $\tilde{s}_{u+1}, \ldots, s^*$ such that the following inequalities hold:

$$\sum_{i=1}^{n} C_i(\tilde{s}_u) > \sum_{i=1}^{n} C_i(\tilde{s}_{u+1}) > \cdots \ge \sum_{i=1}^{n} C_i(s^*). \tag{24}$$

By the construction we have

$$a(\tilde{s}) = a(\tilde{s}_u) > a(\tilde{s}_{u+1}) > \cdots \ge a(s^*). \tag{25}$$

From (20), (23) and (25) we conclude that there are such consecutive schedules $\tilde{s}_w, \tilde{s}_{w+1}$, where $w \ge u$, that

$$a(\tilde{s}_w) > a(s^0) \ge a(\tilde{s}_{w+1}). \tag{26}$$

Without loss of generality, we suppose that Algorithm G increased the amount of processing done in the block $(R_{w+1}; R_{w+2}]$ of s^* by the maximal value $q\delta$. (If Algorithm G increased the amount of processing for this block by the value $q(\lambda - x')$, one can use the analogous reasoning.) According to remark (A), the function $F_k(x) + F_k^*(x) = \sum_{i=1}^{k-1} f_i(x) + \sum_{i=k}^{n} f_i(x)$, which was analyzed during step $w+1$ of Algorithm G, is increasing on the interval $[\lambda - \delta; \lambda]$. Notice, that the continuous function $f_l(x)$ is also increasing on the interval $[\lambda - \delta; \lambda]$ and $f_l(\lambda) = R_z + a(\tilde{s}_w)$, $f_l(\lambda - \delta) = R_z + a(\tilde{s}_{w+1})$. Due to (26) we conclude, that there exists such point $\hat{x} \in [\lambda - \delta; \lambda)$ that $f_l(\hat{x}) = R_z + a(s^0)$

and $F_k(\lambda-\delta)+F_k^*(\lambda-\delta)\le F_k(\hat{x})+F_k^*(\hat{x})$. The function $F_k(\hat{x})+F_k^*(\hat{x})$ determines the schedule $\hat{s}$ such that $C_l(\hat{s})=R_z+a(s^0)$ and

$$\sum_{i=1}^{n} C_i(\tilde{s}_{w+1}) \le \sum_{i=1}^{n} C_i(\hat{s}). \tag{27}$$

Recall, that $\tilde{s}$ is an optimal schedule for the set $\tilde{N}=\{1,2,\dots,l\}$ of jobs. Now suppose that all jobs of the set $\tilde{N}$ are to complete the processing by the time moment $R_z+a(s^0)$. By construction the subschedule of $\hat{s}$ for the set $\tilde{N}$ of jobs is an optimal schedule for the instance $\tilde{I}$ with this additional restriction. So we have

$$\sum_{i=1}^{l} C_i(\hat{s}) \le \sum_{i=1}^{l} C_i(s^0). \tag{28}$$

Analogously to (22), it is not difficult to show that

$$\sum_{i=l+1}^{n} C_i(s^0) = \sum_{i=l+1}^{n} C_i(\hat{s}). \tag{29}$$

Taking into account (24) and (27)–(29), we obtain

$$\sum_{i=1}^{n} C_i(s^*) \le \sum_{i=1}^{n} C_i(\tilde{s}_{w+1}) \le \sum_{i=1}^{n} C_i(\hat{s}) = \sum_{i=1}^{l} C_i(\hat{s}) + \sum_{i=l+1}^{n} C_i(\hat{s})$$
$$\le \sum_{i=1}^{l} C_i(s^0) + \sum_{i=l+1}^{n} C_i(s^0) = \sum_{i=1}^{n} C_i(s^0),$$

which contradicts (13).

Thus, the inequalities (20), (19) and (15) are impossible.

2. Suppose that

$$\sum_{i=1}^{l} C_i(s^0) \ge \sum_{i=1}^{l} C_i(s^*). \tag{30}$$

From (13) and (30) we have

$$\sum_{i=l+1}^{n} C_i(s^0) < \sum_{i=l+1}^{n} C_i(s^*). \tag{31}$$

Since in the schedule s^* block $z-1$ was constructed by Rule 3, in s^* machine 2 does not process the jobs of the set $\tilde{N}=\{1,2,\dots,l\}$ after the time moment R_z. It follows that

$$a(s^0) < a(s^*), \tag{32}$$

because otherwise the inequality (31) does not hold.

