Int. J. Appl. Math. Comput. Sci., 2016, Vol. 26, No. 3, 693–706
DOI: 10.1515/amcs-2016-0048

SCHEDULING PREEMPTABLE JOBS ON IDENTICAL PROCESSORS UNDER VARYING AVAILABILITY OF AN ADDITIONAL CONTINUOUS RESOURCE

RAFAŁ RÓŻYCKI [a], GRZEGORZ WALIGÓRA [a,*], JAN WĘGLARZ [a]

[a] Institute of Computing Science
Poznań University of Technology, Piotrowo 2, 60-965 Poznań, Poland
e-mail: {rafal.rozycki,grzegorz.waligora,jan.weglarz}@cs.put.poznan.pl

In this work we consider a problem of scheduling preemptable, independent jobs, characterized by the fact that their processing speeds depend on the amounts of a continuous, renewable resource allocated to jobs at a time. Jobs are scheduled on parallel, identical machines, with the criterion of minimization of the schedule length. Since two categories of resources occur in the problem: discrete (set of machines) and continuous, it is generally called a discrete-continuous scheduling problem. The model studied in this paper allows the total available amount of the continuous resource to vary over time, which is a practically important generalization that has not been considered yet for discrete-continuous scheduling problems. For this model we give some properties of optimal schedules on a basis of which we propose a general methodology for solving the considered class of problems. The methodology uses a two-phase approach in which, firstly, an assignment of machines to jobs is defined and, secondly, for this assignment an optimal continuous resource allocation is found by solving an appropriate mathematical programming problem. In the approach various cases are considered, following from assumptions made on the form of the processing speed functions of jobs. For each case an iterative algorithm is designed, leading to an optimal solution in a finite number of steps.

Keywords: machine scheduling, preemptable jobs, continuous resource, makespan, mathematical programming.

1. Introduction

In the classical scheduling problems it is assumed that resources can be assigned to jobs in amounts from a given finite set only (i.e., in discrete numbers of units). Such resources are called *discrete* (or discretely divisible). However, in many practical situations resources can be allotted to jobs in arbitrary numbers from a given interval (i.e., in real numbers). Such resources are called *continuous* (or continuously divisible). Situations of this type occur when, e.g., jobs are processed by parallel processing units driven by a common (electric, pneumatic, hydraulic) power source, like the frequently supplied grinding or mixing machines, electrolytic tanks, or refueling terminals. More recently, the so-called power (or energy) aware scheduling problems have been considered, where a set of processors is a discrete resource and power (energy) is a continuous resource (see, e.g., Różycki and Węglarz, 2012).

In general, two job processing models appear in the literature. In the first model, the *processing time vs. resource amount*, the job duration is a function of the amount of a continuous resource allotted to this job. This model is a straightforward generalization of the well-known discrete time-resource tradeoff model. It is implicitly assumed that the resource amount allocated to a job does not change during its execution. In the second model, the *processing speed vs. resource amount*, the processing speed of a job is a function of the amount of a continuous resource allotted to this job at a time. In this case, the amount of the continuous resource allotted to a job may change during its execution. From between the two above-mentioned models, the processing speed vs. resource amount model is more natural in the majority of practical situations, since it reflects directly the temporary nature of renewable resources. As examples, functions like rotational speed vs. electric current, or progress speed vs. the number of primary memory pages allotted to a program can be given. Moreover, the processing speed vs. resource amount model allows performing a deeper analysis of the properties of optimal schedules, and can

*Corresponding author

even lead to analytical results in some cases. Because of that, it is sometimes reasonable to treat a discrete resource as a continuous one in order to use this model. Such an approach may be applied when there are sufficiently many allotments of the discrete resource for processing a job, e.g., in scalable or massively parallel processor systems.

Discrete-continuous scheduling problems arise when jobs simultaneously require discrete and continuous resources for their executions. Machine scheduling problems of this type were considered, e.g., by Gorczyca and Janiak (2010) or Janiak (1991), whereas project scheduling problems were discussed by Kis (2005), Leachman (1983), Leachman *et al*. (1990) or Waligóra (2011; 2014). In this paper we consider a problem of scheduling preemptable, independent jobs on parallel, identical machines under an additional, continuous, renewable resource to minimize the schedule length (or the makespan). However, in all the previous works it was assumed that the availability of the continuous resource did not depend on time, i.e., the total amount of the resource available at a time was constant. In this paper we consider a generalization of those problems, and assume that the total available amount of the continuous resource periodically varies over time. The length of each resource availability period, as well as the amount (nonnegative and constant) of the resource available in the period, is known in advance. Practical motivations for considering such a problem can be very easily found in real life. These are situations where, e.g., the continuous resource is power, and the delivery of the power source varies over time. For example, we can imagine a farm of multicore processor systems driven by a power source supported by an additional solar battery. During sunny periods the delivery of such a power source will be much higher than during cloudy hours. Furthermore, changes in the amount of power taken by a single processor can be forced for thermal reasons. In order to reduce the risk of overheating a processor of the VSP (variable speed processor) type, the frequency of its clock is periodically changed, which results in a varying amount of power taken by the processor. If such a processor is fully loaded, the power changes necessary for thermal reasons can be predicted and taken into account in algorithms for scheduling computational tasks. Similar examples concerning varying availability of a continuous resource can be given from other areas as well. The defined problem was first considered by Kis (2005), but only for convex processing speed functions of activities.

The paper is organized as follows. In Section 2 we recall the most important results concerning the problem of allocating a continuous, renewable resource among independent jobs to minimize the makespan in the absence of limited discrete resources. The section reports the basic results for the continuous resource allocation problem obtained for two classes of the processing speed functions of jobs: convex and concave functions. Section 3 contains the mathematical formulation of the problem under consideration. In Section 4 we recall the results for the case of constant continuous resource amount, whereas in Section 5 we formulate the general methodology underlying the approaches proposed in this paper. Section 6 is devoted to the case of convex processing speed functions. We show that in this case the number of machines in the problem considered is of no importance, and we give an exact polynomial algorithm for finding optimal schedules. In Section 7 we focus on the case of concave processing speed functions. Within this case we distinguish between situations when the number of jobs is not greater than that of machines (Section 7.1), and when there are more jobs than machines (Section 7.2). In both cases we present exact algorithms for finding optimal schedules, in which nonlinear mathematical programming (NLP) problems have to be solved. Some conclusions and directions for future research are given in Section 8.

2. Continuous resource allocation

In this section we recall very briefly the main theoretical results concerning the case when a continuous, renewable resource is the only limited resource, and discrete resources are unlimited.

We assume that n independent jobs with equal ready times, each characterized by the processing speed vs. resource amount function, are to be scheduled to minimize the makespan. One continuous, renewable resource is available. The availability of the resource over time is constant and, without loss of generality, we assume that its total available amount is equal to 1. The resource can be allotted to jobs in (arbitrary) amounts from the interval $[0, 1]$. The amount (unknown in advance) of the continuous resource allotted to job i, $i = 1, 2, \ldots, n$, at time t is denoted by $u_i(t)$, and $\sum_{i=1}^{n} u_i(t) \leq 1$ for any t. The resource amount $u_i(t)$ determines the processing speed of job i, which is described by the following equation:

$$\begin{aligned} \dot{x}_i(t) &= \frac{\mathrm{d}x_i(t)}{\mathrm{d}t} = f_i[u_i(t)], \\ x_i(0) &= 0, \quad x_i(C_i) = w_i, \end{aligned} \tag{1}$$

where $x_i(t)$ is the state of job i at time t; f_i is the processing speed function of job i, continuous, increasing, and such that $f_i(0) = 0$; $u_i(t)$ is the continuous resource amount allotted to job i at time t; C_i is the completion time (unknown in advance) of job i; w_i is the size (final state) of job i.

