INFORMATICA, 2014, Vol. 25, No. 1, 37–53
© 2014 *Vilnius University*

A Problem of Scheduling Jobs with Non-Monotonic Stepwise Values

Adam JANIAK*, Tomasz KRYSIAK, Radosław TRELA
Institute of Computer Engineering, Control and Robotics, Wrocław University of Technology, Janiszewskiego 11/17, 50-372 Wrocław, Poland
e-mail: adam.antoni.janiak@gmail.com, tomasz.a.krysiak@gmail.com, radoslaw.trela@pwr.wroc.pl

Received: November 2011; accepted: November 2013

Abstract. The paper deals with a parallel processor scheduling problem with changeable job values. Contrary to other papers in this area, we assume that job values are characterized by a *non-monotonic* stepwise functions of job completion times (previously only non-increasing functions have been considered). We give examples of real-life systems that can be modelled in such a way, in order to show that the problem is interesting from practical point of view. The problem is shown to be NP-hard and a pseudo-polynomial time algorithm for its special case is constructed. Moreover, a number of heuristic algorithms is provided and experimentally tested.

Key words: scheduling, parallel processors, non-monotonic job value, pseudo-polynomial algorithm, heuristic

1. Introduction

Scheduling problems are a kind of combinatorial optimization problems in which the purpose is to determine when to perform some given tasks (called ‘jobs’), with which resource, and on which equipment (‘processors’ or ‘machines’) in order to optimize some certain criterion or criteria, e.g. time or cost. In this paper we consider a scheduling problem of maximization of the total job value, i.e. scheduling jobs on processors in a such way that the sum of job values is maximal. In such problems, the ‘job value’ is a parameter characterizing each job, which can vary over time. This means that the largeness of the value that a given task will provide depends on the time when it will be finished. In turn, the times at which the jobs are performed depend on an order (sequence) of the jobs on the processors (such an order, together with the moments of starting and completion times of the jobs, is called a ‘schedule’). Therefore, the problem is to find such a schedule of the jobs on the processors that the sum of values of these jobs is maximal. We assume that in our problem the job value is a stepwise function of job completion time. However, contrary to other papers devoted to such problems, we assume that this function may be non-monotonic. The formal definition of the problem and examples of practical application will be given later on.

*Corresponding author.

The term ‘job value’ was introduced to the scheduling theory by Voutsinas and Pappis (2002). They formulated the single processor problem of maximization of the sum of job values as a model of a disassembly process of products (e.g. computers) after reaching the end of their useful life-times in order to be remanufactured. The job value was described by a non-increasing power function of the completion time. In order to solve the problem, they constructed heuristic algorithms. Moreover, they showed polynomial-time exact algorithms for some special cases of the problem.

Yang (2009) dealt with the similar problem – the maximization of the total revenue of products modelled as a non-increasing exponential function of job completion time. As a practical application he gave a production planing in contemporary hyper-competitive marketplace where the amount of revenue generated as a result of completing a product (job) may be decreasing as its completion time is delayed. The paper contains some rules for dominant schedules and a branch and bound algorithm and heuristic algorithms solving the problem.

Different model of job values was suggested by Janiak and Krysiak (2007). They assumed that the job value is a non-increasing stepwise function of the completion time. As a practical application, establishing an order of processing of datagrams sent by a router in IP protocol was given. In the paper, the equivalence of the single processor problem with maximization of the total stepwise job values to the well-known, NP-hard in the ordinary sense, problem of minimizing weighted number of late jobs was shown. Moreover, NP-hardness in the ordinary sense of the case with parallel processors and arbitrary number of common moments of job value changes was proved by a construction of a pseudo-polynomial time algorithm based on the dynamic programming method. Also, heuristic algorithms were constructed.

Janiak *et al.* (2009) investigated a problem with parallel processors and the job value described by a difference between given constant initial value and a non-decreasing power loss function of the job completion time. They proved strong NP-hardness of the general problem and NP-hardness of a case with a single processor. The branch and bound algorithm and several heuristic algorithms were constructed. What is more, some special polynomially solvable cases were shown.

The same model of job values was considered in Janiak *et al.* (2010), but for single processor scheduling problem and with non-zero release dates. The main results of this paper are solution algorithms of the considered problem. They are: a branch and bound exact algorithm and several heuristic algorithms.

The scheduling problems with non-increasing stepwise job values and its extension were additionally analyzed in Janiak and Krysiak (2012). In the paper, the single processor problem with the stepwise job value was proved to be strongly NP-hard. For the case with unrelated parallel processors, a dynamic programming algorithm (pseudo-polynomial time if the number of processors and the number of common moments of job value changes are arbitrary) and several heuristic algorithms (based on the list strategy) were constructed. Additionally, the single processor scheduling problem with piecewise linear losses of job values was considered. It was proved that the problem is strongly NP-hard and that there exists a pseudo-polynomial time algorithm for its special case. Also, heuristic algorithms were constructed.

Scheduling problems with time deteriorating (non-increasing) job values were also addressed by Raut et al. (2008a, 2008b). They considered single processor problems and proved that problem with concave value function is NP-hard, with convex value function is strongly NP-hard and with linear value function is polynomially solvable. They constructed several heuristic algorithms. As a practical application for the problem, scheduling of movies at different screens of a multiplex theater in the motion picture industry was shown.

There are much more papers devoted to scheduling problems with job parameters changeable over time. However, in most of them job processing times are changeable (learning effect, aging effect, deteriorating jobs). In some of them, changeable (non-decreasing) job processing times are even combined with changeable (non-increasing) job values (see e.g., Pappis and Rachaniotis, 2010, and Rachaniotis and Pappis, 2006). In spite of the fact that many different models of changeable job processing times were considered, even non-monotonic, to the best of our knowledge only monotonic (non-increasing) functions of job values were considered so far in the context of scheduling. However, from practical point of view it is also interesting to analyze non-monotonic job value functions. Therefore, in this paper we investigate a parallel processor scheduling problem with non-monotonic stepwise job values, giving appropriate application examples.

It is worth mentioning also that, contrary to most of non-monotonic models of job processing times (see e.g. Huang *et al.*, 2012, where job processing time is a non-decreasing function of the total normal processing times of the jobs already processed and a non-increasing function of the job's position in the sequence, or Huang and Wang, 2012), our model of job value is a function of one variable.

The remaining part of the paper is organized as follows. In the next section, the detailed formulation of the problem is given. Then, in Section 3, we show a few practical applications for it. In Section 4 we show that our problem is at least NP-hard and in Section 5 we construct a pseudo-polynomial time algorithm for a special case of the considered problem. Section 6 contains several heuristic algorithms and in Section 7 numerical experiment comparing them is described and obtained results are shown. Some concluding remarks are given in Section 8.