In s^* machine 1 has no idle periods in the interval $(R_1; R_z+a(s^*)]$ because none of the blocks of s^*, except the last one, is scheduled by Rule 1 of Algorithm G. So in s^0 it is possible to complete the processing of jobs $1,2,\dots,l$ on machine 1 earlier than the time moment $R_z+a(s^*)$ only if machine 2 in s^0 is more busy than in s^*. The following cases may occur.

(a) The block $(R_{z-1}; R_z]$ of the schedule s^* is a block of the first type. Then according to the structure of s^* all blocks 1 to $z-1$ scheduled by Rule 3 are blocks of the first type. We conclude that in s^0 it is impossible to do more processing on machine 2 in the interval $(R_1; R_z]$ than it was done in the schedule s^*. On the other hand, the inequality (32) holds. Therefore, in s^0 machine 2 processes jobs of the set $\tilde{N}$ in block z. We may assume that s^0 passes two or more jobs to block z. Without loss of generality, we assume that s^0 immediately schedules these jobs by the SPT rule in block z, followed by jobs $l+1, l+2,\dots,n$ in a SPT fashion.

Denote $\Delta x=a(s^*)-a(s^0)>0$. Then in s^0 machine 2 completes the processing of jobs of the set $\tilde{N}$ no earlier than at the time moment $R_z+q\Delta x$.

Now recall that $C_l(s^*)=R_z+a(s^*)$. According to formulas (1), we have

$$C_{l+1}(s^*) = R_z + \left(1-\frac{1}{q}\right)a(s^*) + \frac{p}{q},$$

$$C_i(s^*) = \left(1-\frac{1}{q}\right)C_{i-1}(s^*) + \frac{1}{q}C_{i-2}(s^*) + \frac{p}{q}, \qquad l+2\le i\le n.$$

Further, we obtain

$$C_{l+1}(s^0) \ge R_z + (a(s^*)-\Delta x) + \frac{p-(a(s^*)-\Delta x-q\Delta x)}{q}$$
$$= \left(R_z + a(s^*) + \frac{p-a(s^*)}{q}\right) - \Delta x + \frac{\Delta x}{q} + \Delta x = C_{l+1}(s^*) + \frac{\Delta x}{q};$$

$$\begin{aligned} C_{l+2}(s^0) &= \left(1-\frac{1}{q}\right)C_{l+1}(s^0)+\frac{1}{q}C_l(s^0)+\frac{p}{q} \geq \left(1-\frac{1}{q}\right)\left(C_{l+1}(s^*)+\frac{\Delta x}{q}\right) \\ &\quad +\frac{1}{q}(C_l(s^*)-\Delta x)+\frac{p}{q} = \left(\left(1-\frac{1}{q}\right)C_{l+1}(s^*)+\frac{1}{q}C_l(s^*)+\frac{p}{q}\right)+\left(1-\frac{1}{q}\right)\frac{\Delta x}{q}-\frac{\Delta x}{q} \\ &= C_{l+2}(s^*)-\frac{\Delta x}{q^2}; \\ C_{l+3}(s^0) &= \left(1-\frac{1}{q}\right)C_{l+2}(s^0)+\frac{1}{q}C_{l+1}(s^0)+\frac{p}{q} \\ &\geq \left(1-\frac{1}{q}\right)\left(C_{l+2}(s^*)-\frac{\Delta x}{q^2}\right)+\frac{1}{q}\left(C_{l+1}(s^*)+\frac{\Delta x}{q}\right)+\frac{p}{q} = C_{l+3}(s^*)+\frac{\Delta x}{q^3}; \\ C_{l+4}(s^0) &= \left(1-\frac{1}{q}\right)C_{l+3}(s^0)+\frac{1}{q}C_{l+2}(s^0)+\frac{p}{q} \\ &\geq \left(1-\frac{1}{q}\right)\left(C_{l+3}(s^*)+\frac{\Delta x}{q^3}\right)+\frac{1}{q}\left(C_{l+2}(s^*)-\frac{\Delta x}{q^2}\right)+\frac{p}{q} = C_{l+4}(s^*)-\frac{\Delta x}{q^4}; \\ &\dots \end{aligned}$$

