Comput Optim Appl (2011) 48: 399–421
DOI 10.1007/s10589-010-9343-5

Heuristic approaches to discrete-continuous project scheduling problems to minimize the makespan

Grzegorz Waligóra

Received: 1 July 2009 / Published online: 3 August 2010

Abstract In this paper some discrete-continuous project scheduling problems to minimize the makespan are considered. These problems are characterized by the fact that activities of a project simultaneously require for their execution discrete and continuous resources. A class of these problems is considered where the number of discrete resources is arbitrary, and one continuous, renewable, limited resource occurs. A methodology for solving the defined problems is presented. The continuous resource allocation problem is analyzed. An exact, as well as a heuristic approach to the problem is discussed. The idea of the continuous resource discretization is described, and a special case of the problem with identical processing rate functions is analyzed. Some computational experiments for evaluating the efficiency of the proposed heuristic approaches are presented. Conclusions and directions for future research are given.

Keywords Project scheduling · Discrete resource · Continuous resource · Makespan · Heuristic

1 Introduction

In the classical resource-constrained project scheduling problem (RCPSP) (see [3]) it is assumed that resources can be assigned to activities only in amounts from a given set (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 activities in arbitrary numbers from a given interval (i.e. in real numbers). Such resources are called *continuous* (or continuously-divisible). Continuous resources occur when,

G. Waligóra (✉)
Institute of Computing Science, Poznań University of Technology, Piotrowo 2, 60-965 Poznań, Poland
e-mail: grzegorz.waligora@cs.put.poznan.pl

e.g., activities are processed by parallel processing units driven by a common (electric, pneumatic, hydraulic) power source, like commonly supplied grinding or mixing machines, electrolytic tanks, or refuelling terminals. Also in computer systems, where multiple processors share a common primary memory, if it is a paged-virtual memory system and the number of pages goes into hundreds, then the primary memory can be treated as a continuous resource [20]. On the other hand, the processors themselves can be considered as a continuous resource in scalable (SPP) or massively parallel (MPP) systems when the number of them is huge (hundreds or even thousands). Another example concerning the forging process in steel plants was analyzed in [6]. Forgings are preheated by gas up to an appropriate temperature in forge furnaces. Gas flow intensity, limited for the whole battery of forge furnaces, is a continuous resource.

In general, two activity processing models have been proposed in the literature. The first one, examined among others in [7], defines the activity duration as a function of the amount of the continuous resource allotted to this activity. This *processing time vs. resource amount* model is a straightforward generalization of the discrete time-resource trade-off model. It is implicitly assumed that the resource amount allocated to an activity does not change during its execution. In the second model, the processing rate of an activity is a function of the amount of the continuous resource allotted to this activity at a time. In this case, the amount of the continuous resource allotted to an activity may change during its execution. This model was proposed in [2] and further examined in [21, 22] for preemptable schedules. Discrete-continuous project scheduling problems arise when activities simultaneously require discrete and continuous resources. The methodology for discrete-continuous machine scheduling based on the *processing rate vs. resource amount* model was described in [9], and a model of the discrete-continuous project scheduling problem was first presented in [10].

In this paper a class of these problems is considered where the number of discrete resources is arbitrary (i.e. it is a part of the problem instance), and there is one continuous renewable resource whose total amount available at a time is limited. Activities are non-preemptable and precedence-related, and all are available at the start of the project. The objective is to minimize the makespan. The problem can be decomposed into two inter-related subproblems: (i) to find a precedence- and discrete resource-feasible schedule of activities (the discrete part), and (ii) to allocate the continuous resource among activities in the schedule (the continuous part). Some properties of the continuous part of the problem were shown in [10] for certain classes of processing rate functions of activities. Namely, for linear functions and all functions less than linear (including convex functions) the problem is trivial, whereas for concave functions it requires a solution of a convex mathematical programming problem.

The main goal of this paper is to present different approaches to solving the continuous part of the problem. The first one is an exact solution of the convex problem, already presented in [10] in the context of a local search metaheuristic approach. However, since existing exact methods are unable to handle large problem instances, a heuristic approach to the continuous resource allocation has been developed. The heuristic proposed in this paper is a generalization of a heuristic procedure analyzed in [8, 13, 18] for discrete-continuous machine scheduling. The generalization consists

in adapting the procedure to a general discrete-continuous project scheduling problem, in which there can be many discrete-resources (not a set of machines only) and the solution representation is different. In this paper the computational results of the new generalized heuristic are presented, and its performance analysis for instances up to 20 activities is discussed. Another approach consists in the discretization of the continuous resource leading to the purely discrete resource-constrained project scheduling problem with multiple execution modes (MRCPSP). This idea was first proposed in [11], where the simulated annealing metaheuristic was computationally tested, using a simple discretization procedure as a part of the objective function. In this work the continuous resource discretization is discussed as a heuristic approach to solve the general problem considered, and a numerical example is given showing the procedure of obtaining a discrete multi-mode problem under linear and power processing rate functions of activities. Finally, a new aspect of the problem is considered in this paper. It will be shown, namely, that if the processing rate functions of all activities are identical and concave, the process of finding an optimal solution of the discrete-continuous problem can be reduced to searching for an optimal solution of the corresponding discrete problem.

The paper is organized as follows. In Sect. 2 the considered problem is mathematically formulated. A methodology for solving the problem is presented in Sect. 3. Section 4 concerns the heuristic continuous resource allocation, whereas the discretization of the continuous resource is described in Sect. 5. Section 6 reports the case of identical processing rate functions of activities. Computational experiments for evaluating the efficiency of the proposed heuristic approaches are presented in Sect. 7. Some conclusions and directions for future research are given in Sect. 8.

2 Problem formulation

The discrete-continuous resource-constrained project scheduling problem (DCR-CPSP) is defined as follows. A project consists of n precedence-related, non-preemptable activities, which require renewable resources of two types: discrete and continuous ones. A set K_R of discrete resources is given, and r_{il}, $i = 1, 2, \ldots, n$; $l = 1, 2, \ldots, |K_R|$, is the discrete resource request of activity A_i for resource l. The total number of units of discrete resource l, $l = 1, 2, \ldots, |K_R|$, available in each time period is R_l. The single continuous renewable resource can be allotted to activities in (arbitrary) amounts from the interval [0, 1], and its total amount available at a time is equal to 1. The amount $u_i(t)$ (unknown in advance) of the continuous resource allotted to activity A_i at time t determines the processing rate of activity A_i, as it is described by the following equation [9]:

$$\dot{x}_i(t) = \frac{dx_i(t)}{dt} = f_i[u_i(t)], \qquad x_i(0) = 0, \qquad x_i(C_i) = \tilde{x}_i \tag{1}$$

where $x_i(t)$ is the state of activity A_i at time t, f_i is a continuous increasing function, $f_i(0) = 0$, $u_i(t)$ is the amount of the continuous resource allotted to activity A_i at time t, C_i is the completion time (unknown in advance) of activity A_i, $\tilde{x}_i$ is the processing demand (final state) of activity A_i.

