J Sched (2007) 10:129–138
DOI 10.1007/s10951-006-0004-6

Single processor scheduling with job values depending on their completion times

Adam Janiak · Tomasz Krysiak

Published online: 2 February 2007

Abstract The paper deals with a single processor scheduling problem in which the sum of values of all jobs is maximized. The value of a job is characterized by a stepwise nonincreasing function with one or more moments at which the changes of job value occur. Establishing an order of processing of datagrams which are sent by router is a practical example of application of such problems. We prove that the special case of our problem, with a single moment of change of job values, is equivalent to the well-known, NP-hard in the ordinary sense, problem of minimizing weighted number of late jobs. Next, we show that, based on this equivalence, the existing algorithms for solving the latter problem can be adopted to solve special cases of our problem. Additionally, we construct a pseudopolynomial time algorithm based on the dynamic programming method, for the case with arbitrary number of common moments of job value changes. At the end of the paper, we generalize this algorithm to the corresponding case with parallel processors. Thus, we show that these two problems are also NP-hard in the ordinary sense. Moreover, we construct exact polynomial time algorithms for two further special cases of our problem. Finally, in order to solve the general version of the problem, we construct and experimentally test a number of heuristic algorithms.

Keywords Single processor · Job value · NP-hardness · Polynomial time · Heuristic

A. Janiak (✉) · T. Krysiak
Institute of Engineering Cybernetics, Wroclaw University of Technology, Janiszewskiego 11/17, 50-372 Wrocław, Poland
e-mail: Adam.Janiak@pwr.wroc.pl

1 Introduction

The first book concerning the important results obtained in scheduling theory was edited by E. Coffman (1976). Since then, rapid development of scheduling theory permits us to construct better algorithms which solve a lot of the real-life problems. Nevertheless, many practical problems remain still unsolved. Hence, new scheduling problems are formulated and solved continuously. The problem of job values dependent on their completion times are some of such new areas in scheduling theory. Voutsinas and Pappis (2002) formulated a problem of maximizing the sum of job values for the first time. They used an exponential model of the job value. In some polynomially solvable cases, heuristic algorithm for the general version of the problem and case study was presented. Then, a branch and bound algorithm and some new heuristic algorithms for the same problem were constructed and tested by Bachman et al. (2002). In the present paper, a new model of job value is considered, namely a stepwise one. In this model, the job values are described by a nonincreasing stepwise function with one or more moments at which the changes of job value occur.

Establishing an order of processing of datagrams which are sent by router in IP protocol is an application example of such problems. Its precise description is given as follows. TCP/IP is the most popular set of computer network protocols and IP protocol is one of its main components. If the data (grouped into the basic units called *datagrams*) are sent from a source computer to a destination computer in another local network, then the appropriate datagrams have to pass some routers. The routers connect the local networks and determine routes, by which the datagrams will be sent farther. If any datagram arrives to a router then it is placed in its input buffer where it waits for processing (if the router is not overloaded the datagram is processed immediately). Each datagram consists of a header and a data part; the header consists of fields. One of them, called time to live (TTL), determines how long the datagram can be present in a network. A value of TTL is set by a sender for each datagram and in each router, it is decreased proportionally to the time spent by the datagram in the router (i.e., the waiting time in the input buffer and the processing time). However, it has to be decreased by at least 1 unit (even if this time is shorter than 1 s). If the TTL value reaches zero, the datagram is abandoned from the network. This is done to prevent the circulation of datagrams in a network ad infinitum if they cannot achieve their destinations.

In the scheduling model, the processing of datagram by the router may be viewed as a job. Decreasing of the TTL value during the processing of datagram by the router may be modeled by a nonincreasing stepwise function. The problem is to find such an order of processing of datagrams in the router so that the sum of TTL values of these datagrams (obtained after this processing) is maximum.

Other examples deal with a process of distributing and selling the grocery articles with a short sell-by date and establishing an order of picking different kinds of fruits in an orchard (Janiak and Krysiak, 2003).

The remaining part of the paper is organized as follows. In Section 2, we give a precise formulation of the problem under consideration. In Section 3, we show that our problem is NP-hard even for the case with a single moment of job value change. In this section, we also prove some polynomially solvable cases. In Section 4, we propose a pseudopolynomial time algorithm, which was constructed to solve another special case, with arbitrary number of moments of job value changes common for all jobs. In Section 5, we present and experimentally compare some heuristic algorithms. Some generalizations of the results to the parallel processor cases and concluding remarks are given in Section 6.

2 Problem formulation

Consider a single processor and a set $J = \{1, \ldots, n\}$ of n independent and non-preemptive jobs immediately available for processing at time 0. Each job $i \in J$ is characterized by its processing time $p_i > 0$, its value $w_i(C_i)$ calculated at the completion time C_i and the moments $d_{ij} > 0$, $j = 1, \ldots, k-1$ at which changes of value of job i occur. The model of job value is given by nonincreasing stepwise function defined as follows:

$$w_i(C_i) = \begin{cases} w_{i1}, & 0 < C_i \le d_{i1} \\ w_{i2}, & d_{i1} < C_i \le d_{i2} \\ \vdots & \\ w_{ik}, & d_{ik-1} < C_i \end{cases},$$

where $w_{i1} > w_{i2} > \cdots > w_{ik}$. The objective is to find such a sequence of jobs on the processor (i.e., a schedule $\pi = \langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$, where $\pi(i)$ denotes the index of a job occupying ith position in the schedule π, $i = 1, 2, \ldots, n$) that *maximizes* the sum of job values:

$$\sum_{i=1}^{n} w_{\pi(i)}\big(C_{\pi(i)}\big) \Rightarrow \max.$$

Using the three-field notation $\alpha|\beta|\gamma$ for scheduling problems (Graham et al., 1979), the problem considered in this paper is given by

$$1 \left| w_i(C_i) = \begin{cases} w_{i1}, & 0 < C_i \le d_{i1} \\ w_{i2}, & d_{i1} < C_i \le d_{i2} \\ \vdots & \\ w_{ik}, & d_{ik-1} < C_i \end{cases} \right| \sum w_i(C_i).$$

3 Computational complexity and polynomially solvable cases

In this section, it is proved that the problem under consideration is NP-hard, since its special case with a single moment of change of job values is equivalent to the well-known NP-hard $1\|\sum w_i U_i$ problem. We derive this result with the aid of two lemmas given below.