Therefore, we have

$$\sum_{i=l+1}^{n} C_i(s^0) \geq \sum_{i=l+1}^{n} C_i(s^*)+\Delta x\left(\frac{1}{q}-\frac{1}{q^2}+\frac{1}{q^3}-\frac{1}{q^4}+\cdots\right) > \sum_{i=l+1}^{n} C_i(s^*), \tag{33}$$

because $\Delta x(\frac{1}{q}-\frac{1}{q^2}+\frac{1}{q^3}-\frac{1}{q^4}+\cdots) > 0$. The inequality (33) contradicts (31).

(b) The block $(R_{z-1}; R_z]$ of the schedule s^* is a block of the second type.

Denote the smallest number of the second type block in s^* by u. Now beginning from block u, transform the schedule s^* in the following way. Consider blocks u to $z-1$ consecutively and increase in each of them the amount of processing by the maximal possible value δ (the value δ is determined individually for each block). Thus, for each set N_j of jobs, $u \leq j \leq z-1$, we construct the transformed partial schedule $\bar{s}_j$. Then we supplement $\bar{s}_j$ by subschedule for jobs of the set $N \setminus N_j$, in which these jobs complete the processing as soon as possible (see formulas (1)). Denote the obtained schedule for jobs of the set N by $\tilde{s}_j$. Thus, we get a finite sequence of schedules $s^*, \tilde{s}_u, \tilde{s}_{u+1}, \dots, \tilde{s}_{z-1}$.

It should be mentioned that Algorithm G does not transform subschedules of s^* constructed by the SPT-rule for blocks $u, u+1, \dots, z-1$. (Notice, that Algorithm G can transform the subschedule of s^* for block u, but for this block the amount of processing is not increased by the maximal value). This means that it is not profitable to increase the amount of processing by the maximal value for each of these blocks. Moreover, for the larger number of a block it is more unprofitable to increase the amount of processing in this block. So we have

$$\sum_{i=1}^{n} C_i(s^*) < \sum_{i=1}^{n} C_i(\tilde{s}_u) < \sum_{i=1}^{n} C_i(\tilde{s}_{u+1}) < \cdots < \sum_{i=1}^{n} C_i(\tilde{s}_{z-1}). \tag{34}$$

Besides, by the construction we have

$$a(s^*) > a(\tilde{s}_u) > a(\tilde{s}_{u+1}) > \cdots > a(\tilde{s}_{z-1}). \tag{35}$$

At first suppose, that in s^0 the maximal amount of processing is done in each block $1, 2, \dots, z-1$. Notice, that in s^0 it is impossible to do more processing in the time interval $(R_1; R_z]$ than it was done in the schedule $\tilde{s}_{z-1}$. So the inequality

$$a(s^0) < a(\tilde{s}^*_{z-1})$$

holds only if after the time moment R_z machine 2 processes the jobs of the set $\tilde{N} = \{1, 2, \dots, l\}$. However, in this case we have

$$\sum_{i=1}^{n} C_i(\tilde{s}_{z-1}) < \sum_{i=1}^{n} C_i(s_0). \tag{36}$$

Combining (34) and (36), we obtain

$$\sum_{i=1}^{n} C_i(s^*) < \sum_{i=1}^{n} C_i(\tilde{s}_{z-1}) < \sum_{i=1}^{n} C_i(s^0)$$

which contradicts (13). Therefore, we have

$$a(s^0) \geq a(\tilde{s}^*_{z-1}). \tag{37}$$

Now suppose that in some of blocks $1, 2, \dots, z-1$ of the schedule s^0 the amount of processing is not maximal. Then if machine 2 processes the jobs of the set $\tilde{N} = \{1, 2, \dots, l\}$ after the time moment R_z, the schedule s^0 cannot be optimal, because it is more profitable to do more processing on machine 2 before the time moment R_z.