2. Problem Formulation

There are given a set of m identical parallel processors $M = \{M_1, \dots, M_m\}$ and a set of n independent and non-preemptive jobs $J = \{J_1, \dots, J_n\}$ immediately available for processing at time 0. Each processor $M_i \in M$ can execute at most one job at a time. Each job $J_j \in J$ is characterized by its processing time $p_j > 0$, its value $v_j(t)$ calculated at time t and moments $d_{jq} > 0$, $q = 1, \dots, k-1$ when changes of job value occur. The job value is described by the following non-monotonic stepwise function:

$$v_j(t) = \begin{cases} w_{j1}, & 0 < t \leqslant d_{j1}, \\ w_{j2}, & d_{j1} < t \leqslant d_{j2}, \\ \vdots & \\ w_{jk}, & d_{jk-1} < t, \end{cases} \tag{1}$$

such that $\forall_{J_j \in J}\ w_{jl} \geqslant 0,\ 1 \leqslant l \leqslant k$. We assume additionally that p_j and d_{jq} are positive integers. Let us denote model (1) by *NMV step* (non-monotonic stepwise value model).

Solution of the problem is a vector of pairs $C = \{(C_1, z_1), (C_2, z_2), \dots, (C_n, z_n)\}$, where C_j is the completion time of job J_j and $z_j \in M$ denotes the processor in which this job will be processed. The objective is to find such a solution C^* which maximizes the sum of values of all the jobs calculated at their completion times C_j: $\sum_{j=1}^{n} v_j(C_j) \to \max$.

For convenience, the above formulated problem may be represented according to the three field notation (Graham *et al.*, 1979) as $Pm|NMVstep|\sum v_j(C_j)$.

3. Examples of Practical Application of the Problem

3.1. *Timetable for a School*

Assume that a timetable for a school must be prepared. We are given a set of subjects and a weekly fixed number of hours intended for each subject.

A human organism is not able to do some certain mental work at any time with the same efficiency. It is caused by so called circadian rhythms, i.e. a roughly 24-hour cycles in biochemical, physiological, or behavioural processes. In mammals such rhythms are driven by two mechanisms: endogenous (biological clock) and exogenous (concerned with some environmental factors, mainly with light). Despite the fact that in normal conditions the endogenous mechanism is subordinated to the exogenous one, it has considerable individual autonomy, what makes it robust to sudden changes of e.g. light intensity (Sadowski, 2007).

One of the abilities, that are influenced by circadian rhythms, is mental capacity. In scientific research, different mental capacity curves were obtained depending on the tests. For example, searching particular letter in a text was performed relatively slow early in the morning and much faster around noon, but memorizing of text was easier in the morning. Therefore, it is thought that each mental activity can have its own circadian rhythm dependent on a specific oscillator (Sadowski, 2007).

Thus, it will be profitable to prepare such a timetable, in which each subject is located in a period of time when necessary abilities reach their peak. Obviously, usually it is not possible to place all subjects in optimal time. Therefore, some (most important) subjects should have priority over the others.

In the context of job scheduling, lessons per week devoted to each subject may be considered as jobs and days of week may be seen as processors. Job value will be the highest in a period of time when the necessary to this subject specific mental capacity reaches its peak. Moreover, maximal value of some jobs will be higher than the others. Assuming constant mental capacity during the lesson, a stepwise model of job value is obtained. The objective is to find such a timetable in which the sum of job values is maximal.

3.2. *Production Schedule*

Assume that a company has to produce a set of items and each of them must be available to the customer at a certain date. Each product has its own price and each product has its own unit storage cost and a unit tardiness penalty. The profit is a difference between the income (price $\times$ number of sold items) and the total cost (penalties, storage costs, etc.). The profit is the biggest when all products are made in their due dates (no penalties, no storage).

This problem may be concerned as a flow shop problem or even a job shop problem. Processors are machines in the factory, jobs are activities necessary to produce items. Each job consists of a set of operations. An operation is a production activity concerning one job on one machine. What is more, each job is characterized by a function of its value, calculated at a completion of the job – the difference between the price and the total cost. The objective is to find such a schedule that maximizes the sum of job values (total profit).

3.3. *Establishing a Sequence of Gathering of Crops and Fruits*

Scheduling of jobs with changeable values may be also useful in establishing a sequence of gathering of crops and fruits in a farm or in an orchard. Assume that a farmer grows different kinds of crops and fruits. Each kind of crops and fruits ripens at a specific period and should be gathered then, because reaches then the highest price. However, the periods may overlap and there is a limited number of employees and machines in the farm, so it might be not possible to gather everything in the optimal time. Fortunately, some fruits or crops may be gathered earlier or later and used e.g. as a fodder or as an ingredient of jam. Obviously, the price is then lower. Notice that prices of different kinds or species may be different and also may vary in different way.

In the context of scheduling, groups of employees may be concerned as processors and gathering of particular species as jobs. Job values are related with prices of gathered fruits or crops. The objective is to determine such a schedule of gathering that provides the maximal sum of job values (maximal profit).

4. Computational Complexity

In this section, we show that the problem under consideration is at least NP-hard, since its special case with monotonic job values is already NP-hard in the ordinary sense.

Theorem 1. *The problem $Pm|NMVstep|\sum v_j(C_j)$ is at least NP-hard.*

Proof. In Janiak and Krysiak (2007) the problem of scheduling n independent, non-preemptive jobs on a single processor, where job value is described by a stepwise function (1) with monotonicity condition $w_{j1} > w_{j2} > \cdots > w_{jk} > 0$, was proved to be NP-hard, even for the special case with single moment of change of value for every job (the moments may differ for different jobs). Note that such a problem is a special case of the problem formulated in Section 2. If we relax the constraint $w_{j1} > w_{j2} > \cdots > w_{jk} > 0$, we obtain the special case of the problem $Pm|NMVstep|\sum v_j(C_j)$ (with $m = 1$). $\square$

5. Pseudo-Polynomial Time Algorithm

In this section a pseudo-polynomial time algorithm (based on the dynamic programming method) solving a special case of $Pm|NMVstep|\sum v_j(C_j)$ is shown. In this case we have common for all the jobs moments of changes of job values ($d_{jq} = d_q$ for $j = 1, \dots, n$), constant (fixed) number moments of change of job values (fixed k), and constant (fixed) number of processors (fixed m). Let us represent this special case by $Pm|NMVstep,\ d_{jq} = d_q, k\ fixed|\sum v_j(C_j)$.