Lemma 1. *The following two problems*

$$1 \left| w_i(C_i) = \begin{cases} w_{i1}, & 0 < C_i \le d_{i1} \\ w_{i2}, & d_{i1} < C_i \end{cases} \right| \sum w_i(C_i) \tag{1}$$

and

$$1 \left| w_i(C_i) = \begin{cases} w'_{i1} = w_{i1} - w_{i2}, & 0 < C_i \le d_{i1} \\ w'_{i2} = 0, & d_{i1} < C_i \end{cases} \right| \sum w_i(C_i) \tag{2}$$

are equivalent, i.e., in both problems the sequences of jobs are equal and their criterion values differ by a constant.

Proof: If, for a given job sequence π for problem (1), the value $w_i(C_i)$ of each job i ($i = 1, \dots, n$) is decreased by w_{i2}, then the criterion value decreases by $V = \sum_{i=1}^{n} w_{i2}$. Since V is constant (1) can be reduced to (2). □

Lemma 2. *The following two problems* $1\big|w_i(C_i) = \left\{\begin{smallmatrix} w'_{i1}, & 0 < C_i \le d_{i1} \\ 0, & d_{i1} < C_i \end{smallmatrix}\right.\big|\sum w_i(C_i)$ *and* $1\|\sum \hat{w}_i U_i$ *are equivalent, i.e., in both problems the sequences of jobs are equal and their criterion values differ by a constant.*

Proof: At first, let us formulate the problem $1\|\sum \hat{w}_i U_i$: Given the single-processor problem of scheduling $\hat{n}$ jobs with processing times $\hat{p}_i$, weights $\hat{w}_i$, due dates $\hat{d}_i$ and unit penalties $U_i : U_i = 0$ if $C_i \le \hat{d}_i$ or $U_i = 1$ if $C_i > \hat{d}_i$, where C_i denotes the completion time of job i, $i = 1, \dots, \hat{n}$. The parameters $\hat{p}_i$, $\hat{w}_i$, $\hat{d}_i$ are nonnegative values, whereas $\hat{n}$ is a positive integer. The problem is to find a sequence of $\hat{n}$ jobs on a single processor such that the sum of the weighted late jobs is minimized.

Thus, given a sequence $\hat{\pi}$ in the problem $1\|\sum \hat{w}_i U_i$ (with parameters: $\hat{n}$, $\hat{p}_i$, $\hat{d}_i$, $\hat{w}_i$), we have

$$\sum_{i=1}^{\hat{n}} \hat{w}_{\hat{\pi}(i)} U_{\hat{\pi}(i)} \Rightarrow \min.$$

In our problem (with parameters: n, p_i, d_{i1}, w'_{i1} and π), where

$$\sum_{i=1}^{n} w_{\pi(i)}\big(C_{\pi(i)}\big) \Rightarrow \max,$$

assume that

$$\begin{aligned} &n = \hat{n}; \\ &\pi(i) = \hat{\pi}(i); \quad w'_{\pi(i)1} = \hat{w}_{\hat{\pi}(i)}; \\ &p_{\pi(i)} = \hat{p}_{\hat{\pi}(i)}; \quad d_{\pi(i)1} = \hat{d}_{\hat{\pi}(i)}; \quad i = 1, \dots, n = \hat{n}. \end{aligned}$$

From $U_i = \left\{\begin{smallmatrix} 0, & \hat{C}_i \le \hat{d}_i \\ 1, & \hat{C}_i > \hat{d}_i \end{smallmatrix}\right.$, $w_i(C_i) = \left\{\begin{smallmatrix} w'_{i1}, & C_i \le d_{i1} \\ 0, & C_i > d_{i1} \end{smallmatrix}\right.$, and $d_{\pi(i)1} = \hat{d}_{\hat{\pi}(i)}$ follows that $w_i(C_i) = 0$ if $U_i = 1$ and $w_i(C_i) = w'_{i1}$ if $U_i = 0$, $i = 1, \dots, n$. Thus, we have

$$w_{\pi(i)}\big(C_{\pi(i)}\big) = w'_{\pi(i)1}\big(1 - U_{\pi(i)}\big) = \hat{w}_{\hat{\pi}(i)}\big(1 - U_{\hat{\pi}(i)}\big).$$

Therefore,

$$\begin{aligned} \sum_{i=1}^{n} w_{\pi(i)}\big(C_{\pi(i)}\big) &= \sum_{i=1}^{n} \hat{w}_{\hat{\pi}(i)}\big(1 - U_{\hat{\pi}(i)}\big) \\ &= \sum_{i=1}^{\hat{n}} \hat{w}_{\hat{\pi}(i)} - \sum_{i=1}^{\hat{n}} \hat{w}_{\hat{\pi}(i)} U_{\hat{\pi}(i)} \Rightarrow \max. \end{aligned}$$

Since $\sum_{i=1}^{\hat{n}} \hat{w}_{\hat{\pi}(i)}$ is constant, thus maximization of $\sum_{i=1}^{n} w_{\pi(i)}(C_{\pi(i)})$ is equivalent to minimization of $\sum_{i=1}^{\hat{n}} \hat{w}_{\hat{\pi}(i)} U_{\hat{\pi}(i)}$ and solutions to both the problems are equal: $\pi = \hat{\pi}$. □

Therefore, the following conclusion follows directly from the lemmas given above.

Conclusion 1. The following two problems $1\big|w_i(C_i) = \left\{\begin{smallmatrix} w_{i1}, & 0 < C_i \le d_{i1} \\ w_{i2}, & d_{i1} < C_i \end{smallmatrix}\right.\big|\sum w_i(C_i)$ and $1\|\sum \hat{w}_i U_i$ are equivalent, i.e., in both problems the sequence of jobs are equal and their criterion values differ by a constant.

The computational complexity status of the problem with a single change of job values $1\big|w_i(C_i) = \left\{\begin{smallmatrix} w_{i1}, & 0 < C_i \le d_{i1} \\ w_{i2}, & d_{i1} < C_i \end{smallmatrix}\right.\big|\sum w_i(C_i)$ can be derived directly from Conclusion 1, based on the results obtained for the problem $1\|\sum \hat{w}_i U_i$. Moreover, the algorithms solving the latter problem and its special cases, can be adopted to solve our problem (1) and its corresponding special cases (in order to obtain the special cases, set $\hat{p}_i = p_i$, $\hat{d}_i = d_{i1}$, $\hat{w}_i = w_{i1} - w_{i2}$ for $i = 1, 2, \dots, n$). The results of the computational complexity and solution algorithms are summarized in Table 1. Additionally, in this table, we give the references to the papers in which the results dealing with the problem $1\|\sum \hat{w}_i U_i$ and its special cases can be found.