State $x_i(t)$ of job i at time t is an objective measure of work related to the processing of job i up to time t. It may denote, e.g., the number of man-hours already spent on processing job i, the volume (in cubic meters) of a

constructed building, the number of standard instructions in processing computer program i, etc.

In this case, the problem is to find an allocation of the continuous resource to jobs that minimizes the makespan. The continuous resource allocation is defined by a piecewise continuous, nonnegative vector function $u(t) = [u_1(t), u_2(t), \ldots, u_n(t)]$. Completion of job i requires that

$$x_i(C_i) = \int_0^{C_i} f_i[u_i(t)]\,\mathrm{d}t = w_i. \tag{2}$$

For simplicity, $C_{\max} = \max_{i=1,\ldots,n} C_i$ will be denoted by T throughout the remainder of the paper. The following result, proved by Węglarz (1976), is fundamental for the continuous resource allocation problem.

Theorem 1. *The minimum makespan T^* as a function of sizes of jobs $\mathbf{w} = (w_1, w_2, \ldots, w_n)$ is given by*

$$T^*(\mathbf{w}) = \min\{T > 0 : \mathbf{w}/T \in \operatorname{co} V\}.$$

where $\operatorname{co} V$ is the convex hull of V, and set V is defined as

$$V = \Big\{v : v_i = f_i(u_i), u_i \geq 0, \quad i = 1, 2, \ldots, n, \sum_{i=1}^{n} u_i \leq 1\Big\},$$

where $T^*(\mathbf{w})$ is always a convex function.

Now, let us stress that, since the processing speed functions of jobs are increasing, it is always profitable to use the total available amount of the continuous resource at any time in order to shorten the schedule. We will formulate this property as follows.

Property 1. In the continuous resource allocation problem, to minimize the makespan, the total available amount of the continuous resource is used at any time in an optimal schedule.

Two corollaries following directly from Theorem 1 use the above property (Węglarz, 1976).

Corollary 1. *For convex processing speed functions of jobs, the makespan is minimized by sequential processing of all jobs, each of them using the total available amount of the continuous resource.*

Corollary 2. *For concave functions f_i, $i = 1, 2, \ldots, n$, the makespan is minimized by fully parallel processing of all jobs using the following resource amounts:*

$$u_i^* = f_i^{-1}\Big(\frac{w_i}{T^*}\Big), \quad i = 1, 2, \ldots, n, \tag{3}$$

where T^* is the unique positive root of the equation

$$\sum_{i=1}^{n} f_i^{-1}\Big(\frac{w_i}{T}\Big) = 1. \tag{4}$$

Let us also formulate the following property as an immediate consequence of Property 1.

Property 2. In the continuous resource allocation problem, to minimize the makespan at least one job is processed at any time in an optimal schedule.

Property 2 predicates that there are no idle times (i.e., moments when the continuous resource is not used by any job) in an optimal schedule, unless different ready times of jobs constrain the possible moments of starting the jobs.

Finally, the following property is a succession of Corollary 2.

Property 3. In the continuous resource allocation problem, to minimize the makespan with concave processing rate functions of jobs, in an optimal schedule all jobs are finished at the same time.

It is easy to see that the optimum makespan T^* calculated from Eqn. (4) is common for all jobs processed in parallel, as it is used in (3) for calculating the optimal continuous resource amounts. In other words, each job is finished at the same moment T^*.

Properties 1–3 will be used in the approaches proposed further on in the paper.

Let us now comment briefly on Corollaries 1 and 2.

Firstly, Corollary 1 holds, in fact, for all functions fulfilling the condition $f_i \leq c_i u_i$, $c_i = f_i(1)$, $i = 1, 2, \ldots, n$, i.e., functions no greater than a linear one. In the sequel, the results obtained for convex functions are true for all functions fulfilling the above condition.

Secondly, Corollary 2 identifies very important cases in which an optimal resource allocation can be found in an efficient way. Generally speaking, these are the cases when Eqn. (4) can be solved analytically. From among them the ones in which Eqn. (4) is an algebraic equation of order ≤ 4 are of special importance. This is, for example, the case of power processing speed functions of the form

$$f_i(u_i) = c_i u_i^{\frac{1}{\alpha_i}},$$

where $\alpha_i \in \{1, 2, 3, 4\}$, and $i = 1, 2, \ldots, n$.

Using these functions, we can model job processing speeds in a variety of practical problems.

It should also be noticed that in both of the above corollaries preemptability of jobs is of no importance. In Corollary 1 jobs are processed sequentially, each of them using the total available amount of the continuous resource. In Corollary 2 jobs are processed using constant resource amounts (given by Eqn. (3)) from their start to their completion. As a result, allowing job preemptions does not affect optimal schedules. In the remainder of the paper we deal with preemptable jobs; however, nonpreemptable ones can be considered as well.

3. Problem formulation

We consider a discrete-continuous scheduling problem of n preemptable, independent jobs with equal ready times on m parallel, identical machines to minimize the makespan. Each job from set J, $|J| = n$, simultaneously requires for its processing a machine from set M, $|M| = m$, (the discrete resource) and an amount (unknown in advance) $u_i(t)$ of an additional continuous, renewable resource. The processing speed of job i is described by Eqn. (1). However, the important difference to the model presented in Section 2 is that now the assumption $\sum_{i=1}^{n} u_i(t) \leq 1$ for any t is no longer valid. Instead, a number H_c of cyclically repeating periods is given, and each of them will be called a *resource availability period* (RAP).

Definition 1. A *resource availability period* (RAP) is a period in which the continuous resource amount remains constant.

The h-th RAP, $h = 1, 2, \ldots, H_c$, will be denoted by RAP_h, and for each RAP_h its length Δ_h and the amount $U_h > 0$ of the continuous resource available and constant within this RAP are known. Without loss of generality we assume that $U_{h-1} \neq U_h$, $h = 2, 3, \ldots, H_c$, otherwise it is always possible to merge the RAPs not fulfilling this condition. The problem is to find a sequence of jobs on machines and, simultaneously, a continuous resource allocation that minimize the makespan $T = \max_{i=1,\ldots,n} C_i$. Solving the problem, we simultaneously find a minimal number H of RAPs in which all jobs from set J can be completed. Notice that either all n jobs can be executed within the first cycle, or it may be necessary to execute some jobs in more than H_c RAPs.

Notation used for the problem considered is presented in Table 1.

4. Methodology for a constant resource amount

Before we pass on to the case of the problem formulated in Section 3, we will first present the methodology developed for the problem of scheduling n preemptable, independent jobs on m parallel, identical machines to minimize the makespan under a constant amount of the continuous resource, i.e., $\sum_{i=1}^{n} u_i(t) = 1$ for any t. The fundamental difference in comparison with the results presented in Section 2 is, obviously, that now an additional discrete resource appears, which is a set of identical machines. As mentioned in Section 1, simultaneous presence of discrete and continuous resources leads to discrete-continuous scheduling problems.

In the methodology for solving discrete-continuous scheduling problems the two-phase approach has been widely used. It consists in (i) defining an assignment of machines to jobs in the first phase and (ii) finding an optimal continuous resource allocation among jobs under the assignment made in (i). Let us first comment how the assignment (i) is defined.

Notice that each schedule may be divided into intervals in which the continuous resource allocation remains constant. Each change in the resource allocation starts a new interval. Then it is possible to define combinations of jobs processed in parallel in consecutive intervals. A sequence of such combinations defines the assignment corresponding to the schedule considered. Feasibility of the sequence requires that (a) each job appear in at least one combination, and (b) the number of jobs in a combination do not exceed m (the number of machines).

Next, if we allocate optimally the continuous resource for a given sequence constructed in the first phase, we will obtain a minimal-length schedule for this particular sequence in the second phase. In consequence, the problem is to find a sequence leading to a globally optimal schedule. We will call it the *base sequence*.

Definition 2. A *base sequence* is a finite sequence of combinations of jobs that leads to an optimal schedule under an optimal continuous resource allocation.