This algorithm is a generalization of the algorithm for the corresponding special case of the problem with monotonic functions of job values described in Janiak and Krysiak (2007). Since in optimal solution of the problem with non-monotonic functions of job values processor idle times are allowed, additional sorting of jobs (in the non-decreasing order of their processing times inside each interval) and validation (of starting times of the jobs belonging to each interval – see inequalities (3)–(4)) are required to ensure that all jobs fit to the intervals they are assigned to.

Jobs will be considered here in their natural order $1, 2, \dots, n$. We have $k \cdot m$ disjunctive sets of jobs:

$$X_1^1, \dots, X_k^1, X_1^2, \dots, X_k^2, \dots, X_1^m, \dots, X_k^m,$$

where jobs from sets $X_1^i, \dots, X_k^i$ are scheduled on the processor M_i, $i = 1, \dots, m$, in such a way that jobs from the set X_{q-1}^i are sequenced before jobs from the set X_q^i, and processing of jobs from X_q^i finishes before d_q, $q = 1, \dots, k$, and $d_k = \infty$. Moreover, inside the set X_q^i jobs are processed according to the non-decreasing order of p_l, where $l \in X_q^i$. In the algorithm jobs will be assigned to the sets $X_1^1, \dots, X_k^m$, which are empty at the beginning. We assume that assignment of the job J_j to the set X_q^i contributes w_{jq} to the objective function.

Let us define a function

$$W_j\big(P_1^1, \dots, P_{k-1}^1, P_1^2, \dots, P_{k-1}^2, \dots, P_1^m, \dots, P_{k-1}^m\big)$$

as the maximum sum of job values for partial schedules containing first j jobs, for which the sum of processing times of jobs assigned to the set X_q^i equals to P_q^i, $q = 1, \dots, k-1$. Notice that if any job is assigned to any set X_k^i, $i = 1, \dots, m$, for the objective function it does not matter which exactly set it is. Therefore, the values P_k^i, $i = 1, \dots, m$ does not matter as well. We are only interested in the sum of these values, calculated as follows:

$$\sum_{i=1}^{m} P_k^i = T_j - (P_1^1 + \cdots + P_{k-1}^m),$$

where $T_j = \sum_{l=1}^{j} p_l$.

Let us consider a partial schedule $(j, P_1^1, \ldots, P_{k-1}^m)$ and the corresponding value of the function $W_j(P_1^1, \ldots, P_{k-1}^m)$. If the last scheduled job (J_j) was assigned to the set X_q^i, P_q^i increased by p_j and the value of the function $W_{j-1}(P_1^1, \ldots, P_q^i - p_j, \ldots, P_{k-1}^m)$ increased by w_{jq} to $W_j(P_1^1, \ldots, P_q^i, \ldots, P_{k-1}^m)$.

Let us denote the starting time of jobs assigned to the set X_q^i by $S_{X_q^i}$:

$$S_{X_q^i} = \min\{S_{X_{q+1}^i}, d_q\} - P_q^i, \quad q = 1, \ldots, k-2,$$
$$S_{X_{k-1}^i} = d_{k-1} - P_{k-1}^i,$$
$$S_{X_k}^i = d_{k-1}.$$

It is more convenient to assign jobs from the end of the interval to the beginning because of right-closed intervals. If the state $(j, P_1^1, \ldots, P_{k-1}^m)$ corresponds to the partial schedule which may lead to the optimal schedule, the following conditions must be satisfied:

$$\sum_{l=1}^{q} P_l^i \leqslant d_q, \quad q = 1, \ldots, k-1, \tag{2}$$
$$S_{X_1^i} \geqslant 0, \tag{3}$$
$$S_{X_{r+1}^i} > d_r - \max_{l \in X_{r+1}^i} p_l, \quad r = 1, \ldots, k-2, \tag{4}$$

for each $i = 1, \ldots, m$. Taking into account the above inequalities allows us to avoid checking some states that do not lead to the optimal solution.

Thus, for $j = 1, \ldots, n$ we can calculate recurrently $W_j(P_1^1, \ldots, P_{k-1}^m)$ for all possible values of P_q^i, $q = 1, \ldots, k-1$, $i = 1, \ldots, m$ and obtain the optimal value of the objective function

$$W^* = \max_{\substack{1 \leqslant q \leqslant k-1 \\ 1 \leqslant r \leqslant k-2 \\ 1 \leqslant i \leqslant m}} \Bigg\{ W_n\big(P_1^1, \ldots, P_{k-1}^m\big) \,\big|\, P_q^i = 0, 1, \ldots, \min\{T_n, d_q\},$$
$$\sum_{l=1}^{q} P_l^i \leqslant d_q,\ S_{X_1^i} \geqslant 0,\ S_{X_{r+1}^i} > d_r - \max_{l \in X_{r+1}^i} p_l \Bigg\}.$$

A formal description of the above algorithm is given as follows:

Algorithm 1 P-Polyn

Step 1: Set

$$W_j(P_1^1,\dots,P_{k-1}^m) := \begin{cases} 0, & \text{if } (j,P_1^1,\dots,P_{k-1}^m)=(0,0,\dots,0),\\ -\infty, & \text{otherwise,}\end{cases}$$

for $j=0,1,\dots,n$ and $P_q^i=0,1,\dots,\min\{T_n,d_q\}$; $i=1,\dots,m$; $q=1,\dots,k-1$. Then set $j:=1$.

Step 2: For each $P_q^i=0,1,\dots,\min\{T_j,d_q\}$, where $i=1,\dots,m$ and $q=1,\dots,k-1$, determine:

$$W_j(P_1^1,\dots,P_{k-1}^1,P_1^2,\dots,P_{k-1}^2,\dots,P_1^m,\dots,P_{k-1}^m)$$
$$= \begin{cases} \max\limits_{1\leqslant i\leqslant m} \left\{\begin{matrix} W_{j-1}(P_1^1,\dots,P_1^i-p_j,P_2^i,\dots,P_{k-1}^i,\dots,P_{k-1}^m)+w_{j1},\\ W_{j-1}(P_1^1,\dots,P_1^i,P_2^i-p_j,\dots,P_{k-1}^i,\dots,P_{k-1}^m)+w_{j2},\\ \vdots\\ W_{j-1}(P_1^1,\dots,P_1^i,P_2^i,\dots,P_{k-1}^i-p_j,\dots,P_{k-1}^m)+w_{jk-1},\\ W_{j-1}(P_1^1,\dots,P_1^i,P_2^i,\dots,P_{k-1}^i,\dots,P_{k-1}^m)+w_{jk},\end{matrix}\right\}, \\ \quad \text{if inequalities (2)–(4) are satisfied,}\\ -\infty, \quad \text{otherwise.}\end{cases}$$

If the maximum in the above expression is reached on (q,i) component, the job J_j is assigned to the set X_q^i, $q\in\{1,\dots,k\}$, $i\in\{1,\dots,m\}$. Inside the set X_q^i jobs are sequenced according to the non-decreasing order of p_l, $l\in X_q^i$. If $j=n$, go to Step 3, otherwise assign $j:=j+1$ and repeat Step 2.