Table 1 Parameters of the DCRCPSP

Symbol	Definition
$G(V, E)$	directed AoN graph representing the structure of the project
V	set of activities
$n = \|V\|$	number of activities
$A_i \rightarrow A_j$	precedence relation between activities A_i and A_j
E	set of precedence constraints between activities
$Pred_i$	set of direct predecessors of activity A_i
$Succ_i$	set of direct successors of activity A_i
K_R	set of discrete renewable resources
$\|K_R\|$	number of discrete renewable resources
R_l	number of units of discrete resource l available in each time period
r_{il}	request for discrete resource l by activity A_i
f_i	processing rate function of activity A_i
$\tilde{x}_i$	processing demand of activity A_i
S_i	starting time of activity A_i
C_i	completion time of activity A_i
$\mathbf{u}(t) = [u_1(t), u_2(t), \ldots, u_n(t)]$	continuous resource allocation
$(S_1, S_2, \ldots, S_n)$	schedule

State $x_i(t)$ of activity A_i at time t is an objective measure of work related to the processing of activity A_i up to time t. It may denote, e.g., the number of man-hours already spent on processing activity A_i, the number of standard instructions in processing computer program A_i, etc.

A project is represented by an activity-on-node (AoN) directed, acyclic, and topologically ordered graph $G(V, E)$, where the set of nodes V corresponds to the set of activities, and the set of arcs E represents precedence constraints. The activities are subject to finish-to-start precedence constraints with zero minimum time lags. The precedence constraints of activity A_i with other activities are defined by two sets: a set $Pred_i$ of direct predecessors of activity A_i, and a set $Succ_i$ of direct successors of activity A_i. Thus, each activity of the project is characterized by its processing demand, processing rate function, discrete resource requests, and precedence constraints with other activities. It is assumed that all activities and resources are available from the start of the project. The problem is to find a precedence- and discrete resource-feasible schedule and, simultaneously, a continuous resource allocation that minimize the makespan. The continuous resource allocation is defined by a piecewise continuous, non-negative vector function $\mathbf{u}(t) = [u_1(t), u_2(t), \ldots, u_n(t)]$ whose values $\mathbf{u}^* = [u_1^*, u_2^*, \ldots, u_n^*]$ are (continuous) resource allocations corresponding to C_{max}^*—the minimal value of C_{max}. Completion of activity A_i requires that:

$$x_i(C_i) = \int_0^{C_i} f_i[u_i(t)]dt = \tilde{x}_i \tag{2}$$

All the parameters of the DCRCPSP are summarized in Table 1.

Using the project scheduling classification scheme presented in [5], the notation of the considered problem is:

$$m, 1|cpm,\ cont|C_{max}$$

3 Methodology for discrete-continuous project scheduling

The general methodology for discrete-continuous project scheduling [10] uses the idea of a *feasible sequence* introduced in [9] for machine scheduling. Observe that a feasible schedule (i.e. a solution of a discrete-continuous project scheduling problem) can be divided into $s \leq n$ intervals of lengths M_k, $k = 1, 2, \ldots, s$, defined by the completion times of the consecutive activities. Let Z_k denote the combination of activities processed in parallel in the k-th interval. Thus, a feasible sequence $\boldsymbol{S}$ of combinations Z_k, $k = 1, 2, \ldots, s$, is associated with each feasible schedule. The feasibility of such a sequence requires that:

- each activity appears in at least one combination, i.e.:

$$\underset{i\in\{1,\ldots,n\}}{\forall}\ \underset{k\in\{1,\ldots,s\}}{\exists} A_i \in Z_k$$

- non-preemptability of each activity is guaranteed, i.e.:

$$\underset{i\in\{1,\ldots,n\}}{\forall}\ \underset{(k,m):A_i\in Z_k, A_i\in Z_m, m\geq k}{\forall} (k = m) \vee (A_i \in Z_l,\ l = k+1, \ldots, m-1)$$

  which means that each activity appears in exactly one or in successive combinations in $\boldsymbol{S}$
- precedence constraints between activities are satisfied, i.e.:

$$\underset{(A_i,A_j)\in V:A_i\to A_j}{\forall} (A_i \in Z_k \wedge A_j \in Z_l) \Rightarrow (l > k)$$

- the number of units of each discrete resource l, $l = 1, 2, \ldots, |K_R|$, assigned to all activities in combination Z_k, $k = 1, 2, \ldots, s$, does not exceed R_l, i.e.:

$$\underset{l\in\{1,\ldots,|K_R|\}}{\forall}\ \underset{k\in\{1,\ldots,s\}}{\forall} \sum_{A_i\in Z_k} r_{il} \leq R_l$$

Example 1 Consider a 9-activity project in which the precedence constraints are: $1 \to 2, 1 \to 3, 1 \to 4, 2 \to 5, 3 \to 7, 4 \to 6, 5 \to 7, 5 \to 8, 6 \to 7, 6 \to 9, 7 \to 9$, and $8 \to 9$. Assume that there is one discrete resource ($|K_R| = 1$) available in 4 units ($R_1 = 4$). The resource requests of activities are defined by vector $\mathbf{r}_1 = [3, 2, 1, 1, 3, 1, 4, 2, 1]$. For simplicity, in a feasible sequence each activity will be associated with its index, i.e. activity A_i will be called shortly activity i. There are many feasible sequences for this exemplary project. One of them is, e.g.:

$$\boldsymbol{S} = \{1\}, \{2, 3, 4\}, \{3, 4\}, \{3, 5\}, \{5, 6\}, \{6, 8\}, \{7\}, \{9\}$$

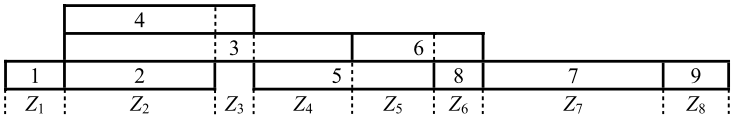

Fig. 1 A schedule corresponding to feasible sequence $\boldsymbol{S}$

The above form of a feasible sequence means that activity A_1 is processed alone in the first interval, and this interval ends with the completion of this activity. The corresponding combination $Z_1 = \{1\}$. Then activities A_2, A_3, and A_4 are processed in parallel, and the completion of activity A_2 ends the second interval. The corresponding combination $Z_2 = \{2, 3, 4\}$. Next, only activities A_3 and A_4 are processed in the third interval, since the resource request of activity A_5 does not allow it to be executed in this interval (although all its direct predecessors are finished). Thus, the corresponding third combination is $Z_3 = \{3, 4\}$. All the next combinations also fulfill the precedence and resource constraints. The last interval ends with the completion of activity A_9, and this is the end of the schedule. The corresponding last combination is $Z_8 = \{9\}$.

It is worth mentioning that there is a sequence of combinations feasible for each project. This sequence is:

$$\{1\}, \{2\}, \ldots, \{n\}$$

which means that the activities are processed sequentially one after another in the order fulfilling the precedence constraints (graph G is, as assumed, topologically ordered). The three remaining feasibility conditions are also satisfied by this sequence. This special sequence will be called *ascending feasible sequence*, and will be used in some cases.

Continuing Example 1, it is now possible to show a schedule corresponding to feasible sequence $\boldsymbol{S}$. This schedule is presented in Fig. 1.

It is important to stress that at this moment the actual durations of activities are still unknown, since the continuous resource has not yet been allocated. However, the form of a feasible sequence gives the information which activities are processed in parallel in consecutive intervals. The continuous resource will be then allocated on a basis on that information. The way of allocating the continuous resource depends on the form of the activity processing rate functions. Below this issue is discussed for two important cases: all functions not greater than a linear function, including linear and convex functions (Sect. 3.1), and concave processing rate functions (Sect. 3.2).