Thus, in order to solve optimally the problem considered, we have to, firstly, define the form of the base sequence and, secondly, determine the way of allocating the continuous resource optimally among jobs in the sequence. Let us first comment on the form of the base sequence. Notice that since we have preemptable jobs, in general it is sufficient to consider a maximal base sequence $S\max$ composed of $s = \binom{n}{m}$ m-element

Table 1. Notation.

Symbol	Meaning
J	set of jobs
M	set of machines
i	index of a job
j	index of a machine
n	number of jobs
m	number of machines
w_i	size of job i
C_i	completion time of job i
$x_i(t)$	state of job i at time t
$u_i(t)$	amount of the continuous resource allotted to job i at time t
$f_i(u_i)$	processing speed function of job i
p_i	processing time of job i
H_c	number of cyclically repeating RAPs
RAP_h	h-th RAP
Δ_h	length of RAP_h
U_h	amount of the continuous resource available in RAP_h
H	number of RAPs where jobs are executed
T	schedule length ($C_{\max}$)

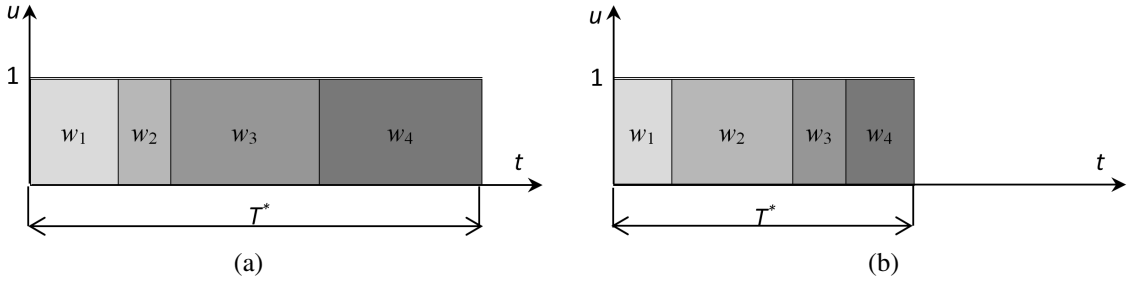


Fig. 1. Base sequence $Sn = [\{1\}, \{2\}, \{3\}, \{4\}]$ for $n = 4$, and two sample optimal schedules for different problem instances.

combinations of jobs. Such a sequence exhausts all possible assignments of m (identical) machines to n jobs, and therefore guarantees finding an optimal assignment. Each feasible schedule can be generated by using the maximal sequence.

Definition 3. A *maximal sequence* $S\max$ is a sequence composed of all $s = \binom{n}{m}$ m-element combinations without repetitions from among n jobs.

We will represent sequences of combinations in a form of vectors since, in a general case, the position of a combination in a sequence is important. For instance, the maximal sequence $S\max$ for a sample problem with n = 4 and m = 3 can be represented as

$$S\max = [\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{2, 3, 4\}].$$

However, in the problem considered, since preemptable jobs are scheduled on identical machines, the order of combinations in the sequence, as well as the order of jobs in a particular combination, does not matter.

Now, for the maximal sequence $S\max$, an optimal continuous resource allocation can be found. It requires, in general, a solution of a mathematical programming (MP) problem. However, we will show in the next two subsections that both the form of the maximal sequence and the optimal continuous resource allocation may be simplified in some special cases.

4.1. Convex processing speed functions. Based on Corollary 1, it is easy to see that in this case the makespan is minimized by sequential processing of all jobs on one machine, where each job uses the total available amount of the continuous resource. As a consequence, $m - 1$ machines remain idle. Obviously, the jobs may or may not be preempted—this will not affect the schedule length unless there are idle times on the assigned machine. However, since preemptions cannot improve the schedule, it is reasonable not to take them into account. As a result, a special sequence, being a simplification of the maximal sequence $S\max$, will be used to represent an assignment of one machine to n jobs. It is simply a sequence of n one-element combinations, and will be denoted by Sn (see Fig. 1). Such a structure will be a base sequence in the case of convex processing speed functions.

Now, let us notice that, if the continuous resource amount allotted to job i does not change over the whole time of its execution, i.e., $u_i(t) = u_i$ for every t, we can write Eqn. (1) at the moment of completion of job i as

$$\frac{x_i(C_i)}{p_i} = f_i(u_i), \tag{5}$$

where p_i is the processing time of job i, and, in consequence

$$p_i = \frac{w_i}{f_i(u_i)}. \tag{6}$$

Since in the case considered u_i = 1 for each job i, $i = 1, 2, \ldots, n$, Eqn. (6) can be rewritten as

$$p_i = \frac{w_i}{f_i(1)}, \tag{7}$$

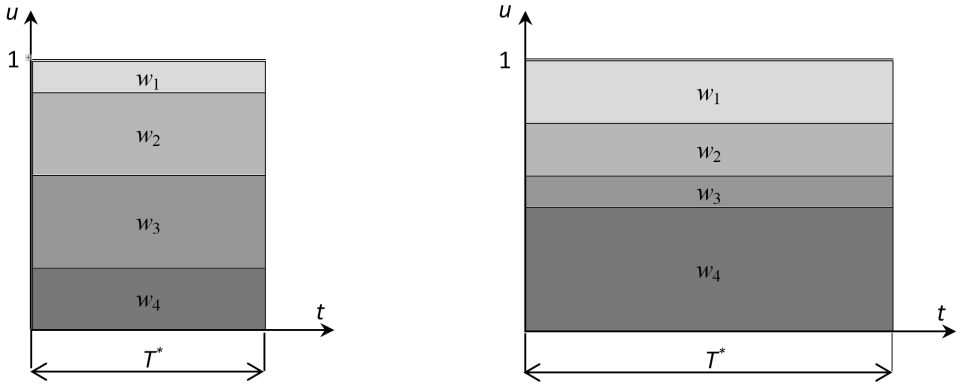
and the length of the optimal schedule can easily be calculated as

$$T^* = \sum_{i=1}^{n} \frac{w_i}{f_i(1)}. \tag{8}$$

Thus, for convex processing speed functions of jobs the problem is trivial since any schedule in which jobs are processed one after another in an arbitrary order, each of them using the total amount of the continuous resource, is optimal. The optimum makespan can be easily calculated from Eqn. (8).

It is also easy to see that in this case the number of machines is of no importance. The same sequential schedule leads to an optimum makespan in the absence (Corollary 1) or in the presence of any number of machines.

4.2. Concave processing speed functions. Based on Corollary 2, it is known that in this case parallel execution of jobs leads to optimal schedules. However, since a parallel assignment of machines to jobs can be restricted by the number of machines, we have to distinguish two cases: $n \leq m$ and $n > m$. In the first case the number of jobs does not exceed that of machines, and therefore all



(a) (b)

Fig. 2. Base sequence $S1 = [\{1, 2, 3, 4\}]$ for $n = 4$, $m \geq 4$, and two sample optimal schedules for different problem instances.

jobs can be performed in parallel. It is not possible in the second case, where only m out of n jobs can be scheduled in the first step, and $n - m$ jobs have to initially wait for machines. Since these two cases result in two completely different methodologies, we will discuss them in separate sections.

4.2.1. Case of $n \leq m$. As mentioned above, in this case all jobs can be performed in parallel, and the set of machines does not constitute any restriction. In consequence, the result presented in Corollary 2 can be implicitly applied, as if discrete resources were unlimited. Obviously, analogically as in Section 4.1, preemptions cannot improve the schedule and will not be taken into account. Consequently, the base sequence for this case is another special sequence being a simplification of $S\max$. It is simply one combination containing all jobs, denoted further by $S1$ (see Fig. 2). The optimum makespan can be calculated directly from Eqn. (4).