Step 3: Determine the optimal value of the objective function:

$$W^* = \max_{\substack{1\leqslant q\leqslant k-1\\1\leqslant r\leqslant k-2\\1\leqslant i\leqslant m}} \Bigg\{ W_n\big(P_1^1,\dots,P_{k-1}^m\big) \mid P_q^i=0,1,\dots,\min\big\{T_n,d_q\big\},$$
$$\sum_{l=1}^{q} P_l^i \leqslant d_q,\ S_{X_1^i}\geqslant 0,\ S_{X_{r+1}^i} > d_r - \max_{l\in X_{r+1}^i} p_l \Bigg\}$$

and construct the corresponding optimal solution C^* by backtracking.

We prove now, that P-Polyn is a pseudo-polynomial time algorithm for the problem $Pm|NMVstep,\ d_{jq}=d_q, k\ \mathit{fixed}|\sum v_j(C_j)$.

Property 1. *The problem $Pm|NMVstep,\ d_{jq}=d_q, k\ \mathit{fixed}|\sum v_j(C_j)$can be solved optimally in pseudo-polynomial time $O(n\log nkm(\prod_{q=1}^{k-1}(\min\{T_n,d_q\})^m))$ by algorithm P-Polyn, where $T_n=\sum_{j=1}^n p_j$.*

Proof. Similarly as in Janiak and Krysiak (2007), the optimality of algorithm P-Polyn can be justified showing that a partial schedule – selected by this algorithm for expansion to a complete solution – can be extended by non-scheduled jobs in the same way as an optimal partial schedule, leading the same final state with non-worse value of the function $W_j(P_1^1, \dots, P_{k-1}^m)$.

One more observation is that our algorithm – similarly as the one presented in Janiak and Krysiak (2007) – checks (in Step 2) all possible choices of schedule for each job, thus obtaining all possible values of the criterion. However, the difference is that more constraints have to be complied in order to ensure feasibility of the final solution, because of non-monotonic job value functions. These additional constraints concern starting times of the scheduled jobs – all jobs have to fit to intervals they are assigned to. This results in possibility of processor idle times in final schedules (in particular in the optimal one). The compliance of the above constraints is achieved by assigning jobs from the end to the beginning of the interval, and by assigning these jobs in a such way that they are arranged in non-decreasing order of their processing times.

Therefore, our algorithm checks all possible values of the criterion and the partial schedules considered by this algorithm are always feasible and they lead to complete solutions, in which there is a one with the criterion value non-worse than the optimal one.

The computational complexity of this algorithm is determined by Step 2. We have n jobs and for each job we have $\prod_{q=1}^{k-1}(\min\{T_n, d_q\})^m$ values of P_q^i. For each value inequalities (2)–(4) must be checked and $W_j(P_1^1, \dots, P_{k-1}^m)$ must be calculated, what can be done in $O(km)$ time. Moreover, each time the position of job J_j in the set X_q^i must be determined, what can be done in $O(\log n)$ time. Thus, the computational complexity of the algorithm is $O(n \log nkm(\prod_{q=1}^{k-1}(\min\{T_n, d_q\})^m))$. □

Therefore, algorithm P-Polyn is pseudo-polynomial time for a given, constant number of processors and constant number of moments of changes of job values. It is also exponential time if the number of processors and/or number of moments of changes of job values is a variable.

6. Heuristic Algorithms

The problem $Pm|NMVstep|\sum v_j(C_j)$is NP-hard, so it is hardly possible to construct a fast (polynomial time) exact algorithm solving it. Therefore, we provide a set of heuristic algorithms, that in fact do not have to provide an optimal solution, but are fast enough.

The constructed heuristics (PH1–PH5) are based on the same scheme (algorithm PHi) and differ in the procedure of sorting the jobs (Step 2). The main idea of the scheme PHi is to make for each job a list of intervals sorted non-increasingly according to values corresponding to them and then to schedule jobs on processors in a proper order. The interval is a period between moments of changes of the job value ($(d_{jq-1}, d_{jq}]$, where $q = 1, \dots, k$, $d_{j0} = 0$, $d_{jk} = \infty$). We try to schedule each job so that the job ends in a period of time when it has a largest value. If it is impossible in one processor, we try in another. If it is impossible in all processors, we consider the next interval (with a smaller value). A formal description of the algorithm is given as follows:

Algorithm 2 PHi

```
Step 1: For each job make a list of intervals sorted non-increasingly according to the values corresponding to them.
Step 2: If $i = 1$, sort jobs non-increasingly according to $\max_{1 \leqslant q \leqslant k}\{w_{jq}\}$.
    If $i = 2$, sort jobs non-increasingly according to $\max_{1 \leqslant q \leqslant k}\{w_{jq}\}/p_j$.
    If $i = 3$, sort jobs non-decreasingly according to $p_j$.
    If $i = 4$, sort jobs non-increasingly according to the mean calculated as $\frac{1}{k}\sum_{i=1}^{k} w_{ji}$.
    If $i = 5$, sort jobs non-increasingly according to the mean calculated as $\frac{1}{k-1}\sum_{i=1}^{k-1} w_{ji}(d_{ji} - d_{ji-1})$. Assume $d_{j0} = 0$.
Step 3: If there is at least one job on the list, take out the first job of the list. Else finish.
Step 4: Take out the first interval for the current job of the list. Set $m\_numb := 0$.
Step 5: Set $m\_numb := m\_numb + 1$. If $m\_numb > m$ go to Step 4.
Step 6: If job can be assigned to processor $M_{m\_numb}$ and end in the considered interval, assign the job to $M_{m\_numb}$, as the job starting time set the end of the interval minus the job processing time or the starting time of the first job already assigned to this interval minus the job processing time, depending on what is earlier and go to Step 3.
    Else go to Step 5.
```

The sorting criteria (Step 2) are based on static priority rules. In the first one jobs are sorted non-increasingly according to the maximal job value. In algorithm PH3 processing times of jobs are considered. It is based on the assumption that it is easier to find a suitable processor idle time in the proper interval for a job with shorter processing time and simultaneously – after assignment the job to a processor – more time of the processor leaves for the next jobs. Algorithm PH2 is a combination of PH1 and PH3 – jobs are sorted according to the ratio of the maximal value to the processing time. In algorithms PH4 and PH5 jobs are sorted according to the average job value and weighted average job value, respectively.