Now, we will consider two further special cases of our general problem, which can be solved optimally in polynomial time. In Property 1, the case in which the values of the parameters $w_{i1}, \dots, w_{ik}$ and $d_{i1}, \dots, d_{ik-1}$ are identical for all the jobs, is considered.

Property 1. *The optimal solution to the problem*

$$1\left|w_i(C_i) = \begin{cases} w_1, & 0 < C_i \le d_1 \\ w_2, & d_1 < C_i \le d_2 \\ \quad\vdots \\ w_k, & d_{k-1} < C_i \end{cases}\right|\sum w_i(C_i)$$

can be obtained by sequencing the jobs (in $O(n \log n)$ time) in a nondecreasing order of p_i (SPT rule).

Proof: Let π be an optimal permutation in which (contrary to the thesis of Property 1) $p_k > p_l$, where $k < l$ and π' be a permutation obtained by interchanging jobs kth and lth in π. Since $p_k > p_l$, the completion times of the jobs kth to $(l-1)$th will decrease after interchanging the jobs k and l in π'. Since $w_i(C_i)$ is nonincreasing and $w_1, w_2, \dots, w_k$ are independent of jobs, values of the jobs from kth to $(l-1)$th will not decrease, rather they can increase. Finally, since the completion times of the other jobs will not change, the interchanging jobs in the permutation π cannot decrease the criterion value, rather it can increase this value, which contradicts the optimality of π. □

Table 1 Computational complexity status and solution algorithms of the problems with single change of job values (where $\Delta w_i = w_{i1} - w_{i2}$)

Problem	Equivalent problem	Complexity/ solution alg.	Reference
$1\left\| w_i(C_i) = \begin{cases} w_{i1}, \ 0 < C_i \le d_{i1} \\ w_{i2}, \ d_{i1} < C_i \end{cases} \right\| \sum w_i(C_i)$	$1\|\|\sum \hat{w}_i U_i$	NP-hard, PP $O\left(n \sum_{i=1}^{n} \hat{p}_1\right)$, B&B, FPTAS	Karp, 1972; Lawler Moore, 1969; Villarreal Bulfin, 1983; Sahni, 1976; Gens Levner, 1978, 1981
$1\left\| w_i(C_i) = \begin{cases} w_1, \ 0 < C_i \le d_{i1} \\ w_2, \ d_{i1} < C_i \end{cases} \right\| \sum w_i(C_i)$	$1\|\|\sum U_i$	$O(n \log n)$	Moore, 1968
$1\left\| w_i(C_i) = \begin{cases} w_{i1}, \ 0 < C_i \le d_{i1} \\ w_{i2}, \ d_{i1} < C_i \end{cases}\right.$, $p_i < p_k \Rightarrow \Delta w_i \ge \Delta w_k \| \sum w_i(C_i)$	$1\|\hat{p}_i < \hat{p}_k \Rightarrow \hat{w}_i \ge \hat{w}_k\| \sum \hat{w}_i U_i$	$O(n \log n)$	Lawler, 1976b
$1\|p_i = 1$, $w_i(C_i) = \begin{cases} w_1, \ 0 < C_i \le d_{i1} \\ w_2, \ d_{i1} < C_i \end{cases} \left\| \sum w_i(C_i)\right.$	$1\|\hat{p}_i = 1\|\sum U_i$	$O(n)$	Monma, 1982

PP: Pseudopolynomial time algorithm; B&B: Branch and bound algorithm; FPTAS: Fully polynomial time approximation scheme.

The problem analyzed in the next property is more general, i.e., the values of the jobs are described by some *arbitrary* functions (not only by the nonincreasing stepwise ones). The only condition is that the processing times of all the jobs are equal.

Property 2. *The problem* $1|p_i = p|\sum w_i(C_i)$, *where* $w_i(C_i)$, *are some arbitrary functions, can be solved optimally in* $O(n^3)$ *time.*

Proof: In order to prove Property 2 we show that our problem can be reduced to the usual maximization assignment problem:

$$\text{Maximize} \quad \Phi = \sum_{i=1}^{m} \sum_{j=1}^{m} c_{ij} x_{ij}$$

$$\text{subject to} \quad \sum_{j=1}^{m} x_{ij} = 1, \quad (i = 1, \dots, m)$$

$$\sum_{i=1}^{m} x_{ij} = 1, \quad (j = 1, \dots, m)$$

$$x_{ij} = 0 \text{ or } 1, \quad (i, j = 1, \dots, m),$$

where c_{ij} is a nonnegative integer cost of assigning job i to position j, $x_{ij} = 1$ denotes that job i is assigned to position j and $x_{ij} = 0$—that job i is not assigned to position j. Since $p_i = p (i = 1, \dots, n)$, we can reduce $1|p_i = p|\sum w_i(C_i)$ (with parameters: n, p and $w_i(C_i)$) to the assignment problem as follows:

$$m = n; \quad c_{ij} = w_i(j \cdot p).$$

It is known that the assignment problem can be solved in $O(n^3)$ steps (Lawler, 1976a). □

4 Pseudopolynomial time algorithm

In this section, in order to solve the following problem:

$$1 \left| w_i(C_i) = \begin{cases} w_{i1}, & 0 < C_i \le d_1 \\ w_{i2}, & d_1 < C_i \le d_2 \\ \vdots & \\ w_{ik}, & d_{k-1} < C_i \end{cases} \right| \sum w_i(C_i), \tag{3}$$

we present a pseudopolynomial time algorithm, which is based on the dynamic programming method (Bellman, 1957; Cheng et al., 1997; Potts and Kovalyov, 2000; Cheng and Kovalyov, 2001).

In our algorithm, the jobs will be considered in their natural order $1, 2, \dots, n$. We shall consider k disjunctive sets of jobs $X_1, \dots, X_k$ such that jobs of the set X_{j-1} are scheduled before the jobs of the set X_j, and jobs of the set X_j complete before d_j, $j = 1, \dots, k$, where $d_k = \infty$. The sets $X_1, \dots, X_k$ are empty at the beginning and we shall assign jobs to these sets assuming that job $i \in X_j$ contributes w_{ij} to the objective function. Since $w_{i1} > w_{i2} > \dots > w_{ik}$, if $i \in X_j$ and $C_i \le d_{j-1}$, then job i contributes $w_{ij-1} > w_{ij}$ to the objective function. Therefore, we do not need to check if job $i \in X_j$ completes before d_{j-1}. This action will not eliminate an optimal schedule.