4.2.2. Case of $n > m$. In this case the number of machines restricts parallel assignments of machines to jobs, and a special methodology has to be developed. The base sequence in this case is the maximal sequence $S\max$ defined in Section 4. For this maximal sequence we look for a division of the sizes of jobs among combinations of the sequence that leads to optimum. More precisely, size w_i of each job i, $i = 1, 2, \ldots, n$, has to be divided into parts $w_{ik} \geq 0$ (unknown in advance) corresponding to particular time intervals (combinations), i.e., w_{ik} is part of job i processed in the interval associated with combination $Z_k, k = 1, 2, \ldots, s$. Such a division of sizes of jobs among successive intervals (combinations) is called a *size division* (see Fig. 3). The number of such divisions is, in general, infinite. Note that approaches based on searching for an optimal division of sizes (or processing times) of jobs in a schedule are often used to solve classical problems of scheduling preemptable jobs (Błażewicz *et al.*, 2007).

Now, an NLP problem can be formulated finding an optimal size division for the maximal sequence $S\max$, i.e., a division that leads to a schedule of the minimal length from among all schedules generated by $S\max$. In the problem the sum of the minimum-length intervals generated by consecutive combinations in $S\max$, as functions of the vector $\mathbf{w_k} = \{w_{ik}\}_{i \in Z_k}$, is minimized subject to the constraints that each activity has to be completed. Let $T_k^*(\mathbf{w_k})$ be the minimal length of the part of the schedule generated by $Z_k \in S\max$, and let K_i be the set of all indices of Z_k's such that $i \in Z_k$. The following NLP problem finds an optimal size division (and, in consequence, an optimal continuous resource allocation) for the maximal sequence $S\max$.

Problem PP. *Mimimize*

$$T = \sum_{k=1}^{s} T_k^*(\mathbf{w_k}) \tag{9}$$

subject to

$$\sum_{k \in K_i} w_{ik} = w_i, \quad i = 1, 2, \ldots, n, \tag{10}$$

$$w_{ik} \geq 0, \quad i = 1, 2, \ldots, n, \quad k \in K_i, \tag{11}$$

where $s = \binom{n}{m}$ and $T_k^*(\mathbf{w_k})$ is the unique positive root of the equation

$$\sum_{i \in Z_k} f_i^{-1}\left(\frac{w_{ik}}{T_k}\right) = 1 \tag{12}$$

if $\forall_{i \in Z_k} w_{ik} > 0$ or is equal to 0 otherwise.

The makespan T is calculated in (9) as the sum of the lengths of all the intervals of the schedule. The constraints (10) correspond to the condition of executing each job in its full size, whereas constraints (11) ensure that the w_{ik}'s are nonnegative. The condition (12) allows calculating the minimal length of the k-th interval following from an optimal continuous resource allocation.

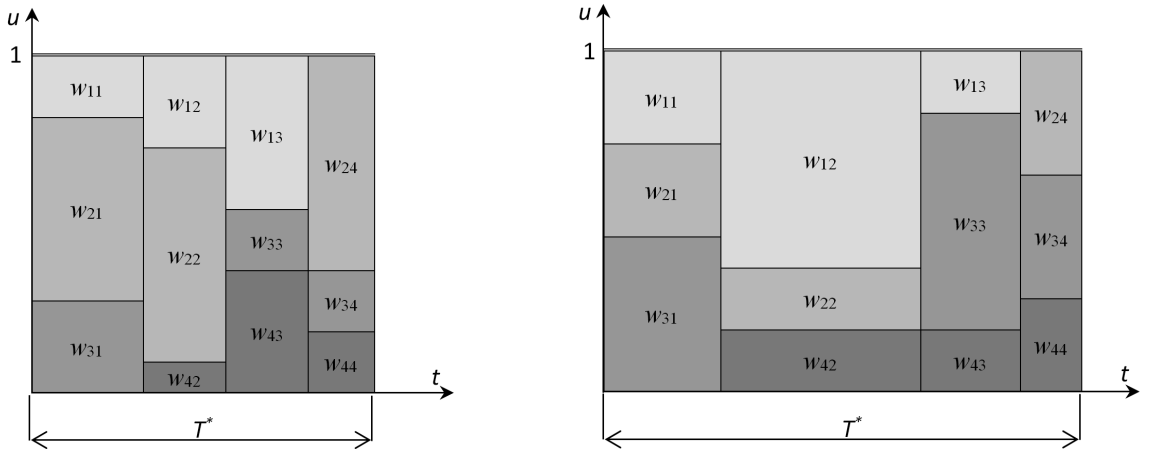


Fig. 3. Base sequence $S\max = [\{1,2,3\},\{1,2,4\},\{1,3,4\},\{2,3,4\}]$ for $n = 4$ and $m = 3$, and two sample optimal schedules for different problem instances.

The equation in (12) is an adaptation of Eqn. (4) to a single combination Z_k. It can be solved analytically for some important cases, as discussed in Section 2. Notice that this equation may be applied only if at least one w_{ik} is positive; otherwise, no part of any job is being executed in the k-th interval and the length of the interval is equal to zero.

Concluding, for concave processing speed functions of jobs and $n > m$, the approach consists of two steps:

- generating the maximal sequence $S\max$, and
- solving Problem PP for sequence $S\max$.

Let us finally stress that the relatively simple approaches presented in Section 4.1 and 4.2 can only be applied for the case with a constant amount of the continuous resource. When the resource amount varies over time, as described in Section 3, a new methodology has to be developed, which will be discussed in the next section.

5. Approach for a varying continuous resource amount

As mentioned in Section 4, a new methodology has to be proposed for the case when the available continuous resource amount varies over time. Developing the methodology is the main goal and contribution of this paper. However, the results presented in Section 4 for the constant continuous resource amount will be the foundation of the analyses carried out through the remainder of the paper.

Let us say again that, if it is known how to find an optimal schedule for a given base sequence, the problem boils down to looking for such a sequence that leads to a globally optimal solution. Base sequences of defined lengths guaranteeing finding optimal schedules are known *a priori* for some cases of discrete-continuous scheduling problems under a fixed (constant in time) continuous resource amount, as discussed in Section 4. Unfortunately, for the problem considered in this paper base sequences of finite length are not known in advance; however, they can be generated by some iterative procedures proposed in the following sections. The presented iterative approach to solving the problem is a general one; some differences in implementing the approach for particular special cases of the problem follow from a proper adjustment of the form of the base sequence and the formulation of the relevant MP problem. A common feature of the presented algorithms is the property that the total number of iterations, as well as the final length of the base sequence, depend on the problem instance.

The general idea underlying each of the proposed algorithms is similar. At the start (in the first iteration) an algorithm attempts to find an optimal schedule for a given base sequence and the amount U_1 of the continuous resource defined in RAP_1. If the length of the schedule found is not longer than the length Δ_1 of RAP_1, it may be considered optimal and the procedure stops. Otherwise, one of the following two situations occurs:

- the obtained schedule is infeasible since at a time the amount of the continuous resource used is greater than its total available amount (Fig. 4), or
- the obtained schedule is feasible but not optimal since at a time some amount of the continuous resource remains unused (Fig. 5).

If one of the above situations occurs, it is necessary to solve an MP problem in which it is assumed that the schedule is processed in two RAPs (RAP_1 and RAP_2). In the formulation of the problem one has to take into

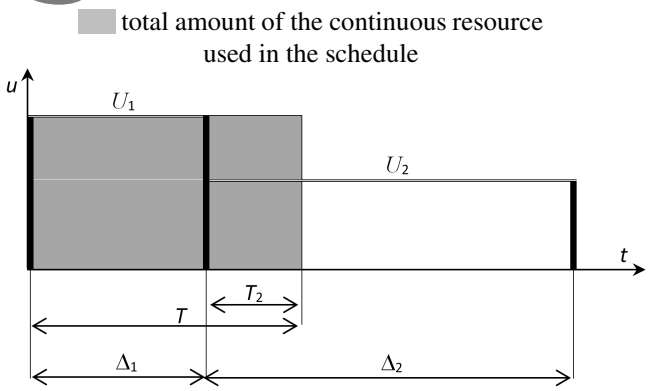


Fig. 4. Infeasible schedule due to excessive use of the resource in RAP_2.