3.1 Processing rate functions not greater than linear

It has been proved in [9] that for processing rate functions f_i such that $f_i \leq c_i u_i$, $c_i = f_i(1)$, $i = 1, 2, \ldots, n$, a makespan optimal schedule for the DCRCPSP is obtained by allotting the total amount of the continuous resource to one activity at a time only. Notice that this class of functions includes linear ones which is a very important case in practice, where the increase of the processing rate of an activity is directly proportional to the amount of the continuous resource allotted to this activity at a

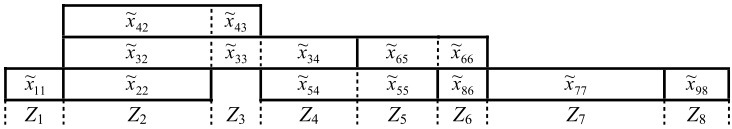

Fig. 2 A division of processing demands of activities for feasible sequence $\boldsymbol{S}$

time. However, all functions less than linear also belong to this class, including all convex functions important from the mathematical point of view.

Consequently, for such functions, in an optimal schedule activities are performed sequentially one after another in an order fulfilling the precedence constraints, and the total amount of the continuous resource is allotted to each activity. Of course, if the condition $r_{il} \leq R_l$, $i = 1, 2, \ldots, n$, $l = 1, 2, \ldots, |K_R|$ holds, then the discrete resource constraints are not violated because only one activity is performed at a time. It is easy to see that in this case the ascending feasible sequence is one of the sequences leading to an optimal schedule. Of course, the same optimal value of the makespan can be obtained by any other feasible sequence of n combinations composed of one activity each, and fulfilling the precedence constraints.

3.2 Concave processing rate functions

In practice it is often the case that, after the initial acceleration, it comes to a point where additional increase of the resource amount does not result in the corresponding increase of the processing rate of an activity. It means, the processing rate function saturates at that point and does not increase that rapidly, i.e. it has a concave form. For example, concave processing rate functions are more adequate for the majority of real-life large-scale parallel computations because the efficiency of task processing degrades while the number of allocated processors increases due to communication delays. Concave processing rates correspond to the realistic hypothesis of parallelizing actual large numerical codes (see, e.g., [1]).

Now, it was proved in [9] that for concave processing rate functions f_i, $i = 1, 2, \ldots, n$, a parallel configuration is optimal in the DCRCPSP. It means that in a makespan optimal schedule as many activities as possible are performed in parallel. The methodology developed in [9] for machine scheduling, and then discussed in [10] for project scheduling, shows that in this case an optimal division of processing demands of activities $\tilde{x}_i$, $i = 1, 2, \ldots, n$, among combinations in $\boldsymbol{S}$ can be found for a given feasible sequence $\boldsymbol{S}$. Such a division leads to a minimum-length schedule from among all feasible schedules generated by $\boldsymbol{S}$. More precisely, since an activity can be processed in several intervals, its processing demand has to be divided among these intervals. The number of such divisions is, in general, infinite. Figure 2 shows the scheme of such a division of processing demands of activities among successive intervals (combinations) for the feasible sequence $\boldsymbol{S}$ from Example 1.

Obviously, the sum of all parts of the processing demand of an activity must be equal to its total processing demand, i.e.:

$$\tilde{x}_{11} = \tilde{x}_1$$

$$\begin{aligned}
\tilde{x}_{22} &= \tilde{x}_2 \\
\tilde{x}_{32} + \tilde{x}_{33} + \tilde{x}_{34} &= \tilde{x}_3 \\
\tilde{x}_{42} + \tilde{x}_{43} &= \tilde{x}_4 \\
\tilde{x}_{54} + \tilde{x}_{55} &= \tilde{x}_5 \\
\tilde{x}_{65} + \tilde{x}_{66} &= \tilde{x}_6 \\
\tilde{x}_{77} &= \tilde{x}_7 \\
\tilde{x}_{86} &= \tilde{x}_8 \\
\tilde{x}_{98} &= \tilde{x}_9
\end{aligned}$$

Now, as already mentioned, an optimal division minimizing the schedule length can be found for a given feasible sequence $\boldsymbol{S}$ and concave processing rate functions. To this end, a non-linear mathematical programming problem can be formulated in which the sum of the minimum-length intervals (i.e. parts of a feasible schedule) generated by consecutive combinations in $\boldsymbol{S}$, as functions of the $\{\tilde{x}_{ik}\}_{A_i \in Z_k}$ where $\tilde{x}_{ik}$ is a part of activity A_i processed in combination Z_k, is minimized subject to the constraints that each activity has to be completed. Then it was also proved that for concave functions it is sufficient to consider only feasible schedules in which the resource allocations among activities remain constant in each interval k, $k = 1, 2, \ldots, s$. More precisely, knowing processing demands $\tilde{x}_{ik}$ of activities executed in the k-th interval, the minimal length of this interval can be calculated using the following theorem proved by Węglarz for the case without discrete resources [19]:

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

$$u_i^* = f_i^{-1}(\tilde{x}_i / M^*), \quad i = 1, 2, \ldots, n \tag{3}$$

where M^* *is the unique positive root of the equation*:

$$\sum_{i=1}^{n} f_i^{-1}(\tilde{x}_i / M) = 1 \tag{4}$$

Now, the $\tilde{x}_{ik}$'s can be treated as variables. Let $M_k^*(\tilde{\mathbf{x}}_k)$ be the minimum length of the part of the schedule generated by $Z_k \in \boldsymbol{S}$, as a function of $\tilde{\mathbf{x}}_k = \{\tilde{x}_{ik}\}_{A_i \in Z_k}$. Let K_i be the set of all indices of Z_k's such that $A_i \in Z_k$. The following mathematical programming problem is obtained to find an optimal demand division of activities for a given feasible sequence $\boldsymbol{S}$ [10]:

Problem P

$$\text{minimize} \quad \sum_{k=1}^{s} M_k^*(\tilde{\mathbf{x}}_k) \tag{5}$$

subject to

$$\sum_{k \in K_i} \tilde{x}_{ik} = \tilde{x}_i, \quad i = 1, 2, \ldots, n \tag{6}$$

$$\tilde{x}_{ik} \geq 0, \quad i = 1, 2, \ldots, n; \; k \in K_i \tag{7}$$

where for every given $\tilde{\mathbf{x}}_k$ the value of $M_k^*(\tilde{\mathbf{x}}_k)$ is calculated as the unique positive root of the equation:

$$\sum_{A_i \in Z_k} f_i^{-1}(\tilde{x}_{ik}/M_k) = 1 \tag{8}$$

which can be solved analytically for some important cases, e.g. linear functions or power functions. It should be stressed that in order to calculate reverse functions f_i^{-1}, it has to be assumed that each function f_i, $i = 1, 2, \ldots, n$, is a bijection. Constraints (6) correspond to the condition of fulfilling the processing demands of all activities, constraints (7) ensure that the $\tilde{x}_{ik}$'s are non-negative. It was also proved in [19] that the objective function (5) is always a convex function. In consequence, the problem is to minimize a convex function subject to linear constraints. After finding an optimal division $\tilde{x}_{ik}$, $i = 1, 2, \ldots, n, k \in K_i$ of $\tilde{x}_i$'s, the corresponding optimal continuous resource allocation for combination Z_k is given as:

$$u_{ik}^* = f_i^{-1}(\tilde{x}_{ik}/M_k^*), \quad A_i \in Z_k \tag{9}$$

Thus, the solution of Problem P allows to find an optimal continuous resource allocation for a given feasible sequence. Consequently, the DCRCPSP decomposes into two interrelated subproblems: (i) to construct a precedence- and resource-feasible sequence of activities with respect to discrete resources only, i.e. a feasible sequence as defined earlier, and (ii) to allocate the continuous resource optimally among activities in the feasible sequence. As a result, the problem of searching for an optimal solution can be seen as a problem of searching for an optimal feasible sequence over the whole set of feasible sequences. This brings down the problem to a combinatorial optimization like problem, since its continuous part can be solved optimally. However, as a whole, it is not a combinatorial optimization problem because its parameters, in particular–processing demands of activities and the factors of processing rate functions, may be irrational numbers, and they cannot be coded by any reasonable encoding scheme as a string of finite length. Moreover, solutions of the problem may contain irrational numbers as well, since the continuous resource allocations can be arbitrary numbers from the interval [0, 1]. If such an allocation, calculated from formula (9), equals, e.g., $\sqrt{2}/2$, then it cannot be coded as a finite string, either. In general, in the continuous part of the problem a non-linear objective function (5) of continuous variables appears, whose values may be irrational numbers. In consequence, the complexity of the DCRCPSP, as a whole, cannot be analyzed in terms of the P and NP classes. What can only be surely stated is that it is at least as hard as the classical RCPSP, since the existence of an additional continuous resource cannot make the problem simpler.