Let us define function $W_i(P_1, \dots, P_{k-1})$ as the maximum sum of job values of partial schedules consisting of the first i jobs $1, \dots, i$, where the sum of the processing times of the jobs assigned to the set X_j is equal to P_j, $j = 1, \dots, k-1$. Denote $T_i = \sum_{l=1}^{i} p_l$ $i = 1, \dots, n$. Notice that for any schedule in the *state* $(i, P_1, \dots, P_k)$, the sum of the processing times of the jobs assigned to the set X_k is uniquely determined as $P_k = T_i - (P_1 + \cdots + P_{k-1})$.

Consider a partial schedule in the state $(i, P_1, \ldots, P_{k-1})$ and the corresponding functional value $W_i\ (P_1, \ldots, P_j, \ldots, P_{k-1})$. If we assigned the last scheduled job (i.e., job i) to the set X_j, then the value of P_j was increased by p_i (from $P_j - p_i$ up to P_j) and the functional value $W_{i-1}(P_1, \ldots, P_j - p_i, \ldots, P_{k-1})$ was increased by w_{ij}, reaching the value $W_i(P_1, \ldots, P_j, \ldots, P_{k-1})$. Furthermore, if the set of state variables $(i, P_1, \ldots, P_{k-1})$ corresponds to a partial schedule that can be extended to an optimal schedule, then inequalities

$$\sum_{l=1}^{j} P_l \le d_j, \quad j = 1, \ldots, k-1,$$

must be satisfied. These inequalities can be used to eliminate nonperspective states. For $i = 1, \ldots, n$, we can calculate recurrently the value $W_i\ (P_1, \ldots, P_{k-1})$ for all possible values of P_j, $j = 1, \ldots, k-1$, and obtain the optimal criterion value

$$W^* = \max\left\{ W_n(P_1, \ldots, P_{k-1}) | P_j = 0, 1, \ldots, \min\{T_n, d_j\}; \sum_{l=1}^{j} P_l \le d_j, j = 1, \ldots, k-1 \right\}.$$

A formal description of the pseudopolynomial time algorithm (PPolyn1) for solving problem (3) is given as follows.

Algorithm PPolyn1

Step 1: Set $X_j = \emptyset$ for $j = 1, \ldots, k$ and

$$W_i(P_1, \ldots, P_{k-1}) := \begin{cases} 0, & \text{if } (i, P_1, \ldots, P_{k-1}) = (0, 0, \ldots, 0), \\ -\infty, & \text{otherwise,} \end{cases}$$

for $i = 0, 1, \ldots, n$ and $P_j = 0, 1, \ldots, \min\{T_n, d_j\}$; $j = 1, \ldots, k-1$. Then set $i := 1$.

Step 2: For each $P_j = 0, 1, \ldots, \min\{T_i, d_j\}$; $j = 1, \ldots, k-1$, calculate:

$$W_i(P_1, \ldots, P_{k-1}) = \begin{cases} \max \begin{Bmatrix} W_{i-1}(P_1 - p_i, P_2, \ldots, P_{k-1}) + w_{i1}, \\ W_{i-1}(P_1, P_2 - p_i, \ldots, P_{k-1}) + w_{i2}, \\ \vdots \\ W_{i-1}(P_1, P_2, \ldots, P_{k-1} - p_i) + w_{ik-1}, \\ W_{i-1}(P_1, P_2, \ldots, P_{k-1}) + w_{ik}, \end{Bmatrix}, & \text{if } \sum_{l=1}^{j} P_l \le d_j, \quad j = 1, \ldots, k-1, \\ -\infty, & \text{otherwise,} \end{cases}$$

If the above maximum is reached on jth component, then job i is assigned to the set X_j, $j \in \{1, \ldots, k\}$.

If $i = n$, then go to Step 3. othewise set $i := i + 1$ and repeat Step 2.

Step 3: Calculate the optimal criterion value:

$$W^* = \max\left\{ W_n(P_1, \ldots, P_{k-1}) | P_j = 0, 1, \ldots, \min\{T_n, d_j\}; \sum_{l=1}^{j} P_l \le d_j, \quad j = 1, \ldots, k-1 \right\}$$

and find the corresponding optimal schedule π^* by backtracking. In the schedule π^*, jobs of the sets $X_1, \ldots, X_k$ are sequenced arbitrarily and jobs of the set X_{j-i} precede jobs of the set X_j, $j = 1, \ldots, k$.

Property 3. *The problem* $1 \left| w_i(C_i) = \begin{cases} w_{i1}, & 0 < C_i \le d_1 \\ w_{i2}, & d_1 < C_i \le d_2 \\ \vdots \\ w_{ik}, & d_{k-1} < C_i \end{cases} \right| \sum w_i(C_i)$ *can be solved optimally in pseudopolynomial time* $O(nk \prod_{j=1}^{k-1} \min\{T_n, d_j\})$ *by the algorithm PPolyn1, where* $T_n = \sum_{i=1}^{n} p_i$

Proof: We first give a justification of the optimality of our algorithm. Consider a partial optimal schedule corresponding to some state $(i, P_1, \ldots, P_{k-1})$, $i \in \{0, 1, \ldots, n-1\}$. Our algorithm can select a different partial schedule for further expansion, the one with the maximum objective function value. This selected partial schedule can be extended by jobs $i+1, \ldots, n$ in the same way as the partial optimal schedule, thus leading to the same final state $(n, P_1^*, \ldots, P_{k-1}^*)$ with nonworse functional value. Assume that in the schedule constructed by our algorithm, a job $i \in X_j$ completes before or at d_{j-1}. In this case, there must exist a final state $(n, P_1^*, \ldots, P_{j-1}^* + p_i, P_j^* - p_i, \ldots, P_{k-1}^*)$ with larger functional value, which is impossible. Therefore, our assumption is incorrect and each job from the set X_j completes in the interval $(d_{j-1}, d_j]$, $j = 1, \ldots, k$, in the schedule constructed by our algorithm.