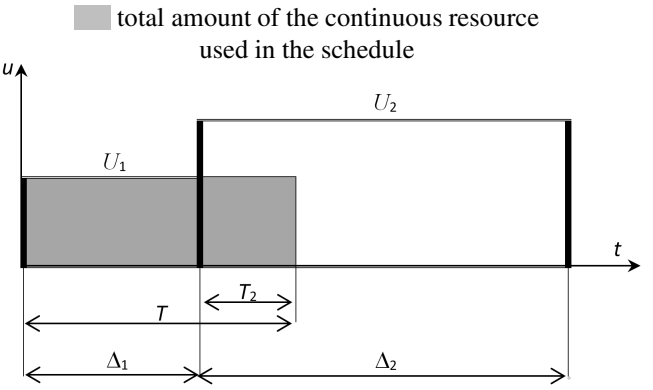


Fig. 5. Suboptimal schedule due to the unused amount of the available resource in RAP_2.

account that the processing of particular jobs may be now divided into two RAPs. Consequently, one has to consider a longer sequence of combinations, in this particular case a sequence being a concatenation of two base sequences. Generally, such a sequence, being a concatenation of a finite number of base sequences, will be called a *multi-sequence*.

Definition 4. A *multi-sequence* is a concatenation of a finite number of base sequences.

Such a multi-sequence will be represented as a vector of base sequences, i.e., in a general case, as a vector of vectors of combinations (see Section 4). For example, a multi-sequence being a concatenation of three maximal sequences $S\max$ for $n = 3$ and $m = 2$ will take the form

$$S = [[\{1,2\},\{1,3\},\{2,3\}],[\{1,2\},\{1,3\},\{2,3\}], [\{1,2\},\{1,3\},\{2,3\}]].$$

In order to generate multi-sequences of the desired form, we will use a *concatenation operator* $\oplus$. For example, the above sequence S can be obtained as

$$S = [[\{1,2\},\{1,3\},\{2,3\}],[\{1,2\},\{1,3\},\{2,3\}]] \oplus [\{1,2\},\{1,3\},\{2,3\}].$$

Now, if the solution of the formulated MP problem generates a schedule no longer than the sum of the lengths of the two RAPs considered (i.e. $\Delta_1 + \Delta_2$), the schedule is optimal. Otherwise, one has to consider the possibility of realizing the schedule in three RAPs, etc. It is easy to show that the discussed procedure always ends in a finite number of iterations; however, in each iteration both the number of variables and the number of constraints in the formulated MP problem grow.

6. Convex processing speed functions

As discussed in Section 4.1, for convex processing speed functions (already considered by Kis (2005)) in a makespan-optimal schedule jobs are processed sequentially, each of them using the total available amount of the continuous resource. Since in the problem considered a schedule is divided into parts according to the RAPs, the following issues should be stressed:

- parts of jobs in consecutive RAPs are processed sequentially with no idle times (see Property 2),
- each part of a job uses the total amount of the continuous resource available in the RAP it is executed in (Property 1),
- the order of processing parts of jobs in a RAP is arbitrary (jobs are independent).

It is known from Section 4.1 that the base sequence for this case is a sequence of n one-element combinations of jobs, i.e., sequence Sn. As a result, the multi-sequence considered here will be a concatenation of such base sequences.

Before presenting the algorithm, let us yet briefly discuss the notation used. In Section 4 variable w_{ik} was introduced, as part of job i processed in the interval associated with combination Z_k. However, in that case the amount of the continuous resource was constant over the entire schedule. Now, we have to distinguish parts of jobs processed in successive RAPs, since the amount of the continuous resource changes from one RAP to another.

Consequently, we introduce a variable w_{ih} which denotes a part of job i processed in RAP_h. Additionally, $T_h, h = 1,2,\ldots,H$, denotes the length of the schedule within RAP_h. Notice that, if the schedule cannot be completed in RAP_h, then T_h is simply equal to Δ_h. In consequence, the objective in the MP problem of Algorithm 1 presented below is the minimization of T_H, since the length of the schedule within the first $H-1$ RAPs is known in advance as $\sum_{h=1}^{H-1} \Delta_h$. It is easy to see that in this case the formulated MP problem is a linear

programming (LP) one, and therefore it is denoted by LP1.

Let us now present the algorithm for the case considered.

Algorithm 1.

Step 1. $H := 0; S := [\,]$;

Step 2. $H := H + 1; S := S \oplus Sn$;

Step 3. IF $H = 1$
THEN:
Calculate T_H^* as

$$T_H^* = \sum_{i=1}^{n} \frac{w_i}{f_i(U_1)}, \tag{13}$$

OTHERWISE:
Calculate T_H^* as an optimal objective function value of Problem LP1 for multi-sequence S.

Problem LP1. *Minimize*

$$T_H$$

subject to

$$T_h = \Delta_h, \quad h = 1, 2, \ldots, H-1, \tag{14}$$

$$\sum_{n=1}^{H} w_{ih} = w_i, \quad i = 1, 2, \ldots, n, \tag{15}$$

$$w_{ih} \geq 0, \quad i = 1, 2, \ldots, n, \quad h = 1, 2, \ldots, H, \tag{16}$$

where $T_h, h = 1, 2, \ldots, H$, is calculated as

$$T_h = \sum_{i=1}^{n} \frac{w_{ih}}{f_i(U_h)}; \tag{17}$$

Step 4. IF

$$T_H^* \leq \Delta_H \tag{18}$$

THEN:

$$T^* = \sum_{h=1}^{H-1} \Delta_h + T_H^* \tag{19}$$

and STOP;
OTHERWISE:
GOTO 2;

The idea of the presented Algorithm 1 is illustrated in Fig. 6. In each iteration (denoted by the value of parameter H) an optimal schedule is attempted to be constructed in a given number of RAPs. The algorithm starts with trying to schedule all jobs in RAP_1 ($H = 1$). The multi-sequence S is now equal to sequence Sn. Equation (13) in Step 3 is a modification of Eqn. (8), and allows finding the optimal length T_1^* of the schedule under the amount U_1 of the continuous resource available within RAP_1. This equation is only solved once (for $H = 1$). Then in Step 4 it is determined if the obtained schedule length T_1^* is not greater than Δ_1 (condition (18)). If this is the case, it means that all n jobs can be executed within RAP_1. In consequence, the optimal makespan is equal to the one calculated by Eqn. (13), and the algorithm stops. Otherwise, an optimal schedule requires more than one RAP to be processed. Thus, the algorithm goes back to Step 2, where the number of RAPs is increased by one, and now $H = 2$. The multi-sequence S becomes a concatenation of two sequences Sn since it is assumed that each job (or its part) may be processed in both RAP_1 and RAP_2. Consequently, a linear programming problem LP1 is solved which finds an optimal division of sizes of jobs among both the RAPs. It is natural that in $\text{RAP}_1, \ldots, \text{RAP}_{H-1}$ there is no idle time (Property 2), and therefore the conditions (14) are formulated in the form of equalities. The equalities (15) ensure that all the jobs are completed in the schedule built on the basis of the solution of Problem LP1, whereas the conditions (16) assure that all job parts are nonnegative. Equation (17) is an adaptation of Eqn. (13) to a single RAP_h. The schedule is feasible (and also optimal) if the length of its last part (related to RAP_H) is not longer than Δ_H. It is checked in Step 4 (condition (18)). If the obtained schedule is feasible, its optimal length is calculated by Eqn. (19), and the algorithm stops. Otherwise, the number of RAPs is incremented by one, and the procedure is repeated.