It should be stressed that the decomposition of the DCRCPSP into two subproblems (as discussed above) is of huge importance. Firstly, notice that an optimal solution can be found by solving Problem P for all feasible sequences, and choosing the one with the minimum makespan. This full enumeration approach can be applied for small problem sizes, and guarantees finding an optimal schedule. In general, the number of all feasible sequences grows exponentially with the number of activities, and therefore, searching for an optimal feasible sequence may be performed by various search algorithms, e.g. local search metaheuristics (see [10]). Secondly, the decomposition into the discrete and the continuous part allows to incorporate some knowledge on the properties of solutions to both the subproblems, and, in that way, identify cases which are easier to solve. An alternative approach could be a formulation of the DCRCPSP as a mixed integer non-linear programming (MINLP) problem, as some problem variables are discrete and some are continuous. However, as it is known, MINLP problems are typically very difficult to solve (see, e.g., [4]), and therefore this approach has not been taken into consideration so far.

4 Heuristic continuous resource allocation

In this section a heuristic approach to the DCRCPSP defined in Sect. 2, with concave processing rate functions of activities, is presented. The importance of the convex mathematical programming Problem P defined in Sect 3.2 follows from the fact that it finds an optimal continuous resource allocation for a given feasible sequence and arbitrary concave (bijection) functions f_i. This can be very useful for small problem sizes when each feasible sequence may be valued by the minimal schedule length it generates. In such a case, an optimal schedule can be found by solving Problem P for each feasible sequence and choosing one with the minimal value of the makespan. However, as mentioned before, the number of all feasible sequences is exponentially dependent on the number of activities. Thus, for larger problems heuristics should be used, e.g. local search algorithms searching a part of the set of all feasible sequences, where the objective function is defined as the minimal makespan for a given feasible sequence. However, mathematical programming problems, although theoretically extremely valuable are, in general, computationally intractable. This is also the case as far as Problem P is concerned. Therefore, another approach was proposed in [13], where heuristic algorithms were examined for allocating the continuous resource.

The results presented in the above paper were quite promising, and therefore the research was continued in [8], where several heuristic procedures for allocating the continuous resource were examined. A special case of the DCRCPSP was considered in order to simplify the calculations. There was one discrete resource given, resource requests of all activities were identical and equal to 1, and there were no precedence constraints imposed. In fact, this special case reduces to a discrete-continuous machine scheduling problem, where the discrete resource is a set of parallel identical machines, and the number of machines is equal to the number of available units of the single discrete resource (R_1). The heuristics were compared on a basis of an extensive computational experiment, and the HUDD heuristic turned out to be the most efficient for the considered problem. It was then further analyzed in [18] in the

context of the tabu search metaheuristic for discrete-continuous machine scheduling problems. The results confirmed its efficiency for the considered class of problems. Below the issue of applying the HUDD heuristic to the general case of the DCRSPSP will be discussed. The general heuristic will be called HUDD-PS.

The idea of the HUDD-PS heuristic is to uniformly distribute the processing demand of each activity among these combinations of a feasible sequence which contain this activity. To this end, the number of all combinations containing the particular activity must be counted, and then the processing demand of the activity is divided by that number in order to distribute it uniformly over all those combinations. Having known the parts of the processing demands of activities in each combination Z_k, $k = 1, 2, \ldots, s$, the length of the k-th interval can be then calculated as the unique positive root of the equation:

$$\sum_{A_i \in Z_k} f_i^{-1}(y_i / M_k) = 1 \tag{10}$$

where y_i is the part of the processing demand of activity A_i, $i = 1, 2, \ldots, n$, executed in a single interval. Thus, heuristic HUDD-PS can be formulated as follows:

HEURISTIC HUDD-PS

Input data: a set of n activities with processing demands $\tilde{x}_1, \tilde{x}_2 \ldots, \tilde{x}_n$ and processing rate functions $f_1, f_2, \ldots, f_n$, as well as a feasible sequence $\boldsymbol{S}$ for the particular instance of the DCRCPSP.

Step 1. Count the number s of all combinations in the feasible sequence $\boldsymbol{S}$.

Step 2. For each activity A_i, $i = 1, 2, \ldots, n$, count the number h_i of combinations in $\boldsymbol{S}$ which contain activity A_i.

Step 3. Calculate the part y_i of the processing demand of each activity A_i executed in a single combination (assuming the uniform distribution of the processing demand over all combinations) as:

$$y_i = \frac{\tilde{x}_i}{h_i}$$

Step 4. For $k = 1, 2, \ldots, s$ calculate the length M_k of the k-th interval as the unique positive root of the equation:

$$\sum_{A_i \in Z_k} f_i^{-1}(y_i / M_k) = 1$$

Step 5. Calculate the value of the makespan as:

$$M = \sum_{k=1}^{s} M_k$$

Example 2 Assume a problem with $n = 4$ and power processing rate functions of activities of the form $f_i = u_i^{1/a_i}$ with $a_i \in \{1, 2\}$, $i = 1, 2, \ldots, n$. The processing demands of the activities, and their values of a_i are defined by the vectors:

$\tilde{\mathbf{x}} = [25, 36, 64, 47]$ and $\mathbf{a} = [1, 2, 1, 2]$. The considered feasible sequence is $\boldsymbol{S} = \{1, 2, 3\}, \{2, 3\}, \{2, 4\}$.

Steps 1, 2, and 3

$$s = 3$$

$$h_1 = 1,\ y_1 = \frac{25}{1} = 25;\ h_2 = 3,\ y_2 = \frac{36}{3} = 12;\ h_3 = 2,\ y_3 = \frac{64}{2} = 32;$$

$$h_4 = 1,\ y_4 = \frac{47}{1} = 47$$

Step 4

$$k = 1 : \frac{25}{M_1} + \frac{12}{M_1^2} + \frac{32}{M_1} = 1;\ M_1 = 57.21$$

$$k = 2 : \left(\frac{12}{M_2}\right)^2 + \frac{32}{M_2} = 1;\ M_2 = 31.62$$

$$k = 3 : \left(\frac{12}{M_3}\right)^2 + \left(\frac{47}{M_3}\right)^2 = 1;\ M_3 = 48.51$$

Step 5

$$M = M_1 + M_2 + M_3 = 137.34$$

It should be stressed that the HUDD-PS heuristic only takes care of the continuous part of the DCRCPSP, the discrete part of the problem is handled by the feasible sequence. In other words, the HUDD-PS heuristic operates on a feasible sequence which already manages the precedence and discrete resource constraints. Thus, it is not important for the heuristic how the precedence constraints graph looks like, or how many discrete resources are available and in what amounts. This procedure just allocates the continuous resource for a given feasible sequence. The only additional data it needs are the processing demands and processing rate functions of activities.