Let us calculate the computational complexity of the algorithm. It is determined by Step 2. Since there are n jobs, Step 2 is repeated n times. Since we have $\min\{T_n, d_j\}$ different values of P_j for $j = 1, \dots, k-1$, in each iteration of Step 2, $\prod_{j=1}^{k-1} \min\{T_n, d_j\}$ different values of $W_i(P_1, \dots, P_{k-1})$ are calculated. Each of them can be obtained in $O(k)$ time. Therefore, Step 2 requires $O\ (nk \prod_{j=1}^{k-1} \min\{T_n, d_j\})$ time, which is the overall computational complexity of Algorithm PPolyn1. $\square$

5 Heuristic algorithms

Since the problem $1\left| w_i(C_i) = \begin{cases} w_{i1}, & 0 < C_i \le d_{i1} \\ w_{i2}, & d_{i1} < C_i \le d_{i2} \\ \vdots & \\ w_{ik}, & d_{ik-1} < C_i \end{cases} \right| \sum w_i(C_i)$ is NP-hard, it is highly unlikely to construct the fast algorithms (i.e., polynomial) which give optimal solutions of the problem. Therefore, in order to solve it, we construct and experimentally test some heuristic algorithms. Thus, in this section, we describe four fast heuristic algorithms with computational complexity equal to $O(n \log n)$ and a modification of one of them with computational complexity equal to $O(kn \log n)$, where k denotes the number of moments of job value changes. Then, we show the results of numerical experiments which compare the efficiency of the considered algorithms.

The algorithms $p_i \nearrow$, $w_{i1} \searrow$ and $p_i/w_{i1} \nearrow$ construct the solutions in $O(n \log n)$ time by sequencing the jobs in nondecreasing order of p_i, in nonincreasing order of w_{i1} and in nondecreasing order of p_i/w_{i1}, respectively. Two next algorithms M_{mdf} and kM_{mdf} are minor and major modifications of Moore Algorithm, which solves optimally the problem $1\|\sum U_i$ (Moore, 1968; Pinedo, 1995; Lawler et al., 1993). Let us recall Moore algorithm at first.

Moore algorithm
creates solution $\pi = \pi^1\pi^a$, where

π^1—subpermutation of jobs i completed before their d_i,
π^a—subpermutation of jobs i completed after their d_i.

Step 1: Find unscheduled job i^* with the minimum d_{i^*} and put it at the end of the current π^1. If $\sum_{i\in\pi^1} p_i > d_{i^*}$, find job $k^* \in \pi^1$ that satisfies $p_{k^*} = \max_{i\in\pi^1}(p_i)$ and move it from π^1 to the end of π^a.
Step 2: Repeat Step 1 until there are no unscheduled jobs. The solution is given by $\pi = \pi^1\pi^a$.

Minor modification of Moore algorithm (M_{mdf}) differs from Moore algorithm only by the following two aspects:

(i) $d_i = d_{i1}$,
(ii) job k^* should satisfy $p_{k^*}/w_{k^*1} = \max_{i\in\pi^1}(p_i/w_{i1})$.

Note that M_{mdf} uses only the first moment of job value change d_{i1}, ignoring $d_{i2}, d_{i3}, \dots, d_{ik-1}$.

Major modification of Moore algorithm (kM_{mdf}) uses all the moments of job value changes: $d_{i1}, d_{i2}, \dots, d_{ik-1}$. The main element of kM_{mdf} is a loop, which is executed $k-1$ times, i.e., the number of job value changes. In each iteration j of this loop there is created subpermutation π^j, in which jobs are scheduled before their moments d_{ij} using M_{mdf} Algorithm. The final permutation is obtained as follows: $\pi = \pi^1\pi^2 \dots \pi^j \dots \pi^{k-1}$.

Major modification of Moore algorithm (kM_{mdf})

Step 1: Set $\pi := \emptyset$, $\pi^j := \emptyset$ $(j = 1, 2, \dots, k-1)$, $J^c := \{1, 2, \dots, n\}$, $J^d := \emptyset$ and $j := 1$.
Step 2: Find a job i^* that satisfies $d_{i^*j} = \min_{i\in J^c}(d_{ij})$. Put job i^* on the last position in π^j. Set $J^c := J^c \backslash \{i^*\}$ and go to Step 3.
Step 3: If $\sum_{i\in\pi} p_i + \sum_{i\in\pi^j} p_i \le d_{i^*j}$ then go to Step 4, otherwise find job $k^* \in \pi^j$ that satisfies $p_{k^*}/w_{k^*j} = \max_{i\in\pi^j}(p_i/w_{ij})$ and add it to J^d. Set $\pi^j := \pi^j/\{k^*\}$.
Step 4: If $J^c = \emptyset$ then set $\pi := \pi\pi^j$, $J^c := J^d$, $J^d := \emptyset$ and go to Step 5, otherwise go to Step 2.
Step 5: If $J^c = \emptyset$ or $j = k$ then STOP – π is the solution, otherwise set $j := j + 1$ and go to Step 2.

Computational complexity of M_{mdf} and kM_{mdf} is equal to $O(n \log n)$ and $O(kn \log n)$, respectively.

In the numerical experiment, made on PC with Pentium II 300 MHz, 176 MB RAM and Windows XP, the parameter values of the problem were randomly generated according to the uniform distribution, and the algorithms were tested for the following different sets of parameter values:

Set 1 (with small values w_{ij}):

$$p_i \in (0, 90),\ w_{i1} \in [1, 10),\ k \in [1, 10),$$
$$d_{ij} \in [d_{ij-1} + 100, d_{ij-1} + 200);$$

Set 2 (with large values w_{ij}):

$$p_i \in (0, 90),\ w_{i1} \in [1, 100),\ k \in [1, 10),$$
$$d_{ij} \in [d_{ij-1} + 100, d_{ij-1} + 200);$$

Set 3 (with large processing times p_i and large values w_{ij}):

$$p_i \in (0, 160),\ w_{i1} \in [1, 100),\ k \in [1, 10),$$
$$d_{ij} \in [d_{ij-1} + 100, d_{ij-1} + 200);$$

for $i = 1, \dots, n$ and $j = 1, \dots, k-1$. Moreover, the values of w_{ij} for $j = 2, \dots, k$ are randomly generated from the following intervals: $w_{ij} \in (0, w_{ij-1})$.

For each set of the parameter values there were randomly-generated 500 instances of the problem for each $n = 9$, 50, 100 and 500 jobs. Thus, it generated 6000 instances of the problem in all. For $n = 9$, for each generated instance the optimal solution was found by explicit enumeration (let OPT denote the criterion value for this solution). Then, for each algorithm, we calculated a performance ratio: $(OPT/ALG - 1) \cdot 100\%$, where ALG denotes an objective function value obtained by the considered algorithm. For $n > 9$, for each generated instance of the considered problem the algorithm which gave a solution with the largest criterion value was found. Let A_{BEST} denote the largest criterion value. Then, the following performance ratio $(A_{\text{BEST}}/ALG - 1) \times 100\%$ was calculated for the other algorithms. The average values of the performances defined above, obtained for all 500 generated instances of the problem, with a given n and a given set of parameters values, are shown in Table 2.