A computational complexity of Step 1 is $O(nk \log k)$ since for all jobs we have to sort k intervals. A computational complexity of Step 2 depends on algorithm – it is $O(n \log n)$ for PH3, $O(\log k \cdot n \log n)$ for PH1 and PH2, and $O(k \cdot n \log n)$ for PH4 and PH5. Analyzing our implementation of Steps 3–6 in purely formal way, we have to check in the worst situation k intervals on m processors for each job and in each interval check n positions (before and after each of the jobs already scheduled on the processor). Therefore, the formal (pessimistic) computational complexity of Steps 3–6 is $O(n^2km)$ and the computational complexity of the whole scheme PHi is equal to $O(nk \cdot (\log k + \log n) + n^2km)$.

However, notice that, for a given job, the checking process in Steps 4–6 is interrupted immediately after finding the proper interval (i.e., the first interval within which the job will be finished). Moreover, for typical instances the proper interval is usually found in one of the first iterations (or – on the other hand – the pessimistic situations are rather artificial, e.g., all intervals on all processors will be checked if the job processing time is larger than all moments of changes of the job value d_{jq}). Therefore, the computational complexity of Steps 3–6 for usual instances can be reduced to $O(n^2)$ or even to $O(n^2/m)$, what implies $O(nk \cdot (\log k + \log n) + n^2)$ and $O(nk \cdot (\log k + \log n) + n^2/m)$ computational complexity for PHi, respectively. The detailed description concerning this reduction is presented in Appendix A.

7. Numerical Experiments

In order to evaluate the quality of the provided heuristic algorithms, the numerical experiments were performed. Algorithms were tested for five numbers of jobs: $n = 9, 50, 100, 250, 500$, five different numbers of processors $m = 2, 3, 5, 7, 10$, and three different numbers of intervals for each job: $k = 2, 10, 20$. What is more, experiments were performed in three series, in each series values were randomly generated from the uniform distribution on different intervals:

- set 1 (small values w_{jq}):
  $p_j \in (0, 90)$, $w_{jq} \in [1, 10)$, $\Delta d_{jq} = d_{jq} - d_{jq-1} \in [100, 200)$,
- set 2 (large values w_{jq}):
  $p_j \in (0, 90)$, $w_{jq} \in [1, 100)$, $\Delta d_{jq} \in [100, 200)$,
- set 3 (large processing times p_j and values w_{jq}):
  $p_j \in (0, 160)$, $w_{jq} \in [1, 100)$, $\Delta d_{jq} \in [100, 200)$.

For each set, 500 instances of problem were generated for each combination of k, m and n. It gives 112 500 instances in total.

Since the number of possible solutions for each instance is enormous (it is pseudo-polynomial for each permutation of the jobs), it is hardly possible to compare results obtained by heuristic algorithms to the optimal one. Therefore, obtained results are compared to the best found solution. For each instance i, the criterion value of the best solution provided by the algorithms PH1–PH5 is denoted by A_{BEST} and the criterion value of the solution provided by the algorithm j ($j \in \{\text{PH1, PH2, PH3, PH4, PH5}\}$) by A_j. Then the performance ratio $\rho_{ij} = \frac{A_{BEST} - A_j}{A_{BEST}} \cdot 100\%$ was calculated. The mean values for 500 instances are presented in Tables 1–3 (only for 2, 5 and 10 processors, since the results for 3 and 7 processors do not add anything important to the analysis).

As regards the computation times of the considered algorithms, we have made similar observations for all algorithms (PH1–PH5) and for all sets (Set 1–Set 3). Therefore, we present in Table 5 average computation times only for algorithm PH1 and Set 1 and based on these results we describe the computation times of all considered algorithms in all performed tests.

In order to show the difference in computation times between the constructed pseudo-polynomial time algorithm P-Polyn (Algorithm 1) and the heuristic algorithms PH1–PH5, several tests for small instances with common for all jobs moments of change of

Table 1
Average performance ratios ρ_{ij} [%] of the considered algorithms for $m = 2$.

n	$k = 2$					$k = 10$					$k = 20$				
	PH1	PH2	PH3	PH4	PH5	PH1	PH2	PH3	PH4	PH5	PH1	PH2	PH3	PH4	PH5
Set 1: $p_j \in (0, 90)$, $w_{jq} \in [1, 10)$, $\Delta d_{jq} \in [100, 200)$															
9	**0.00**	1.48	3.10	0.58	0.58	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
50	2.13	**1.16**	2.42	5.62	1.38	**0.33**	1.80	1.54	1.01	1.28	**0.07**	**0.07**	0.08	0.12	0.08
100	2.11	**0.22**	0.89	5.62	2.51	**0.75**	1.41	1.80	4.50	2.03	**0.07**	0.76	0.87	0.72	0.65
250	2.49	**0.02**	0.72	4.74	2.68	5.63	**0.15**	0.38	9.18	6.09	2.88	**0.16**	0.30	5.71	3.72
500	2.37	**0.00**	0.43	4.01	2.95	7.93	**0.08**	0.25	10.55	8.27	7.49	0.14	**0.11**	10.05	7.79
Set 2: $p_j \in (0, 90)$, $w_{jq} \in [1, 100)$, $\Delta d_{jq} \in [100, 200)$															
9	0.15	0.61	1.61	1.13	**0.12**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
50	1.66	**1.09**	2.23	6.26	2.23	**0.14**	0.80	0.94	0.61	0.39	**0.02**	0.03	0.03	0.05	0.03
100	2.32	**0.27**	1.05	5.42	2.44	1.23	**0.90**	1.57	3.97	1.86	**0.07**	0.72	0.68	0.42	0.48
250	2.70	**0.03**	0.70	5.17	3.27	5.56	**0.11**	0.35	9.46	6.52	2.31	**0.16**	0.29	5.11	2.86
500	2.78	**0.01**	0.40	4.34	3.27	8.75	**0.08**	0.30	11.32	8.95	7.30	**0.13**	0.14	9.44	7.18
Set 3: $p_j \in (0, 160)$, $w_{jq} \in [1, 100)$, $\Delta d_{jq} \in [100, 200)$															
9	1.13	2.19	3.40	3.33	**0.99**	0.10	0.01	0.01	0.23	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
50	1.77	**1.01**	2.03	4.69	1.18	**0.75**	2.44	2.91	3.94	2.03	**0.16**	0.98	1.19	0.65	0.58
100	1.95	**0.15**	0.90	4.04	1.73	3.52	**0.52**	0.64	7.69	4.24	1.11	2.16	2.13	2.38	**1.00**
250	2.06	**0.04**	0.55	3.67	2.32	6.32	**0.18**	0.34	9.16	7.06	5.05	**0.21**	**0.21**	7.98	5.51
500	1.66	**0.01**	0.34	2.72	2.03	7.70	**0.08**	0.30	9.50	7.40	7.99	**0.07**	0.27	10.09	8.37

job value were performed. Moreover, the performance ratio of the heuristic algorithms $\rho^*_{ij} = \frac{A_{OPT} - A_j}{A_{OPT}} \cdot 100\%$, where A_{OPT} is an optimal value obtained by P-Polyn, was calculated. The results are presented in Table 4.