Let us emphasize that variables w_{ih} are unknowns in Problem LP1, while variables T_h are only auxiliary and can be calculated knowing the values of w_{ih}'s. Thus, Problem LP1 finds an optimal size division of jobs among all RAPs, i.e., a division leading to an optimal schedule, and, as a result, a minimal schedule length. The complexity of Algorithm 1 is polynomial since an LP problem is solved in Step 3 (Karmarkar, 1984). The number of variables in the formulated Problem LP1 is equal to $H \cdot n$ and grows linearly with each iteration of the algorithm.

7. Concave processing speed functions

As discussed in Section 4.2, for concave processing speed functions jobs should be executed in parallel, and therefore two cases $n \leq m$ and $n > m$, have to be analyzed separately.

7.1. Case of $n \leq m$. In contrast to the results presented in Section 4.2.1, now a schedule is divided into parts according to the RAPs. Consequently, the following

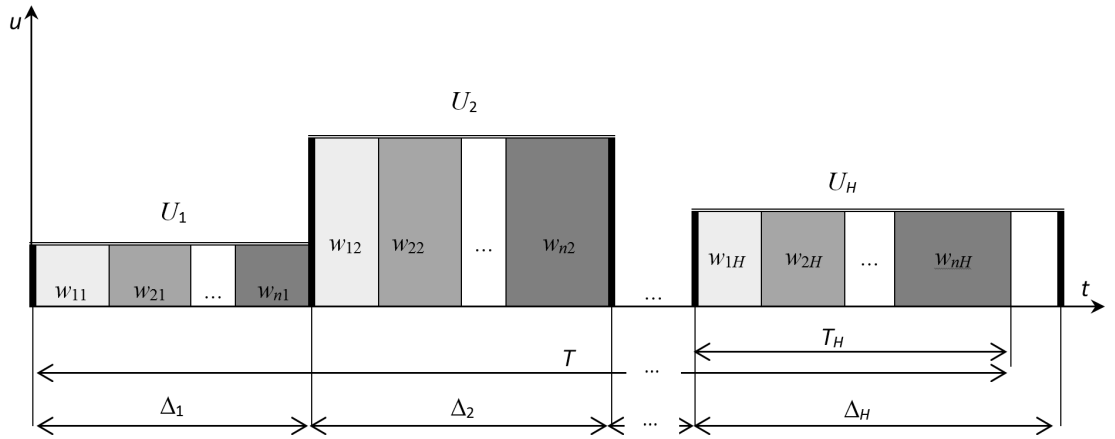


Fig. 6. Example of a potentially optimal schedule for convex processing speed functions and $S = [[\{1\}, \{2\}, \ldots, \{n\}], [\{1\}, \{2\}, \ldots, \{n\}], \ldots, [\{1\}, \{2\}, \ldots, \{n\}]]$.

issues should be addressed:

- parts of all jobs in consecutive RAPs are processed in parallel on n machines ($m - n$ machines remain idle) with no idle times (see Property 2),
- the job parts in any RAP start and finish at the same moment (Property 3),
- all job parts processed in parallel at any moment use the total amount of the continuous resource available in the RAP they are executed (in Property 1).

It is known from Section 4.2.1 that the base sequence for this case is just one combination containing all jobs, i.e., sequence $S1$. As a result, the multi-sequence considered here will be a concatenation of such base sequences. Figure 7 presents the idea of Algorithm 2 for the case considered, which is quite similar to the idea underlying Algorithm 1 discussed in Section 6. The substantial differences are the form of the base sequence which, in this case, is sequence $S1$, and the form of the equation finding an optimal length of a parallel (rather than sequential) schedule. In Algorithm 2 we use the same notation as in Algorithm 1.

Algorithm 2 operates similarly to Algorithm 1. However, Eqn. (20) differs from Eqn. (13) since it allows calculating an optimal length of a parallel schedule, and therefore it is a modification of Eqn. (4). Then, the objective function, as well as the constraints (21)–(23), in Problem NLP1 are identical as in Problem LP1. The condition (24) is different from Eqn. (17) since, as previously, it concerns calculating an optimal length of a parallel schedule, and therefore it is an adaptation of condition (12) to a single RAP$_h$. Notice that, since the equation in the condition (24) is nonlinear, the formulated MP problem is not linear anymore. The conditions (25) and (26) are, again, identical as in Algorithm 1.

The complexity of Algorithm 2 cannot be established in terms of combinatorial optimization since an NLP problem has to be solved in Step 3. Nevertheless, it is worth noting that the number of variables equal to $H \cdot n$, as well as the number of constraints equal to $(H \cdot n + H + n)$, in Problem NLP1 grow linearly with each iteration of Algorithm 2.

7.2. Case of $n > m$. As already known from Sect. 4.2.2, this case is computationally the most difficult one from among the cases considered in this paper. Although it allows applying an iterative approach similar to the previous ones, now an assignment of machines to jobs is defined by a multi-sequence composed of a number (nonzero and unknown in advance) of maximal sequences, each of them containing $s = \binom{n}{m}$ combinations of jobs. Such an extended multi-sequence requires generalized notation in order to properly define parameters of a schedule. Below we present the notation generalizing the one used in Sections 6 and 7.1:

Z_{hk} : k-th combination of the h-th maximal sequence $S\max$ (corresponding to RAP$_h$) in multi-sequence S,

K_{ih} : set of indices of combinations of the h-th maximal sequence (corresponding to RAP$_h$) in multi-sequence S that contain job i,

w_{ihk} : part of job i to be executed in the interval corresponding to combination Z_{hk},

T_{hk} : length of interval corresponding to combination Z_{hk} (known only after allocation of the continuous resource).

The above qualities are illustrated in Fig. 8.

Algorithm 3 presented below is based on the same idea underlying Algorithms 1 and 2. In this case, however, the algorithm starts from a solution of Problem PP

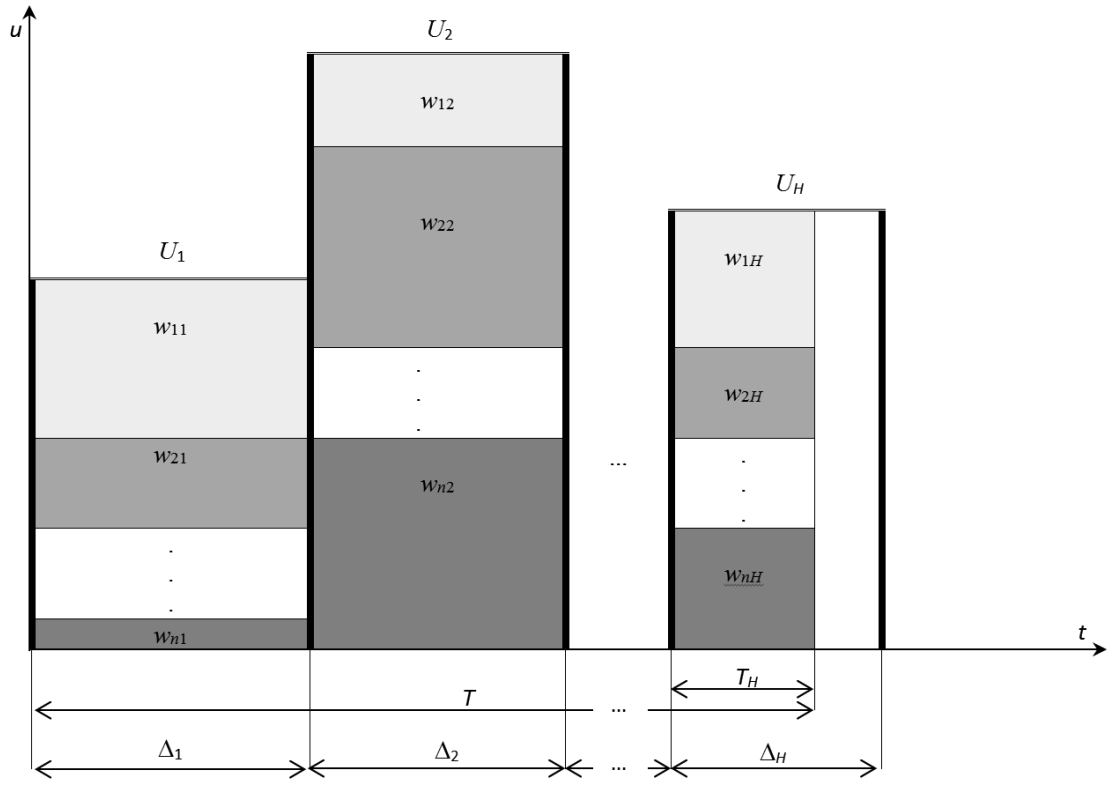


Fig. 7. Example of a potentially optimal schedule for concave processing speed functions and $S = [[\{1, 2, \ldots, n\}], [\{1, 2, \ldots, n\}], \ldots, [\{1, 2, \ldots, n\}]]$.