As it can be seen in Table 2, kM_{mdf} algorithm is the most accurate (e.g., for $n > 9$, for most cases it gives best solutions, and there are usually, about several percent worse than the optimum). But two others ($p_i/w_{i1} \nearrow$ and M_{mdf}) are also accurate—for $n = 9$; for some cases they deliver even better solutions than kM_{mdf}. The simple heuristics ($p_i \nearrow$ and $w_{i1} \searrow$) are the least accurate from among the considered algorithms—e.g., they deliver solutions about 10–40% worse than the optimal ones for $n = 9$. Moreover, one can notice that the values of processing times p_i (to be precise—the ratio of processing times p_i to the differences $(d_{ij-1} - d_{ij})$, $j = 1, \dots, k-1$) have an influence on algorithms accuracy—all considered algorithms deliver worse solutions if the processing times p_i have large values.

In general, the most accurate results were obtained at the expense of large execution times, since kM_{mdf} algorithm (which delivers most of these results) has $O(kn \log n)$ complexity, which is linearly dependent on the number of moments of job value changes (parameter k), while the remaining algorithms work in $O(n \log n)$ time. Hence, the influence of k on the algorithms execution times was tested. An example of average execution times (represented in milliseconds) of the algorithms for Set 3 is given in Table 3, where notation '0.000' means that an average execution time is less than 1 μs.

The results given in Table 3 confirm the observation following from the computational complexity analysis of the algorithms—the execution times of kM_{mdf} increases with the increase of the parameter k while the execution times of the remaining algorithms do not increase. However, we can observe that the execution times of kM_{mdf} are usually comparable to the other ones for $k \le 100$ (obviously, the differences between the execution times may become noticeable for large problem instances, e.g., for $n = 500$ and $k = 100$).

Therefore, we can use kM_{mdf} algorithm if we solve the problem instances with not many moments of job value changes. Otherwise, we have to choose between the algorithms that are fast but not much accurate and a bit slower ones with quite good accuracy.

The presented heuristic algorithms can also be used to solve the pseudopolynomially solvable case analyzed in Section 4 (i.e., the one with common moments of job value changes for all the jobs). Thus, we made the experiments, similar to the ones described above, comparing the efficiency of the heuristic algorithms. The sizes of the problem, n, and the sets of parameter values are given along with the results of the tests in Table 4, where each entry is an average performance ratio of 100 randomly generated instances of the problem. For Set 1 and $n = 9$, the performances were obtained based on the optimal solution values OPT given by explicit enumeration, and for Set 4 and $n = 25$

Table 3 An example of average execution times (ms) of the considered algorithms (for Set 3), depending on the parameter k

n	k	$p_i \nearrow$	$w_{i1} \searrow$	$p_i/w_{i1} \nearrow$	M_{mdf}	kM_{mdf}
50	10	0,000	0,000	0,625	0,166	0,078
	100	0,000	0,625	0,001	0,003	5,625
	250	0,000	0,000	0,006	1,253	0,156
	500	1,907	0,000	0,000	5,000	0,626
	1000	0,005	0,000	0,000	0,156	0,010
100	10	5,625	0,000	0,391	6,875	0,463
	100	5,044	0,010	1,253	5,669	2,914
	250	0,000	0,078	1,885	0,159	7,966
	500	5,039	0,001	0,167	6,289	3,569
	1000	0,000	0,649	0,003	0,089	9,905
500	10	10,313	15,000	40,834	20,347	89,699
	100	10,853	10,000	40,005	29,674	152,064
	250	18,947	15,224	49,999	30,469	206,366
	500	15,654	18,586	48,347	35,887	210,292
	1000	17,402	19,687	44,261	36,038	223,708

Table 2 Average performance ratios (%) for the considered algorithms

n	$p_i \nearrow$	$w_{i1} \searrow$	$p_i/w_{i1} \nearrow$	M_{mdf}	kM_{mdf}
Set 1: $p_i \in (0, 90)$, $w_{i1} \in [1, 10)$, $k \in [1, 10)$, $\Delta d_{ij} \in [100, 200)$					
9	5.99	11.13	5.62	4.35	**4.07**
50	23.88	88.42	13.76	14.37	**0.22**
100	10.19	84.18	4.48	11.66	**0.00**
500	23.16	405.05	12.22	23.87	**0.00**
Set 2: $p_i \in (0, 90)$, $w_{i1} \in [1, 100)$, $k \in [1, 10)$, $\Delta d_{ij} \in [100, 200)$					
9	8.60	11.92	8.00	**5.12**	5.26
50	19.73	81.38	1.79	11.37	**0.00**
100	33.86	145.04	14.93	24.10	**0.00**
500	20.11	199.53	9.14	16.90	**0.00**
Set 3: $p_i \in (0, 160)$, $w_{i1} \in [1, 100)$, $k \in [1, 10)$, $\Delta d_{ij} \in [100, 200)$					
9	47.01	46.18	30.42	19.21	**18.45**
50	60.36	121.64	13.69	32.01	**0.00**
100	25.68	141.55	13.37	20.64	**0.00**
500	14.21	141.44	7.63	11.67	**0.00**

Table 4 Average performance ratios (%) for the considered algorithms for the case with common moments of job value changes

n	$p_i \nearrow$	$w_{i1} \searrow$	$p_i/w_{i1} \nearrow$	M_{mdf}	kM_{mdf}
Set 1: $p_i \in (0, 90)$, $w_{i1} \in [1, 10)$, $k \in [1, 10)$, $\Delta d_{ij} \in [100, 200)$					
9	16.71	10.08	6.31	5.22	**4.97**
50	13.47	65.05	3.61	25.35	**0.02**
100	16.30	154.31	5.40	22.01	**0.00**
500	23.83	310.62	7.70	46.74	**0.00**
Set 4: $p_i \in (0, 9)$, $w_{i1} \in [1, 10)$, $k \in [1, 5)$, $\Delta d_{ij} \in [5, 10)$					
25	17.40	45.25	9.47	13.86	**6.07**
50	29.84	108.72	11.27	28.30	**2.23**
100	16.41	63.59	9.99	16.07	**0.00**
500	13.29	20.71	4.73	8.89	**0.00**

and 50, the performances were obtained based on *OPT* values given by PPolyn1 Algorithm. For the remaining cases, the performances were obtained, based on the largest-known criterion values for each instance, A_{BEST}. In Set 4, the intervals of generating the parameter values (especially for k and Δd_{ij}) were limited because of too long execution times and too large memory occupation of the dynamic programming algorithm PPolynl.