5 Continuous resource discretization

As already discussed in Sect. 4, the convex mathematical programming Problem P defined in Sect. 3.2 finds an optimal continuous resource allocation for a given feasible sequence and arbitrary concave (bijection) functions f_i. However, for larger problem sizes, when the number of variables in the problem grows rapidly, the computational time needed for solving it using specialized non-linear solvers can be quite long. Therefore a heuristic approach to allocating the continuous resource has been presented in Sect. 4.

Another approach was proposed in [11] where continuous resource allotments were discretized. As a result, the classical discrete MRCPSP is obtained in a version

without non-renewable resources. The number of modes for the activities depends on the discretization level, i.e. the number of discretized allotments. It should be stressed that in this approach the processing rate functions of activities need not be concave, but may be arbitrary continuous, increasing functions. Below the idea of the continuous resource discretization is presented in more detail.

Denote by $u_i^{b(i)}, u_i^{b(i)} \in (0, 1]$, the discretized continuous resource allotment for activity A_i, where $b(i) = 1, 2, \ldots, B_i$, and B_i is the maximal number of allotments for activity A_i. Then the processing time (duration) of activity A_i for this particular allotment can be obtained from formula (1) as:

$$d_i^{b(i)} = \frac{\tilde{x}_i}{f_i(u_i^{b(i)})} \tag{11}$$

Thus, in other words, this defines B_i execution modes for activity A_i where each mode is a relation between the duration of the activity and its request for the discretized continuous resource. The discretized resource becomes an additional discrete resource, i.e. resource $|K_R| + 1$st. The resource is renewable and its total available amount is equal to 1. Note that the resource requests of activity A_i with respect to the other discrete resources, i.e., values of r_{il}, $l = 1, 2, \ldots, |K_R|$, do not change and are identical in each mode. Obviously, there can be many different ways of defining the values of $u_i^{b(i)}$, $b(i) = 1, 2, \ldots, B_i$. One of them may be a simple method of calculating the allotments as:

$$u_i^{b(i)} = \frac{b(i)}{B_i} \tag{12}$$

computationally tested for simulated annealing in [11]. It is easy to see that the bigger value of B_i, the more the discretized resource approaches its original continuous version. On the other hand, the bigger value of B_i, the larger number of modes for activity A_i which makes the problem more difficult.

Below a brief example of the discretization idea is presented, for a small DCRCPSP instance with four activities and two discrete resources.

Example 3 Consider a DCRCPSP with $n = 4$ and $|K_R| = 2$. The processing demands, processing rate functions, and discrete resource requests of the activities are the following:

i	$\tilde{x}_i$	r_{i1}	r_{i2}	$f_i(u_i)$
1	36	2	3	u_i
2	22	4	0	u_i
3	45	1	2	$\sqrt{u_i}$
4	18	0	3	$\sqrt[3]{u_i}$

The discrete resources are available in $R_1 = R_2 = 5$ units each. For simplicity, it is assumed that processing rate functions of activities A_1 and A_2 are identical and

linear, i.e. $f_1(u_i) = f_2(u_i) = u_i$. The processing rate functions for activities A_3 and A_4 are power functions of the form $u_i^{1/2}$ and $u_i^{1/3}$, respectively. Moreover, precedence constraints between activities are neglected since they have no impact on the discretization procedure.

Assume now that the number of modes for activity A_1 and A_2 is $B_1 = B_2 = 2$, whereas for activities A_3 and A_4 : $B_3 = B_4 = 3$. For the latter two activities formula (12) is used for calculating the discretized resource allotments, i.e.:

$$u_3^1 = u_4^1 = \frac{1}{3}, \qquad u_3^2 = u_4^2 = \frac{2}{3}, \qquad u_3^3 = u_4^3 = 1$$

whereas for the former two activities arbitrary values are assumed, e.g.:

$$u_1^1 = \frac{1}{2}, \qquad u_2^1 = \frac{2}{5}, \qquad u_1^2 = u_2^2 = 1$$

It is important to assure for each activity a mode in which its resource request for the discretized resource is equal to 1. This enables the activity to be executed with full speed in case it should be performed alone in an interval.

Continuing Example 3, the durations of activities in successive modes are now obtained from formula (11), e.g. the duration of activity A_3 in mode 2 is calculated as:

$$d_3^2 = \frac{\tilde{x}_3}{f_3(u_3^2)} = \frac{45}{\sqrt{\frac{2}{3}}} = 55.11 \approx 55$$

The durations of the activities have to be rounded as all the parameters of the MRCPSP are integer valued. Moreover, for the same reason the activity requests for the discretized resource, which are proper fractions from interval (0, 1], have to be multiplied by their lowest common denominator (LCD). At the same time, the value of the LCD will be the number of available units of the new discretized resource. After all necessary calculations, the following instance of the MRCPSP in a version without non-renewable resources is obtained:

i	number of modes $\lvert M_i \rvert$	set of modes M_i	mode m_i	d_{im}	r_{im1}	r_{im2}	r_{im3}
1	2	{1, 2}	1	72	2	3	15
			2	36	2	3	30
2	2	{1, 2}	1	55	4	0	12
			2	22	4	0	30
3	3	{1, 2, 3}	1	75	1	2	10
			2	55	1	2	20
			3	45	1	2	30
4	3	{1, 2, 3}	1	26	2	3	10
			2	21	2	3	20
			3	18	2	3	30

Obviously, now there are $|K_R| = 3$ discrete resources, available in the following numbers of units: $R_1 = R_2 = 5$, and $R_3 = 30$. However, there is no continuous resource anymore.

The above multi-mode resource-constrained project scheduling problem can be solved using one of the numerous existing methods, e.g. efficient metaheuristics. This will generate an approximate schedule for the original DCRCPSP, however the calculations will be much less time consuming with no need for solving the non-linear mathematical programming problem.

6 Identical processing rate functions

In this section a special case of the DCRCPSP will be considered, where the processing rate functions of all activities are identical and concave, i.e. $f_i(u_i) = f(u_i)$, $i = 1, 2, \ldots, n$. Such situations often occur in practice, e.g. in multiprocessor computer systems where several similar parallel applications (tasks) apply for processors for their execution. As mentioned in Sect. 1, when the number of processors in a MPP or SPP system is huge, the set of processors can be treated as a continuous resource, allowing to apply the discrete-continuous methodology. If increasing the number of processors allotted to each task results in the same acceleration of its execution, which is very likely if they are applications of the same kind or even the same applications performed on different data sets, it means that the processing rates of all tasks are defined by the same function of the amount of the continuous resource (processors) allotted. Similar situations from other areas can be given as well. It will be shown in this section that if the processing rate functions of all activities are identical, the process of finding an optimal solution of a discrete-continuous project scheduling problem can be reduced to searching for an optimal solution of the corresponding discrete problem.

Notice that if the amount u_i of the continuous resource is allotted to activity A_i over the whole period of its execution, the actual duration d_i of activity A_i can be obtained from (1) as:

$$d_i = \frac{\tilde{x}_i}{f_i(u_i)} \tag{13}$$

For the case where $f_i = f, i = 1, 2, \ldots, n$, it means that:

$$d_i = \frac{\tilde{x}_i}{f(u_i)} \tag{14}$$

or, consequently:

$$d_i = \tilde{x}_i g(u_i) \tag{15}$$

where $g(u_i) = \frac{1}{f(u_i)}$.