formulated in Section 4.2.2, with the only modification that the actual amount of the continuous resource (equal to U_1) available in RAP_1 has to be used in Eqn. (12) instead of the amount equal to 1, as assumed earlier. If the length of the obtained schedule is not longer than Δ_1, the schedule is optimal and the algorithm stops. Otherwise, another maximal sequence $S\max$ is added (by the concatenation operator $\oplus$) to the recent form of multi-sequence S, and the next iteration ($H := H + 1$) is performed. In this step Problem NLP2 is solved for the new form of multi-sequence S. The algorithm terminates when the schedule constructed in its successive iteration is not longer than $\Delta_1 + \Delta_2 + \cdots + \Delta_H$.

As was the case for Algorithm 2, also the complexity of Algorithm 3 cannot be established in terms of combinatorial optimization because of the presence of Problem NLP2 solved in Step 3. However, it can be stated that the number of variables in NLP2 is equal to $H \cdot n \cdot \binom{n}{m}$, whereas the number of constraints is equal to $(H \cdot n \cdot \binom{n}{m} + H + n)$. Both these numbers grow linearly with each iteration of Algorithm 3.

8. Conclusions

In this paper a problem of scheduling preemptable, independent jobs on parallel, identical machines under an additional continuous, renewable resource to minimize the schedule length has been considered. Such problems where jobs simultaneously require for their execution discrete and continuous resources are generally called discrete-continuous scheduling problems. In all previous works in that field it was assumed that the total amount of the continuous resource available at a time was constant. In this paper we have relaxed this assumption, and allowed the total available amount of the continuous resource to periodically vary over time. The lengths of the resource availability periods, as well as the amounts (nonnegative and constant) of the resource in each period, are known in advance. It is an important generalization from the practical point of view.

For the problem considered we have presented three exact iterative algorithms based on the general methodology developed for the problem and discussed in the paper. The first one, named Algorithm 1, has been designed for convex processing speed functions of jobs. In Algorithm 1 an LP problem has to be solved, and therefore the complexity of the algorithm is polynomial. The second algorithm (Algorithm 2) has been constructed for the case of concave processing speed functions and the number of jobs not greater than that of machines. In Algorithm 2 an NLP problem is to be solved in each iteration. The third and most

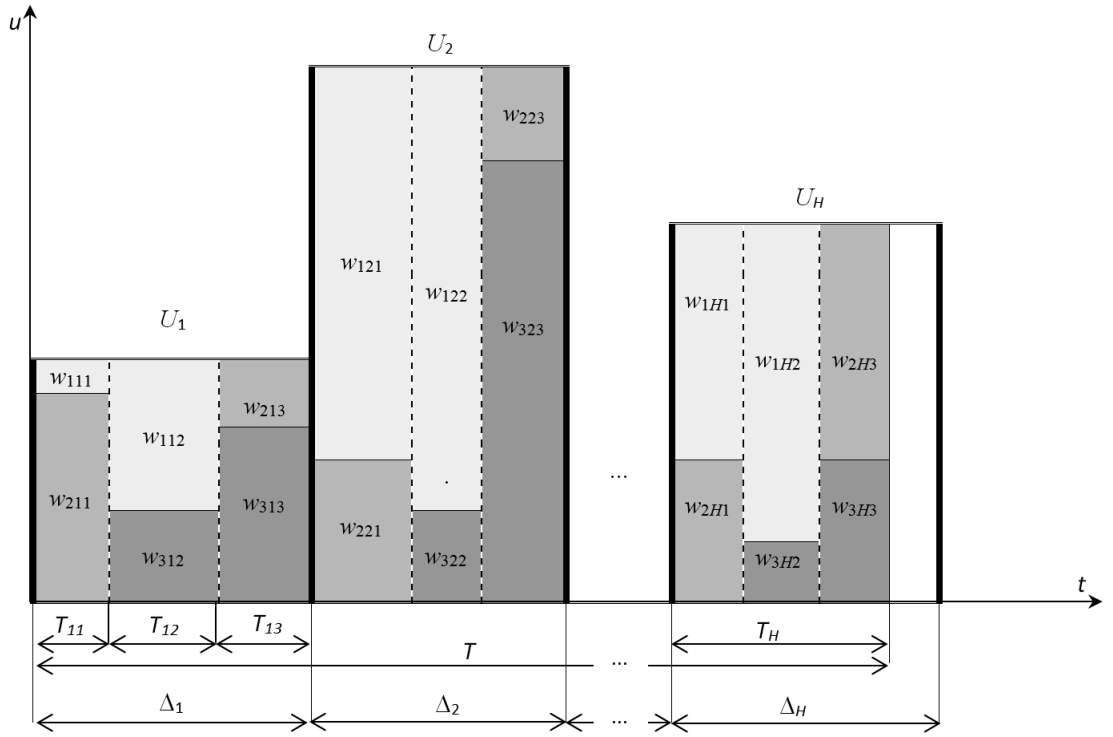


Fig. 8. Example of a potentially optimal schedule for concave processing speed functions and $n = 3$, $m = 2$ and $S = [[Z_{11}, Z_{12}, Z_{13}], [Z_{21}, Z_{22}, Z_{23}], \ldots, [Z_{H1}, Z_{H2}, Z_{H3}]] = [[\{1,2\}, \{1,3\}, \{2,3\}], [\{1,2\}, \{1,3\}, \{2,3\}], \ldots, [\{1,2\}, \{1,3\}, \{2,3\}]]$.

complex algorithm (Algorithm 3) has also been designed for concave processing speed functions, but for the case when there are more jobs than machines. This is the most general case, and therefore the NLP problem solved in Algorithm 3 is computationally most difficult. The number of variables, as well as that of constraints, in Problem NLP2 grows exponentially with the number of jobs.

In the future we plan to carry out the research in a few directions. The first one will be focused on searching for special cases for which optimal solutions can be found easier than by using the general approaches presented in this paper. In particular, taking into account some regularities in changes in the continuous resource availability seems interesting. The second direction that naturally comes to mind is constructing heuristic algorithms for those cases where solving complex mathematical programming problems is especially computationally demanding. In such cases it seems reasonable and justified to use an alternative two-phase approach in which, firstly, the continuous resource is allocated to jobs, and, secondly, jobs (with already known processing times) are scheduled on machines. Another interesting direction seems to be an approach using an upper bound on the minimal value of H (the number of resource availability periods in which all jobs are completed). Slight reformulations of the proposed NLP problems could allow finding the appropriate value of H by, e.g., the binary search method. In this method the initial bottom and top ranges of the search interval would be set, respectively, at 1 and the upper bound on minimum H. Such an approach could appear to be more efficient than the iterative algorithms proposed in the paper. However, what impact on the efficiency of the approach the accuracy of the evaluation of the upper bound on minimum H has remains now an open question.