The results presented in Table 4 show that the accuracies of the considered algorithms concerning the pseudopolynomially solvable case are similar to ones dealing with the general version of the problem. The execution times of the algorithms in both cases are also similar and they do not exceed 50 ms in the realized tests. Therefore, the heuristic algorithms (especially $p_i/w_{i1} \nearrow$, M_{mdf} and kM_{mdf}) can be alternative tools to the pseudopolynomial time algorithm PPolynl to solve the analyzed case of the problem, especially for the instances with large number of the moments $d_1, d_2, \dots, d_{k-1}$ and their values.

6 Generalizations and conclusions

A single processor scheduling problem with a stepwise function of change of job values was considered in this paper. We proved that the special case of this problem, with a single moment of job value change, is NP-hard in the ordinary sense, showing that it is equivalent to the well-known problem of minimizing weighted number of late jobs. Additionally, an exact pseudopolynomial time algorithm, based on the dynamic programming method, was constructed for the case with an arbitrary number of common moments of job value changes for all the jobs. Thus, the last case is also NP-hard in the ordinary sense. However, an open question is whether the general case of the problem is NP-hard in the ordinary sense or in the strong sense.

The dynamic programming algorithm mentioned above can be generalized to the following case with fixed number, m, of the *parallel processors:*

$$Pm \left| w_i(C_i) = \begin{cases} w_{i1}, \ 0 < C_i \le d_1 \\ w_{i2}, \ d_1 < C_i \le d_2 \\ \vdots \\ w_{ik}, \ d_{k-1} < C_i \end{cases} \right| \sum w_i(C_i).$$

As in the single processor version, the jobs will be considered in the natural order $1, 2, \dots, n$. We shall consider mk 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 of the sets $X_1^l, \dots, X_k^l$ are scheduled on processor $M_l, l = 1, \dots, m$, such that jobs of X_{j-1}^l are processed before the jobs of X_j^l, and jobs of X_j^l complete before d_j, $j = 1, \dots, k$ ($d_k = \infty$). All sets are empty at the beginning and we shall assign jobs to these sets assuming that job $i \in X_j^l$ contributes w_{ij} to the objective function (we do not need to check if job $i \in X_j^l$ completes before d_{j-1} like for the single processor case).

For partial schedules, consisting of the jobs $1, \dots, i$, we can define function

$$W_i\left(P_1^1, \dots, P_{k-1}^1, P_1^2, \dots P_{k-1}^2, \dots, P_1^m, \dots P_{k-1}^m\right)$$

as the maximum sum of job values, where P_j^l denotes sum of processing times of jobs assigned to the set $X_j^l, l = 1, \dots, m$ and $j = 1, \dots, k-1$. If any job is assigned to one of the sets $X_k^l, l = 1, \dots, m$, then there is no difference in criterion value, for which of them this job will be assigned. Thus, the values of $P_k^l, l = 1, \dots, m$, are not important. Denote, as in the single processor case, $T_i = \sum_{q=1}^{i} p_q, i = 1, \dots, n$.

Consider a partial schedule in the state $(i, P_1^1, \dots, P_{k-1}^1, P_1^2, \dots, P_{k-1}^2, \dots, P_1^m, \dots, P_{k-1}^m)$ and the corresponding functional value $W_i(P_1^1, \dots, P_{k-1}^1, P_1^2, \dots, P_{k-1}^2, \dots, P_1^m, \dots, P_{k-1}^m)$. If we assigned job i to the set X_j^l, then the value of P_j^l was increased by p_i and the functional value $W_{i-1}(P_1^1, \dots, P_1^l, \dots, P_j^l - p_i, \dots, P_{k-1}^l, \dots, P_{k-1}^m)$ was increased by w_{ij}, up to $W_i(P_1^1, \dots, P_1^l, \dots, P_j^l, \dots, P_{k-1}^l, \dots, P_{k-1}^m)$.

We can use the following inequalities, similar to the ones for the single processor problem:

$$\sum_{q=1}^{j} P_q^l \le d_j, \quad j = 1, \dots, k-1; \quad l = 1, \dots, m,$$

to eliminate nonperspective states, since only the partial schedules, for which these inequalities hold, can be extended to an optimal schedule.

Thus, for $i = 1, \dots, n$, we can calculate recurrently the value $W_i(P_1^1, \dots, P_{k-1}^1, P_1^m, \dots, P_{k-1}^m)$ for all possible P_j^l, $l = 1, \dots, m$ and $j = 1, \dots, k-1$, in order to obtain the optimal criterion value:

$$W^* = \max_{\substack{1 \le j \le k-1 \\ 1 \le l \le m}} \left\{ W_n\left(P_1^1, \dots, P_{k-1}^m\right) \middle| \sum_{q=1}^{j} P_q^l \le d_j, \quad P_j^l = 0, 1, \dots, \min\{T_n, d_j\} \right\}.$$

A formal description of the pseudopolynomial time algorithm (PPolyn2) for solving the parallel processor case of the problem is given as follows.

Algorithm PPolyn2

Step 1: Set $X_j^l = \emptyset$ for $j = 1, \dots, k$; $l = 1, \dots, m$ and

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

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

Step 2: For each $P_j^l = 0, 1, \dots, \min\{T_i, d_j\}$; $l = 1, \dots, m$; $j = 1, \dots, k-1$, calculate:

$$W_i\left(P_1^1, \dots, P_1^l, P_2^l, \dots, P_{k-1}^l, \dots, P_{k-1}^m\right) = \begin{cases} \max\limits_{1 \le l \le m} \left\{ \begin{array}{l} W_{i-1}\left(P_1^1, \dots, P_1^l - p_i, \dots, P_{k-1}^m\right) + w_{i1}, \\ W_{i-1}\left(P_1^1, \dots, P_2^l - p_i, \dots, P_{k-1}^m\right) + w_{i2}, \\ \qquad \vdots \\ W_{i-1}\left(P_1^1, \dots, P_{k-1}^l - p_i, \dots, P_{k-1}^m\right) + w_{ik-1}, \\ W_{i-1}\left(P_1^1, \dots, P_1^l, P_2^l, \dots, P_{k-1}^l, \dots, P_{k-1}^m\right) + w_{ik}, \end{array} \right\}, & \text{if } \sum_{q=1}^{j} P_q^l \le d_j, \quad j = 1, \dots, k-1, \quad l = 1, \dots, m, \\ -\infty, & \text{otherwise,} \end{cases}$$

If the above maximum is reached on component (j, l), then job i is assigned to the set X_j^l, $j \in \{1, \dots, k\}, l \in \{1, \dots, m\}$.