As it can be seen in Tables 1–3, algorithm PH1 is usually the best for small instances (for $n = 9, 50$ and sometimes 100 jobs), especially for large numbers of moments of changes of job values ($k = 10, 20$). Algorithm PH2 is the best for large instances (usually for these cases, for which PH1 is not the best). Moreover, for several cases of small instances, for which PH1 is not the best, the most accurate is usually algorithm PH5. Even if other algorithms give better solutions for some specific instances, the average performance ratio for the best algorithm for the given set of parameters is usually close to the one for PH1, PH2 or PH5. For example, for the case with $m = 10$ processors, Set 3, $k = 20$ and $n = 500$ jobs (see Table 3), the best is not PH2, but PH3 with $\rho_{ij} = 0.08$. However, for PH2 $\rho_{ij} = 0.12$. This might be due to the fact that lengths of intervals and processing times are independent of k and n, therefore the sum of processing times per interval increases with the size of the instance and decreases with the number of intervals. When the ratio of the sum of processing times to the number of intervals is smaller, it is easier to assign jobs to the intervals when they have the largest value. In such cases, the maximal or average job value is sufficient criterion (algorithms PH1, PH5). This conclusion is also confirmed in Table 4, where the results obtained by the considered heuristic algorithms are compared with the optimal solutions for small ($n = 9, 20$ and 30 jobs) and specific instances (i.e., with common moments of change of job value). The best algorithms for these cases are always PH1 and PH5. Algorithm PH2 is better for larger instances, because it rewards jobs with the large maximal value and a small processing time. It is easier to

Table 2
Average performance ratios ρ_{ij} [%] of the considered algorithms for $m = 5$.

n	$k=2$					$k=10$					$k=20$				
	PH1	PH2	PH3	PH4	PH5	PH1	PH2	PH3	PH4	PH5	PH1	PH2	PH3	PH4	PH5
Set 1: $p_j \in (0, 90)$, $w_{jq} \in [1, 10)$, $\Delta d_{jq} \in [100, 200)$															
9	**0.11**	2.23	2.74	1.46	1.10	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
50	1.72	**1.15**	2.13	5.92	2.18	**0.39**	1.62	1.36	1.34	1.25	**0.04**	0.08	0.08	0.06	0.07
100	1.41	**0.44**	0.85	4.92	2.12	1.29	**0.75**	1.05	5.00	2.53	**0.07**	0.94	0.94	0.61	0.52
250	2.35	**0.03**	0.53	4.78	2.81	6.33	**0.16**	0.39	9.44	6.23	2.67	**0.08**	0.23	5.37	3.59
500	2.29	**0.01**	0.41	3.94	2.61	7.47	**0.11**	0.18	10.10	8.15	7.04	**0.06**	0.15	9.78	7.51
Set 2: $p_j \in (0, 90)$, $w_{jq} \in [1, 100)$, $\Delta d_{jq} \in [100, 200)$															
9	0.45	0.88	0.93	0.75	**0.38**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.02	**0.00**	**0.00**
50	1.83	**0.95**	2.63	5.97	1.41	**0.22**	1.30	1.47	1.16	0.83	**0.03**	0.07	0.07	0.06	0.06
100	1.87	**0.34**	0.90	5.08	1.97	**0.82**	1.01	1.16	3.98	2.17	**0.09**	0.68	0.65	0.44	0.27
250	2.29	**0.05**	0.50	5.06	2.97	5.78	**0.08**	0.48	9.89	6.78	2.25	**0.20**	0.29	5.18	2.86
500	2.51	**0.00**	0.48	4.02	2.93	8.10	**0.04**	0.37	10.98	8.78	7.36	**0.11**	0.19	9.83	7.43
Set 3: $p_j \in (0, 160)$, $w_{jq} \in [1, 100)$, $\Delta d_{jq} \in [100, 200)$															
9	0.55	1.98	3.94	2.02	**0.30**	0.02	**0.00**	**0.00**	**0.00**	0.02	**0.00**	0.02	0.01	**0.00**	**0.00**
50	1.48	**0.77**	1.85	5.06	1.69	**0.76**	2.55	2.88	3.96	2.23	**0.16**	0.93	0.82	0.65	0.62
100	1.67	**0.29**	0.88	4.17	2.24	2.94	**0.42**	0.80	6.73	3.84	**0.90**	1.77	2.11	3.66	1.51
250	1.80	**0.02**	0.42	3.62	2.23	6.10	**0.14**	0.29	8.98	6.93	5.04	0.29	**0.14**	7.48	5.46
500	1.75	**0.01**	0.34	2.92	2.26	8.11	**0.07**	0.28	9.91	7.60	8.09	**0.08**	0.22	10.22	8.28

Table 3
Average performance ratios ρ_{ij} [%] of the considered algorithms for $m = 10$.