Acknowledgment

This research is a part of the project no. 2013/08/A/ST6/00296 funded by the Polish National Science Centre.

References

Błażewicz, J., Ecker, K., Schmidt, G., Pesch, E. and Węglarz, J. (2007). *Handbook of Scheduling: From Theory to Applications*, Springer, Berlin.

Gorczyca, M. and Janiak, A. (2010). Resource level minimization in the discrete-continuous scheduling, *European Journal of Operational Research* **203**(1): 32–41.

Janiak, A. (1991). Single machine scheduling problem with a common deadline and resource dependent

Algorithm 2.

Step 1. $H := 0; S := [\,];$

Step 2. $H := H + 1; S := S \oplus S1;$

Step 3. IF $H = 1$
THEN:
Calculate T_H^* as the only positive root of the equation

$$\sum_{i=1}^{n} f^{-1}\left(\frac{w_i}{T_H}\right) = U_1, \tag{20}$$

OTHERWISE:
Calculate T_H^* as an optimal objective function value of Problem NLP1 for multi-sequence S.

Problem NLP1. *Minimize*

$$T_H$$

subject to

$$T_h = \Delta_h, \quad h = 1, 2, \ldots, H-1, \tag{21}$$

$$\sum_{n=1}^{H} w_{ih} = w_i, \quad i = 1, 2, \ldots, n, \tag{22}$$

$$w_{ih} \geq 0, \quad i = 1, 2, \ldots, n, \quad h = 1, 2, \ldots, H, \tag{23}$$

where $T_h, h = 1, 2, \ldots, H$, is the unique positive root of the equation

$$\sum_{i=1}^{n} f_i^{-1}\left(\frac{w_{ih}}{T_h}\right) = U_h \tag{24}$$

if $\forall_i w_{ih} > 0$ or is equal to 0 otherwise;

Step 4. IF

$$T_H^* \leq \Delta_H \tag{25}$$

THEN:

$$T^* = \sum_{h=1}^{H-1} \Delta_h + T_H^* \tag{26}$$

and STOP;
OTHERWISE:
GOTO 2;

Algorithm 3.

Step 1. $H := 0; S := [\,];$

Step 2. $H := H + 1; S := S \oplus S\max;$

Step 3. IF $H = 1$
THEN:
Calculate T_H^*, an optimal objective function value of Problem PP (in which the value of 1 is replaced by U_1 in Eqn. (12) for multi-sequence S
OTHERWISE:
Calculate T_H^* as an optimal objective function value of Problem NLP2 for multi-sequence S.

Problem NLP2. *Minimize*

$$T_H = \sum_{k=1}^{s} T_{Hk}$$

subject to

$$\sum_{k=1}^{s} T_{hk} = \Delta_h, \quad h = 1, 2, \ldots, H-1, \tag{27}$$

$$\sum_{h=1}^{H} \sum_{k \in K_{ih}} w_{ihk} = w_i, \quad i = 1, 2, \ldots, n, \tag{28}$$

$$\begin{aligned} w_{ihk} \geq 0, \quad & i = 1, 2, \ldots, n, \\ & h = 1, 2, \ldots, H, \\ & k = 1, 2, \ldots, s, \end{aligned} \tag{29}$$

where $T_{hk}, h = 1, 2, \ldots, H$ and $k = 1, 2, \ldots, s$, is the unique positive root of the equation

$$\sum_{i \in Z_{hk}} f_i^{-1}\left(\frac{w_{ihk}}{T_{hk}}\right) = U_h \tag{30}$$

if $\forall_{i \in Z_{hk}} w_{ihk} > 0$ or is equal to 0 otherwise;

Step 4. IF

$$T_H^* \leq \Delta_H \tag{31}$$

THEN:

$$T^* = \sum_{h=1}^{H-1} \Delta_h + T_H^* \tag{32}$$

and STOP;
OTHERWISE:
GOTO 2;

release dates, *European Journal of Operational Research* **53**(3): 317–325.

Karmarkar, N.K. (1984). A new polynomial time algorithm for linear programming, *Combinatorica* **4**(4): 373–395.

Kis, T. (2005). A branch-and-cut algorithm for scheduling of projects with variable-intensity activities, *Mathematical Programming* **103**(3): 515–139.

Leachman, R.C. (1983). Multiple resource leveling in construction systems through variation of activity intensities, *Naval Research Logistics Quarterly* **30**(3): 187–198.

Leachman, R.C., Dincerler, A. and Kim, S. (1990). Resource-constrained scheduling of projects with variable-intensity activities, *IIE Transactions* **22**(1): 31–39.

Różycki, R. and Węglarz, J. (2012). Power-aware scheduling of preemptable jobs on identical parallel processors to meet deadlines, *European Journal of Operational Research* **218**(1): 68–75.

Waligóra, G. (2011). Heuristic approaches to discrete-continuous project scheduling problems to minimize the makespan, *Computational Optimization and Applications* **48**(2): 399–421.

Waligóra, G. (2014). Discrete-continuous project scheduling with discounted cash inflows and various payment models—a review of recent results, *Annals of Operations Research* **213**(1): 319–340.

Węglarz, J. (1976). Time-optimal control of resource allocation in a complex of operations framework, *IEEE Transactions on Systems, Man and Cybernetics* **6**(11): 783–788.

Rafał Różycki, Ph.D., D.Sc., graduated from the Poznań University of Technology in computing science in 1994. Since 1994 he has been working there at the Institute of Computing Science (Laboratory of Operational Research and Artificial Intelligence), currently as an assistant professor. In 2000 he received a Ph.D. degree in computing science from the Poznań University of Technology, and published his postdoctoral thesis on scheduling computational jobs in 2014. His main areas of interest are power-aware scheduling, discrete-continuous scheduling, combinatorial optimization and metaheuristic algorithms. He is the author or a co-author of over 70 scientific publications in international journals, monographs and conference proceedings, and has presented his results at 50 international and domestic scientific conferences and workshops.

Grzegorz Waligóra, Ph.D., D.Sc., graduated from the Poznań University of Technology in computing science in 1994. Since 1994 he has been working there at the Institute of Computing Science (Laboratory of Operational Research and Artificial Intelligence), currently as an assistant professor. In 2000 he received a Ph.D. degree in computing science from the Poznań University of Technology, and published his postdoctoral thesis on discrete-continuous project scheduling in 2009. His main areas of interest are project and machine scheduling, discrete-continuous scheduling, combinatorial optimization, metaheuristic algorithms, resource management and scheduling in computational grids. He is the author or a co-author of over 80 scientific publications in international journals, monographs and conference proceedings and presented his results at 45 international and domestic scientific conferences and workshops. He is a laureate of the Foundation for Polish Science Award for Young Researches (1999) and of the Prime Minister of Poland Award for a Ph.D. dissertation (2000).

Jan Węglarz, academician, professor (Ph.D. in 1974, D.Sc. in 1977), in the years 1978–83 an associate professor and then a professor in the Institute of Computing Science at the Poznań University of Technology, a member of the Polish Academy of Sciences (PAS), the director of the Institute of Computing Science of the Poznań University of Technology and its predecessors since 1987, the director of the Poznań Supercomputing and Networking Center. A member of several editorial boards, a representative of Poland in the Board of Representatives of IFORS and in the EURO Council (president of EURO in 1997–1998). A member of several professional and scientific societies, including the American Mathematical Society and the Operations Research Society of America. The author or a co-author of 11 monographs, 3 textbooks (3 editions each), and over 200 papers in major professional journals and conference proceedings. A frequent visitor in major research centers in Europe and the USA. A co-laureate of the State Award (1988) and the EURO Gold Medal (1991), a laureate of the Foundation for Polish Science Award (2000).

Received: 7 September 2015
Revised: 1 February 2016
Re-revised: 10 March 2016
Accepted: 18 May 2016