Thus, since $\tilde{x}_i$ is known for activity $A_i, i = 1, 2, \ldots, n$, formula (15) shows that the duration of each activity is expressed by the same function g of the continuous resource amount, multiplied by the known factor $\tilde{x}_i$. As a result, to construct a schedule, continuous resource division may be temporarily neglected and processing demands

of activities can be treated as their actual durations. In this way, the classical discrete RCPSP arises where no feasible sequence is needed to build a schedule but just an activity list (AL).

For a given activity list AL, a makespan optimal schedule can be constructed using the serial Schedule Generation Scheme (SGS) [14] decoding rule. It was proved in [15] that (i) SGS always generates active schedules, i.e. schedules where none of the tasks can be started earlier without delaying some other task, and (ii) for makespan minimization problems an optimal solution is always a member of the set of active schedules, i.e. there always exists an activity list for which the serial SGS gives an optimal solution. The serial SGS acts quite simply. It just takes the first yet unscheduled task A_i from the list, and schedules it at the earliest possible starting time S_i that does not violate precedence or resource constraints. Next, based on this schedule, the completion times of consecutive activities define the ends of successive intervals, and for each interval a combination of activities processed in parallel can be constructed. In consequence, a feasible sequence can be obtained as a sequence of such combinations, as discussed in Sect. 3. Finally, an optimal continuous resource allocation can be found by solving Problem P for the constructed feasible sequence.

Formalizing the above idea, the following procedure is proposed which will be called the DCSGS (Discrete-Continuous based on the serial Schedule Generation Scheme) decoding rule. In general, the DCSGS procedure builds a feasible schedule of the DCRCPSP and calculates its length for a given precedence-feasible activity list.

PROCEDURE DCSGS

Step 1. For each activity $A_i, i = 1, 2, \ldots, n$, set d_i equal to $\tilde{x}_i$.

Step 2. Construct a precedence- and discrete resource-feasible, makespan-optimal schedule for the given instance of the DCRCPSP and the given activity list AL, using the serial SGS decoding rule and assuming d_i as the duration of activity A_i.

Step 3. Construct a feasible sequence $\boldsymbol{S}$ of combinations of activities processed in parallel in successive intervals, where the intervals are defined by completion times of consecutive activities.

Step 4. Find an optimal makespan for feasible sequence $\boldsymbol{S}$ by solving Problem P for this sequence.

Example 4 Consider the DCRCPSP instance presented in Example 1 where all processing rate functions of activities are identical and concave, and the processing demands of activities are: $\tilde{\mathbf{x}} = [2, 3, 5, 1, 4, 4, 3, 2, 1]$. Assuming $d_i = \tilde{x}_i$, $i = 1, 2, \ldots, 9$, and the simplest precedence feasible activity list AL $= (1, 2, 3, 4, 5, 6, 7, 8, 9)$ in which activities occur in an ascending order, the following schedule is obtained after the execution of the serial SGS (Fig. 3).

At this point, steps 1 and 2 of the DCSGS procedure are completed. Step 3 can be performed now, on a basis of the generated schedule. Successive combinations Z_1 up to Z_8 arise, each one containing activities processed in parallel in consecutive intervals. It is easy to see in Fig. 3 that $Z_1 = \{1\}$, $Z_2 = \{2, 4, 3\}$, $Z_3 = \{2, 3\}$, $Z_4 = \{5, 3\}$, $Z_5 = \{5, 6\}$, $Z_6 = \{8, 6\}$, $Z_7 = \{7\}$, and $Z_8 = \{9\}$, which results in a feasible

sequence $\boldsymbol{S}$ of the form:

$$\boldsymbol{S} = \{1\}, \{2, 4, 3\}, \{2, 3\}, \{5, 3\}, \{5, 6\}, \{8, 6\}, \{7\}, \{9\}$$

For the above feasible sequence $\boldsymbol{S}$, Problem P can now be formulated and solved (step 4), giving an optimal schedule length for this sequence.

Now, it must be stressed that, in principle, the DCSGS can be applied to any instance of the DCRCPSP. However, if it is not the case of identical processing rate functions, it will be a heuristic approach with a questionable chance to find an optimal solution because of the variety of processing rate functions. Although, it could be computationally tested in such a case as well (see Sect. 7.3). On the other hand, if all processing rate functions of activities are identical and concave, the DCSGS procedure can be safely applied to find a minimal-length schedule for a given activity list. This result reduces the search space dramatically in such cases, since the number of all feasible sequences is much bigger than the number of all precedence-feasible lists of activities.

Furthermore, in fact, the above approach may be applied to a wider class of functions, which are of identical form with the accuracy of a multiplier. Namely, it was proved in [9] that a discrete-continuous scheduling problem with concave processing rate functions of the form: $f_i(u_i) = c_i f(u_i)$, $c_i > 0$, $i = 1, 2, \ldots, n$, is equivalent to a problem with identical functions $f_i(u_i) = f(u_i)$, $i = 1, 2, \ldots, n$, where processing demands of activities take the values of $\tilde{y}_i = \frac{\tilde{x}_i}{c_i}$, $i = 1, 2, \ldots, n$. As a result, the DCRCPSP with processing rate functions of activities of the form $f_i(u_i) = c_i f(u_i)$, $c_i > 0$, $i = 1, 2, \ldots, n$, can be easily transformed into a problem with identical processing rate functions, and the DCSGS procedure can be directly applied.

Example 5 Consider a DCRCPSP instance with $n = 3$ where:

$$f_1 = 3\sqrt{u_1}, \qquad \tilde{x}_1 = 60; \qquad f_2 = \sqrt{u_2}, \qquad \tilde{x}_2 = 50;$$
$$f_3 = \frac{\sqrt{u_3}}{2}, \qquad \tilde{x}_3 = 40$$

This instance can be transformed to a DCRCPSP instance with identical processing rate functions $f_i = \sqrt{u_i}$, $i = 1, 2, 3$, where the new processing demands of activities are:

$$\tilde{y}_1 = \frac{\tilde{x}_1}{c_1} = \frac{60}{3} = 20; \qquad \tilde{y}_2 = \frac{\tilde{x}_2}{c_2} = \frac{50}{1} = 50; \qquad \tilde{y}_3 = \frac{\tilde{x}_3}{c_3} = \frac{40}{\frac{1}{2}} = 80$$

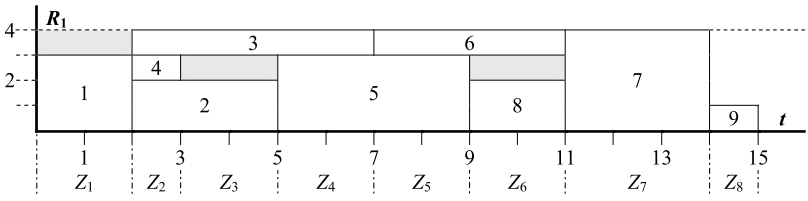

Fig. 3 Schedule generated by the serial SGS

7 Computational experiments

Some computational experiments have been performed in order to evaluate the efficiency of the proposed heuristic approaches. The experiments are described below.