n	$k=2$					$k=10$					$k=20$				
	PH1	PH2	PH3	PH4	PH5	PH1	PH2	PH3	PH4	PH5	PH1	PH2	PH3	PH4	PH5
Set 1: $p_j \in (0, 90)$, $w_{jq} \in [1, 10)$, $\Delta d_{jq} \in [100, 200)$															
9	0.67	1.36	1.50	1.24	**0.05**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
50	1.89	**1.00**	1.96	5.68	1.47	**0.34**	1.37	1.16	0.76	0.90	**0.07**	0.08	0.09	0.18	0.13
100	1.31	**0.43**	1.44	5.20	2.06	1.18	**1.15**	1.45	4.31	1.98	**0.09**	0.90	1.03	0.58	0.64
250	2.28	**0.03**	0.66	4.56	2.68	6.22	**0.13**	0.39	9.84	6.72	2.93	**0.22**	0.31	5.72	3.58
500	2.34	**0.00**	0.46	3.73	2.56	7.64	**0.07**	0.36	10.39	8.06	6.96	**0.09**	0.16	9.43	7.18
Set 2: $p_j \in (0, 90)$, $w_{jq} \in [1, 100)$, $\Delta d_{jq} \in [100, 200)$															
9	0.49	**0.40**	0.89	0.65	1.08	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
50	1.74	**0.88**	1.96	4.73	2.38	**0.17**	1.19	1.52	0.82	0.52	**0.04**	0.10	0.06	**0.04**	0.05
100	1.85	**0.36**	1.20	5.11	2.29	**1.09**	**1.09**	1.22	4.20	2.42	**0.07**	0.77	0.83	0.46	0.46
250	2.41	**0.03**	0.65	4.61	3.12	5.67	**0.14**	0.40	10.14	7.03	2.61	**0.23**	0.31	5.43	3.12
500	2.27	**0.03**	0.33	3.95	2.86	8.15	**0.04**	0.29	11.11	8.57	7.30	**0.12**	0.21	10.10	7.59
Set 3: $p_j \in (0, 160)$, $w_{jq} \in [1, 100)$, $\Delta d_{jq} \in [100, 200)$															
9	**0.74**	0.84	2.76	**0.74**	0.97	0.02	**0.00**	**0.00**	0.17	0.17	**0.00**	0.02	0.02	**0.00**	**0.00**
50	1.80	**0.93**	1.92	4.46	2.15	**1.80**	3.10	3.63	4.34	2.08	**0.24**	1.02	1.10	0.44	0.54
100	2.16	**0.18**	0.94	3.97	2.15	2.86	**0.77**	0.96	6.45	3.91	**0.70**	2.06	2.33	2.63	1.16
250	2.04	**0.05**	0.48	3.35	2.30	7.21	**0.11**	0.37	9.09	6.86	5.10	**0.16**	0.22	7.60	5.35
500	1.78	**0.01**	0.28	2.89	2.14	7.78	**0.10**	0.24	9.46	7.94	7.70	0.12	**0.08**	9.95	7.70

Table 4
Average performance ratios ρ^*_{ij} [%] and computation times t [s] of the considered algorithms for $m = 2$ and $k = 2$.

n	ρ^*_{ij}					t					
	PH1	PH2	PH3	PH4	PH5	P-Polyn	PH1	PH2	PH3	PH4	PH5
Set 1: $p_j \in (0, 90)$, $w_{jq} \in [1, 10)$, $\Delta d_{jq} \in [100, 200)$											
9	**0.26**	0.77	1.58	0.59	**0.26**	12.006	0.000	0.001	0.000	0.001	0.001
20	**3.25**	3.73	4.61	7.02	**3.25**	102.100	0.002	0.001	0.002	0.002	0.001
30	**3.51**	3.64	4.70	9.42	**3.51**	328.103	0.005	0.003	0.002	0.003	0.006

Table 5
Exemplary average computation times [s] – alg. PH1, Set 1.

m	k														
	2	10	20	2	10	20	2	10	20	2	10	20	2	10	20
2	0.00	0.00	0.00	0.01	0.01	0.01	0.04	0.06	0.03	0.32	0.61	0.69	1.14	3.01	4.06
3	0.00	0.00	0.00	0.01	0.00	0.01	0.04	0.03	0.02	0.29	0.51	0.41	1.25	2.74	3.36
5	0.00	0.00	0.00	0.01	0.00	0.01	0.04	0.01	0.02	0.28	0.31	0.12	1.31	2.20	2.10
7	0.00	0.00	0.00	0.01	0.00	0.01	0.04	0.02	0.02	0.27	0.18	0.09	1.29	1.81	1.11
10	0.00	0.00	0.00	0.01	0.00	0.01	0.03	0.02	0.02	0.25	0.10	0.09	0.99	1.25	0.44
	$n = 9$			$n = 50$			$n = 100$			$n = 250$			$n = 500$		

assign such jobs to the optimal interval, moreover they save the processor time for next jobs. Note that PH3, where job processing times are considered, is also good for larger instances and its performance ratio is close to the one for PH2.

Since the computational complexity of all constructed heuristic algorithms is similar and we tested these algorithms mostly based on rather usual instances (number of processors less than number of jobs, job processing times less than intervals $(d_{jq-1}, d_{jq}]$, etc.), their computation times are also similar. Thus, the conclusions drawn for PH1 and Set 1 (based on Table 5) can be applied to all algorithms and all sets. First of all, let us notice that the computation times depend on the number of processors, m, only for large numbers of moments of changes of job values ($k \geqslant 10$) and large numbers of jobs ($n \geqslant 250$). What is more, this dependency is decreasing, which confirms the reduced computational complexity of the algorithms PH1–PH5 (see the appendix for justification of such phenomena). On the other hand, the computation times increase slightly with the increase of k if the number of jobs is large ($n \geqslant 250$) and the number of processors is small ($m = 2$ or 3), which also confirms the reduced computational complexity of the analyzed algorithms. Obviously, the computation times increase significantly with the increase of the number of jobs, since the computational complexity of the algorithms depends squarely on the number of jobs. Therefore, the described numerical experiments show that for typical instances the considered algorithms PH1–PH5 perform according to the reduced computational complexity.

Finally, Table 4 shows that the heuristic algorithms are faster by several orders of magnitude than the constructed pseudo-polynomial time algorithm. The difference for more general problem with different for all jobs moments of change of job value might be even larger.

8. Conclusions

In this paper, a scheduling problem on parallel processors with job values characterized by non-monotonic stepwise functions was addressed. We showed that the problem is interesting from practical point of view by giving examples of real-life systems which can be modelled in such a way. We proved that the problem is NP-hard in the ordinary sense and we provided a pseudo-polynomial time algorithm solving its special case with identical moments of job value changes. Since the computational complexity of the algorithm is quite high and it cannot solve the general version of the problem, we also constructed and tested a number of heuristic algorithms. The obtained results of the tests confirm that three of these algorithms perform clearly better than the others.

The aim of further research is to construct even more exact and faster heuristic (or maybe approximation) algorithms and determine computational complexity of a more general case of the problem with variable number of processors.

Appendix A: Estimation of the Computational Complexity of the Scheme PHi for Usual Instances

For the instances with job processing times less than the intervals $(d_{jq-1}, d_{jq}]$ we can assume that for the first m jobs the proper intervals will be found in $1, 2, \dots, m$ iterations, respectively (the first interval is always empty in this situation, but in the worst case the jobs will be placed on different processors). Thus, the time needed to find these intervals is $1 + 2 + \cdots + m = O(m^2)$. Similarly, for the second portion of m jobs, the proper intervals will be found at most for the second time. Thus, the time needed to find these intervals is $O(2m^2)$. We have $\lceil n/m \rceil$ such portions of jobs. Therefore, the time needed to find the intervals for all the jobs is equal to:

$$O(m^2) \cdot \bigl(1 + 2 + \cdots + \lceil n/m \rceil\bigr) = O\bigl(m^2 \cdot \bigl(n^2/m^2\bigr)\bigr) = O\bigl(n^2\bigr).$$

In an optimistic situation we can assume that the best intervals for different jobs will be in different places on the time axis and we can assume that for the first m jobs the proper interval will be found in 1 iteration, for the second portion of m jobs it will be 2 iterations, etc. Thus, we obtain:

$$m \cdot \bigl(1 + 2 + \cdots + \lceil n/m \rceil\bigr) = O\bigl(m \cdot \bigl(n^2/m^2\bigr)\bigr) = O\bigl(n^2/m\bigr).$$

Acknowledgements. The research was partially supported by National Science Center, decision No. DEC-2011/03/B/ST6/00364.