Thus, in the schedule s^0 machine 2 does not process the jobs of the set $\tilde{N}$ after the time moment R_z, and the inequality (37) holds.

From (32), (35) and (37) we conclude, that either there are such consecutive schedules $\tilde{s}_w, \tilde{s}_{w+1}$, where $u \leq w \leq z-2$, that

$$a(\tilde{s}_w) > a(s^0) \geq a(\tilde{s}_{w+1}),$$

or

$$a(s^*) > a(s^0) \geq a(\bar{s}_u).$$

According to remark (B), the function $F_k(x) + F_k^*(x) = \sum_{i=1}^{k-1} f_i(x) + \sum_{i=k}^{n} f_i(x)$, which can be constructed at step $w+1$ (or at step u) of transformation of s^*, is decreasing on the interval $[\lambda - \delta; \lambda]$. (If Algorithm G increased the amount of processing for block u by the value $q(\lambda - x')$, one can use remark (C) and the analogous reasoning). So we conclude that there exists such point $\hat{x} \in (\lambda - \delta; \lambda]$ that $f_l(\hat{x}) = R_z + a(s^0)$ and $F_k(\lambda) + F_k^*(\lambda) \leq F_k(\hat{x}) + F_k^*(\hat{x})$. The function $F_k(\hat{x}) + F_k^*(\hat{x})$ determines the schedule $\hat{s}$ such that $C_l(\hat{s}) = R_z + a(s^0)$ and

$$\sum_{i=1}^{n} C_i(\bar{s}_w) < \sum_{i=1}^{n} C_i(\hat{s}) \tag{38}$$

$$\left(\text{or} \quad \sum_{i=1}^{n} C_i(s^*) < \sum_{i=1}^{n} C_i(\hat{s})\right).$$

Recall, that in the schedules s^0 and $\hat{s}$ machine 2 does not process the jobs of the set $\tilde{N}$ after the time moment R_z. Therefore, the inequality of the form (29) holds. By construction the subschedule of $\hat{s}$ for the set $\tilde{N}$ of jobs is an optimal schedule for the instance $\tilde{I}$ with the restriction that all jobs of the set $\tilde{N}$ are to complete the processing by the time moment $R_z + a(s^0)$. It follows that the inequality of the form (28) holds. Therefore, from (34) and (38) it is easy to obtain

$$\sum_{i=1}^{n} C_i(s^*) < \sum_{i=1}^{n} C_i(\hat{s}) = \sum_{i=1}^{l} C_i(\hat{s}) + \sum_{i=l+1}^{n} C_i(\hat{s}) \leq \sum_{i=1}^{l} C_i(s^0) + \sum_{i=l+1}^{n} C_i(s^0) = \sum_{i=1}^{n} C_i(s^0),$$

which contradicts (13).

Thus, the inequality (30) is impossible.

Therefore, s^* must be an optimal schedule. □

Acknowledgements

I am grateful to the anonymous referees for their comments that improved the presentation.This research was partially supported by INTAS, Project 03-51-5501.

References

[1] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic scheduling and sequencing: A survey, Annals of Discrete Mathematics 5 (1979) 287–326.
[2] P. Brucker, S. Knust, Complexity results for scheduling problems, http://www.mathematik.uni-osnabrueck.de/research/OR/class/.
[3] L.A. Herrbach, J.Y.-T. Leung, Preemptive scheduling of equal length jobs on two machines to minimize mean flow time, Operations Research 38 (1990) 487–494.
[4] P. Baptiste, P. Brucker, M. Chrobak, C. Dürr, S.A. Kravchenko, F. Sourd, The complexity of mean flow time scheduling problems with release times, Journal of Scheduling 10 (2007) 139–146.
[5] J. Du, J.Y.-T. Leung, G.H. Young, Minimizing mean flow time with release time constraint, Theoretical Computer Science 75 (1990) 347–355.
[6] T. Gonzalez, Optimal mean finish time preemptive schedules, Technical Report 220, Computer Science Department, Pennsylvania State University, 1977.
[7] E.L. Lawler, Recent results in the theory of machine scheduling, Math. Progr. State Art: 11th Int. Symp.-Bonn (1982) 202–234.