If $i = n$, then go to Step 3, otherwise set $i := i + 1$ and repeat Step 2.

Step 3: Calculate the optimal criterion value:

$$W^* = \max_{\substack{1 \le j \le k-1 \\ 1 \le l \le m}} \left\{ W_n\left(P_1^1, \dots, P_{k-1}^m\right) \middle| \sum_{q=1}^{j} P_q^l \le d_j \ , \quad P_j^l = 0, 1, \dots, \min\{T_n, d_j\} \right\}$$

and find the corresponding optimal solution $\Pi^* = \{\pi_1^*, \dots, \pi_m^*\}$ by backtracking. In the sequence π_l^* (i.e., sequence of jobs assigned to execute on the processor $M_l, l = 1, \dots, m$), jobs of the sets $X_1^l, \dots, X_k^l$ are ordered, such that they are processed in arbitrary order in each set and jobs of the set X_{j-1}^l precede jobs of the set X_j^l, $j = 1, \dots, k$.

Property 4. *The problem* $Pm \left| w_i(C_i) = \begin{cases} w_{i1}, & 0 < C_i \le d_1 \\ w_{i2}, & d_1 < C_i \le d_2 \\ \vdots \\ w_{ik}, & d_{k-1} < C_i \end{cases} \right|$ $\sum w_i(C_i)$ *can be solved optimally in pseudopolynomial time* $O(nkm(\Pi_{j=1}^{k-1} \min\{T_n, d_j\})^m)$ *by the algorithm PPolyn2, where* $T_n = \sum_{i=1}^{n} p_i$.

Proof: A justification of the optimality of algorithm PPolyn2 is analogous to the one of algorithm PPolynl (which solves the corresponding single processor problem).

Let us calculate the computational complexity of algorithm PPolyn2. It is determined by Step 2. Since there are n jobs, Step 2 is repeated n times. Since we have $\min\{T_n, d_j\}$ different values of P_j^l for $j = 1, \dots, k-1$ and $l = 1, \dots, m$, in each iteration of Step 2, $(\Pi_{j=1}^{k-1} \min\{T_n, d_j\})^m$ different values of $W_i(P_1^1, \dots, P_{k-1}^m)$ are calculated. Each of them can be obtained in $O(km)$ time. Therefore, Step 2 requires $O(nkm(\Pi_{j=1}^{k-1} \min\{T_n, d_j\})^m)$ time and it is the overall computational complexity of algorithm PPolyn2. $\square$

Thus, we have proved that the considered parallel processor case of the problem is also NP-hard in the ordinary sense.

References

Bachman, A., A. Janiak, A. Kozik, and M. Winczaszek, "Heuristic algorithms for a single machine scheduling problem with changeable job values," *Scientific Bulletins of Silesian Polytechnics, s.Automatics*, **134**, 23–32 (2002) (in Polish).

Bachman, A., A. Janiak, A. Kozik, and M. Winczaszek, "Single machine scheduling problem with exponentially dependent job values," *Scientific Bulletins of Silesian Polytechnics, s.Automatics*, **134**, 13–22 (2002) (in Polish).

Bellman, R., *Dynamic Programming*. Princeton University Press, Princeton, MJ, 1957.

Cheng, T. C. E. and M. Y. Kovalyov, "Single machine batch scheduling with sequential job processing," *IIE Transplantation*, **33**, 413–420 (2001).

Cheng, T. C. E., M. Y. Kovalyov, and B. M.-T. Lin, "Single machine scheduling to minimize batch delivery and job earliness penalties," *SIAM Journal of Optimization*, **7**(2), 547–559 (1997).

Coffman, Jr., E. G. (ed.), *Computer & Job-Shop Scheduling Theory*, Wiley, New York, 1976.

Gens, G. V. and E. V. Levner, "Approximation algorithm for some scheduling problems," *Engineering Cybernetics*, **6**, 33–46 (1978).

Gens, G. V. and E. V. Levner, "Fast approximation algorithm for job sequencing with deadlines," *Discrete Applied Mathematics*, **3**, 313–318 (1981).

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

Janiak, A. and T. Krysiak, "A single machine scheduling problem with a single change of job value," in: *Proceedings of the 9th IEEE International Conference on Methods and Models in Automation and Robotics (MMAR 2003)*, 25–28 August 2003, Miedzyzdroje, Poland (2003), pp. 1139–1143.

Karp, R. M., "Reducibility among combinatorial problems," in: Miller R. E. and Thatcher J. W. (eds.), *Complexity of Computer Computations*, Plenum Press, New York (1972), pp. 85–103.

Lawler, E. L., *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart & Winston, New York, 1976a.

Lawler, E. L., "Sequencing to minimize the weighted number of tardy jobs," *RAIRO Research Operations*, **10**(Suppl.), 27–33 (1976b).

Lawler, E. L., J. K. Lenstra, A. H. G. Rinooy Kan, and D. B. Shmoys, "Sequencing and scheduling: Algorithms and complexity," in: Graves S. C., Rinnooy Kan A. H. G. and Zipkin P. (eds.), *Handbooks in Operations Research and Management Science*, North-Holland (1993), pp. 445–522.

Lawler, E. L. and J. M. Moore, "A functional equation and its application to resource allocation and sequencing problems," *Management Science*, **16**, 77–84 (1969).

Monma, C. L., "Linear-time algorithms for scheduling on parallel processors," *Oper Res*, **30**, 116–124 (1982).

Moore, J. M., "An *n* jobs, one machine sequencing algorithm for minimizing the number of late jobs," *Management Science*, **15**, 102–109 (1968).

Pinedo, M., *Scheduling: Theory, Algorithms and Systems*, Prentice Hall, NJ, 1995.

Potts, C. N. and M. Y. Kovalyov, "Scheduling with batching: A review," *European Journal of Operational Research*, **120**, 228–249 (2000).

Sahni, S., "Algorithms for scheduling independent tasks," *Journal of the Association for Computing Machinery*, **23**, 116–127 (1976).

Villarreal, F. J. and R. L. Bulfin, "Scheduling a single machine to minimize the weighted number of tardy jobs," *IIE Transition*, **15**, 337–343 (1983).

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