References

Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Methods*, 5, 287–326.

Huang, X., Wang, M.-Z. (2012). Single machine group scheduling with time and position dependent processing times. *Optimization Letters*. DOI:10.1007/s11590-012-0535-z.

Huang, X., Li, G., Huo, Y., Ji, P. (2012). Single machine scheduling with general time-dependent deterioration, position-dependent learning and past-sequence-dependent setup times. *Optimization Letters*. DOI:10.1007/s11590-012-0522-4.

Janiak, A., Krysiak, T. (2007). Single processor scheduling with job values depending on their completion times. *Journal of Scheduling*, 10, 129–138.

Janiak, A., Krysiak, T. (2012). Scheduling jobs with values dependent on their completion times. *International Journal of Production Economics*, 135, 231–241.

Janiak, A., Krysiak, T., Pappis, C.P., Voutsinas, T.G. (2009). A scheduling problem with job values given as a power function of their completion times. *European Journal of Operational Research*, 193(3), 836–848.

Janiak, A., Kołodka, P., Krysiak, T. (2010). Scheduling jobs with changeable job values and different release dates – solution algorithms. *Przeglad Elektrotechniczny*, 9, 91–96 (in Polish).

Pappis, C.P., Rachaniotis, N.P. (2010). Scheduling in a multi-processor environment with deteriorating job processing times and decreasing values: the case of forest fires. *Journal of Heuristics*, 16(4), 617–632.

Rachaniotis, N.P., Pappis, C.P. (2006). Scheduling fire-fighting tasks using the concept of "deteriorating jobs". *Canadian Journal of Forest Research*, 36, 652–658.

Raut, S., Swami, S., Gupta, J.N. (2008a). Scheduling a capacitated single machine with time deteriorating job values. *International Journal of Production Economics*, 114(2), 769–780. Special Section on Logistics Management in Fashion Retail Supply Chains.

Raut, S., Swami, S., Gupta, J.N. (2008b). Single machine scheduling with time deteriorating job values. *Journal of the Operational Research Society*, 59(2), 105–118.

Sadowski, B. (2007). *Biological Mechanisms of Behavior of Humans and Animals*. Polish Scientific Publishers PWN, Warsaw (in Polish).

Voutsinas, T.G., Pappis, C.P. (2002). Scheduling jobs with values exponentially deteriorating over time. *International Journal of Production Economics*, 79(3), 163–169.

Yang W.-H. (2009). Scheduling jobs on a single machine to maximize the total revenue of jobs. *Computers & Operations Research*, 36(2), 565–583.

A. Janiak received the MEng and PhD degrees from the Wrocław University of Technology, Wrocław, Poland, in 1972 and 1977, respectively, and the DrSc degree from the Warsaw University of Technology, Warsaw, Poland, in 1992.

He received the Professor title in 1999. He was invited as a Visiting Professor to universities in Australia, Canada, Germany, Hong Kong, Israel, New Zealand, Thailand, China, Spain, US, Greece, Great Britain, Holland, France. He is currently a Full Professor in computer science and operations research of industrial engineering areas with the Institute of Computer Engineering, Control and Robotics, Wrocław University of Technology, where he is the Head of the Department of Artificial Intelligence and Algorithms Design. He is the author of 5 books and more than 270 papers in edited books, international journals, conference proceedings (including over 80 publications in journals from ISI Master Journal List). His papers were cited over 1600 times (according to data from ISI journal database), index $h = 23$. His research interests include sequencing and scheduling problems with classical and generalized models of operations in computer and manufacturing systems, resource allocation problems, complexity theory, and theory of algorithms (physical design automation of very large scale integration).

Prof. Janiak is an Associate Editor for IEEE Trans. SMCA, Int. J. of Applied Mathematics and Computer Science, Decision Making in Manufacturing and Services, and Recent Patents on Computer Science, and the book series "Computational Intelligence and Its Applications". He is a member of the Polish Academy of Sciences, the Vice-President of the Computer Science Committee of the Polish Academy of Sciences, and the Head of the Department of Computer Methods in Science of the Polish Research Council. He is an expert of the State Accreditation Committee for the quality of higher education. He has served on the program committees for several international conferences on operations research and is a regular reviewer for a number of prestigious journals and conferences.

T. Krysiak received the MEng and PhD degrees from the Wrocław University of Technology, Wrocław, Poland, in 2001 and 2005, respectively. He is currently working as an assistant professor in the Institute of Computer Engineering, Control and Robotics, Wrocław University of Technology.

He is the author of about 40 papers in scientific journals, books, and conference proceedings (including 4 publications in journals from ISI Master Journal List). His main research interests are scheduling and resource allocation problems with variable parameters (job values, time and resource dependent jobs, etc.).

R. Trela received the MEng degree in control engineering and robotics from the Wrocław University of Technology, Wroclaw, Poland in 2010.

He is currently a PhD student in the Institute of Computer Engineering, Control and Robotics at the Wroclaw University of Technology. His main research interests are scheduling problems with variable parameters (learning and aging effects, job values, etc.).

Darbų tvarkaraščių uždavinys su nemonotoninėmis laiptinėmis vertėmis

Adam JANIAK, Tomasz KRYSIAK, Radosław TRELA

Straipsnyje nagrinėjamas lygiagrečiųjų procesorių tvarkaraščio uždavinys su kintamomis darbų vertėmis. Skirtingai nuo kitų straipsnių šioje srityje, daroma prielaida, kad darbai yra charakterizuojami nemonotoninėmis laiptinėmis darbų baigimo laiko funkcijomis (anksčiau tik nedidėjančios funkcijos buvo naudojamos). Pateikiami pavyzdžiai realių sistemų, kurios gali būti taip modeliuojamos, demonstruoja, kad toks uždavinys yra įdomus praktikoje. Parodyta, kad uždavinys yra NP-sudėtingas, tačiau specialiam uždavinio atvejui yra sudarytas pseudopolinominis algoritmas. Pateikta keletas euristinių algoritmų ir jie eksperimentiškai ištirti.