7.1 Heuristic HUDD-PS

A computational experiment was carried out to examine the performance of the HUDD-PS heuristic, presented in Sect. 4, for allocating the continuous resource in the DCRCPSP. The heuristic was implemented in C++ and ran on SGI ORIGIN 3800 machine with 160 RISC R12000 processors and 128 Gflops overall computing power, installed in the Poznań Supercomputing and Networking Center. The results (makespans) generated by the heuristic for random feasible sequences were compared with optimal makespans found by solving Problem P (5)–(8). In order to solve the non-linear mathematical programming problem, a specially adopted solver CFSQP—C code for Feasible Sequential Quadratic Programming—was applied [16]. CFSQP is a set of C functions for the minimization of the maximum of a set of smooth objective functions, possibly a single one or even none at all, subject to nonlinear equality and inequality constraints, linear equality and inequality constraints, and simple bounds on the variables. It has proved to work very well in solving the continuous part of discrete-continuous scheduling problems in many experiments before. The solver stopped when the absolute difference in consecutive values of the objective function was less than or equal to 10^{-3}.

The experiment was performed for $n = 10$, 15, and 20 activities. One discrete resource was considered ($|K_R| = 1$) but various numbers of its available units were examined: $R_1 \in \{2, 5, 10\}$. The values of r_{i1} (discrete resource requests of activities) were generated randomly as integers from the interval $[0,\ R_1/2]$. Processing rate functions of activities were of the form $f_i = u_i^{1/a_i}$ with $a_i \in \{1, 2\}$, $i = 1, 2, \ldots, n$. The values of a_i were generated randomly with equal probabilities, whereas processing demands of activities were generated from the uniform distribution in the interval [1, 100]. For each problem size 100 instances were generated, and for each instance 500 feasible sequences were randomly generated. The random generation was based on the two-list representation of a feasible sequence described in [17]. It simply consists in generating randomly two precedence-feasible n-element list of activities. The decoding procedure producing a sequence feasible with respect to discrete resource constraints is very simple [17].

The results of the experiments are presented in Table 2. For each problem size it shows:

- *ARD*—the average relative deviation from optimum, where the relative deviation *RD* for a given instance is calculated according to the formula:

$$RD = \frac{M - M^*}{M^*} \cdot 100\%$$

  in which M is the value obtained by the HUDD-PS heuristic and M^* is the optimal value of the makespan found by the CFSQP solver;

Table 2 Results of the experiment

		HUDD-PS			CFSQP
n	R_1	*ARD* [%]	*MRD* [%]	*CPU* [sec]	*CPU* [sec]
10	2	4.78	12.60	0.8465	716.66
	5	4.22	12.37	0.7971	823.99
	10	5.79	11.92	0.8123	767.53
15	2	4.33	10.99	0.8895	2961.04
	5	5.15	11.46	0.8512	3013.56
	10	4.87	12.74	0.7996	3221.45
20	2	4.09	12.08	0.9014	8567.35
	5	4.71	11.15	0.8442	7978.21
	10	5.20	12.72	0.8870	8340.36

- *MRD*—the maximal relative deviation over all feasible sequences tested;
- *CPU*—the average computational time for 1 instance (i.e. for 500 feasible sequences).

The results show that the proposed HUDD-PS heuristic can be considered as quite effective for the analyzed problem. The average relative deviation from optimum does not exceed 6%, whereas the maximal relative deviation oscillates around 12%. Moreover, these values do not increase with the growth of the problem size, so that the heuristic seems to be independent of the number of activities, at least for such small problems. As it was expected, the number of discrete resource units does not have any influence on the obtained results, since the HUDD-PS heuristic, as well as the CFSQP solver, start when a feasible sequence is already constructed. The value of R_1 is only taken into account while constructing the sequence, i.e. during the execution of the decoding rule. Thus, the results obtained by the heuristic are quite reasonable as far as the quality is concerned. Taking into account that it allows to shorten the computations from about 1000 times for 10 activities up to about 10000 times for 20 activities, it may become a good alternative for exact solving the continuous part of the problem. Despite the rather simple idea underlying HUDD-PS, it can be seen that allocating the continuous resource as uniformly as possible (taking into account the numbers of occurrences of particular activities in a feasible sequence) leads to quite good schedules.

7.2 Continuous resource discretization

As described in Sect. 5, discretization of the continuous resource is a heuristic approach to discrete-continuous project scheduling problem, where continuous resource allotments are discretized. This leads to a classical discrete multi-mode problem. It should be stressed that within this approach many various heuristic algorithms can be formulated, depending on the method of calculating the discretized allotments. In

[11] a simple method of calculating the allotments according to formula (12) was used, and tested within a simulated annealing (SA) approach. In this section, the results of the experiment presented in [11] are briefly recalled.

The experiment was performed for $n = 10$ activities, and one discrete resource available in various numbers of units $R_1 \in \{2, 5, 10, 15\}$. Processing rate functions of activities were also of the form $f_i = u_i^{1/a_i}$ with $a_i \in \{1, 2\}$, $i = 1, 2, \ldots, n$. For simplicity, the same number of modes following from the continuous resource division was assumed for each activity of the project, i.e. $B_i = B$, $i = 1, 2, \ldots, n$. Different values of parameter B were examined, from $B = 2$ up to $B = 100$. For each combination of parameters R_1 and B, 30 instances were randomly generated and solved by two SA implementations:

- SAM+ implementation for the MRCPSP obtained as a result of the continuous resource discretization in the DCRCPSP. SAM+ found a schedule for the MRCPSP, then steps 3 and 4 of Procedure DCSGS presented in Sect. 6 were applied to find a solution of the corresponding DCRCPSP. As the base for the SAM+ implementation, a very efficient simulated annealing algorithm for the MRCPSP presented in [12] was used.
- SAA implementation for the DCRCPSP in which the continuous resource was allocated optimally for each feasible solution tested. To this end, the CFSQP solver was used.

The experiment showed that the distance of the results produced by SAM+ from the ones obtained by SAA was 4.2% on average, and did not exceed 5.5% for all problem sizes. For a few instances the SAM+ procedure was also able to find solutions of the same quality as SAA (up to 5 out of 30). However, the average computational time for one instance was about 190 sec. for SAM, whereas for SAA it was about 13000 sec. on average. The increase of the computational time was caused by the application of the CFSQP solver in each step of the SAA implementation. The number of visited solutions, defining the stop criterion for both the algorithms, was set at 1000.

Summarizing the results, the SAM+ approach allowed to reduce the computational time up to 70 times while losing not more than 5.5% of solution quality. Moreover, the SAM+ results were quite stable and did not seem to depend on the number of available units (R_1) of the discrete resource or on the discretization level (B). On the other hand, it could be seen that a dramatic increase of the number of modes B in the discretized model did not improve the quality of solutions, even if it made the discrete resource "closer" to the continuous one. Quite good results were obtained for $B = 5$ for all data sets considered in the experiment. Thus, the discretization method can be recommended for finding good solutions of the DCRCPSP in a reasonable time. For more details concerning the experiment described in this section, as well as the simulated annealing implementations, see [11].

7.3 Procedure DCSGS

As mentioned in Sect. 6, the DCSGS procedure is an exact approach in the case of identical processing rate functions only. However, it can be tested as a heuristic

Table 3 Results of the experiment

n	R_1	SAD *ARD* [%]	SAD *MRD* [%]	SAD *CPU* [sec]	SAA *CPU* [sec]
10	2	23.36	37.33	1.59	1687.23
	5	25.87	39.18	1.67	1712.45
	10	21.99	36.75	1.72	1699.08
	15	22.21	36.89	1.69	1691.43
15	2	34.98	47.90	6.11	6029.67
	5	36.02	45.65	6.18	6103.70
	10	35.11	49.01	6.35	6097.47
	15	36.43	48.52	6.33	6219.52

approach for a case of different functions which is the subject of this section. An experiment concerning such an approach is described below.

The same SA implementation, as before taken from [12], was adapted to the considered DCRCPSP problem in such a way that the mode assignment list was neglected. Thus, the SA algorithm approached to the DCRCPSP as to the discrete RCPSP by assuming that the processing rate functions of all activities are identical. Consequently, activity processing demands were treated as their durations, and activity list (AL) was considered as a solution representation. This SA implementation is denoted by SAD. Thus, SAD searches over the set of all feasible activity lists, for each AL visited it performs step 2 of Procedure DCSGS in order to calculate the makespan, and for the best AL found it performs steps 3 and 4 of DCSGS in order to find a solution of the corresponding DCRCPSP.

The SAD implementation was compared with the SAA implementation, already discussed in Sect. 7.2. As before, the number of visited solutions, defining the stop criterion for both the algorithms, was set at 1000. For the experiment the same set of instances for $n = 10$ activities was used as described in Sect. 7.2. Moreover, also 30 instances for $n = 15$ activities were randomly generated under the same parameter settings concerning discrete resources and processing rate functions. Table 3 presents the average and maximal relative deviations of the results obtained by SAD from the results produced by SAA (calculated as described in Sect. 7.1), as well as computational times needed by both the algorithms for one problem instance (1000 solutions).

The results clearly show that the DCSGS procedure cannot be considered as a reasonable heuristic alternative for the case of various processing rate functions of activities. Although the computational time of SAD is about 1000 times smaller than of SAA (it is understandable since SAA uses the non-linear solver for each of the 1000 solutions whereas SAD just for the final one), the deviations of the results obtained by SAD from the SAA results are not acceptable. The average deviation over 20% and the maximal up to 40% for 10 activities, as well as over 30% on average and almost 50% maximum for 15 activities show that even for a simple case of functions $f_i = u_i^{1/a_i}$, $a_i \in \{1, 2\}$, the problem may not be simplified by neglecting the processing rate functions of activities. It can be assumed that for problems of bigger size, as

well as for functions of a greater variety, the results produced by SAD will deteriorate even more.

8 Conclusions

In this paper a discrete-continuous project scheduling problem to minimize the makespan was considered. The problem was decomposed into the discrete and the continuous part. Different approaches to solving the continuous part of the problem were presented—an exact approach requiring solving a convex mathematical programming problem, a heuristic approach to the continuous resource allocation problem (heuristic HUDD-PS), and the approach based on the continuous resource discretization. The two latter approaches allow to avoid using specialized solvers for solving the convex problem. Similar situation was shown for identical processing rate functions of activities. It was proved that in this case the discrete-continuous problem reduces to the corresponding discrete one. Computational results presented in the paper show that the HUDD-PS heuristic, as well as the discretization approach, can be considered as reasonable heuristic alternatives for an exact solution of the DCRCPSP, whereas the DCSGS procedure should only be used as an exact approach to problems with identical processing rate functions of activities.

In the future research it is planned to develop heuristic approaches to the continuous part of the problem under other optimization criteria, like mean flow time, maximum lateness, or financial criteria like the net present value. Also more extensive computational experiments are intended, especially for the discretization approach and for the case with identical processing rate functions. Finally, further work on continuous resource allocation heuristics is planned, in order to improve the efficiency of HUDD-PS, or to find another even more effective procedure.

Acknowledgement This research has been supported by the Polish Ministry of Education and Science, grant no N N519 403437.

References

1. Błażewicz, J., Kovalyov, M.Y., Machowiak, M., Trystram, D., Węglarz, J.: Malleable task scheduling to minimize the makespan. Ann. Oper. Res. **129**(1–4), 65–80 (2004)
2. Burkov, V.N.: Raspriedielenije riesursow kak zadacza optimalnogo bystrodiejstwia. Avtom. Telem. **27**(7) (1966)
3. Demeulemeester, E.L., Herroelen, W.S.: Project Scheduling—A Research Handbook. Kluwer, Boston (2002)
4. Floudas, C.A.: Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications. Oxford University Press, Oxford (1995)
5. Herroelen, W.S., Demeulemeester, E.L., De Reyck, B.: A classification scheme for project scheduling. In: Węglarz, J. (ed.) Project Scheduling—Recent Models, Algorithms and Applications, pp. 1–26. Kluwer, Dordrecht (1999)

6. Janiak, A.: Scheduling and resource allocation problems in some flow type manufacturing processes. In: Fandel, G., Zapfel, G. (eds.) Modern Production Concepts, pp. 404–415. Springer, Berlin (1991)
7. Janiak, A.: Single machine scheduling problem with a common deadline and resource dependent release dates. Eur. J. Oper. Res. **53**(3), 317–325 (1991)
8. Józefowska, J., Waligóra, G.: Heuristic procedures for allocating the continuous resource in discrete-continuous scheduling problems. Found. Comput. Decis. Sci. **29**(4), 315–328 (2004)
9. Józefowska, J., Węglarz, J.: On a methodology for discrete-continuous scheduling. Eur. J. Oper. Res. **107**(2), 338–353 (1998)
10. Józefowska, J., Mika, M., Różycki, R., Waligóra, G., Węglarz, J.: Project scheduling under discrete and continuous resources. In: Węglarz, J. (ed.) Project Scheduling—Recent Models, Algorithms and Applications, pp. 289–308. Kluwer, Dordrecht (1999)
11. Józefowska, J., Mika, M., Różycki, R., Waligóra, G., Węglarz, J.: Solving the discrete-continuous project scheduling problem via its discretization. Math. Methods Oper. Res. **52**(3), 489–499 (2000)
12. Józefowska, J., Mika, M., Różycki, R., Waligóra, G., Węglarz, J.: Simulated annealing for multi-mode resource-constrained project scheduling problem. Ann. Oper. Res. **102**(1–4), 137–155 (2001)
13. Józefowska, J., Mika, M., Różycki, R., Waligóra, G., Węglarz, J.: A heuristic approach to allocating the continuous resource in discrete-continuous scheduling problems to minimize the makespan. J. Sched. **5**(6), 487–499 (2002)
14. Kelley, J.E.: The critical path method: resource planning and scheduling. In: Muth, J.F., Thompson, G.L. (eds.) Industrial Scheduling, pp. 347–365. Prentice-Hall, Englewood Cliffs (1963)
15. Kolisch, R.: Project Scheduling under Resource Constraints—Efficient Heuristics for Several Problem Classes. Physica, Heidelberg (1995)
16. Lawrence, C., Zhou, J.L., Tits, A.L.: Users guide for CFSQP Version 2.5. Available by email: andre@eng.umd.edu (1997)
17. Waligóra, G.: Discrete-continuous project scheduling with discounted cash flows—a tabu search approach. Comput. Oper. Res. **35**(7), 2141–2153 (2008)
18. Waligóra, G.: Tabu search for discrete-continuous scheduling problems with heuristic continuous resource allocation. Eur. J. Oper. Res. **193**(3), 849–856 (2009)
19. Węglarz, J.: Time-optimal control of resource allocation in a complex of operations framework. IEEE Trans. Syst. Man Cybern. **6**(11), 783–788 (1976)
20. Węglarz, J.: Multiprocessor scheduling with memory allocation—a deterministic approach. IEEE Trans. Comput. **C-29**, 703–709 (1980)
21. Węglarz, J.: Project scheduling with continuously-divisible, doubly constrained resources. Manag. Sci. **27**(9), 1040–1052 (1981)
22. Węglarz, J.: Modelling and control of dynamic resource allocation project scheduling systems. In: Tzafestas, S.G. (ed.) Optimization and Control of Dynamic Operational Research Models, pp. 105–140. North-Holland, Amsterdam (1982)