RAIRO-Oper. Res. 48 (2014) 1–24
DOI: 10.1051/ro/2013045

RAIRO Operations Research
www.rairo-ro.org

SIMULATED ANNEALING AND TABU SEARCH FOR DISCRETE-CONTINUOUS PROJECT SCHEDULING WITH DISCOUNTED CASH FLOWS

Grzegorz Waligóra[1]

Abstract. Discrete-continuous project scheduling problems with positive discounted cash flows and the maximization of the *NPV* are considered. We deal with a class of these problems with an arbitrary number of discrete resources and one continuous, renewable resource. Activities are nonpreemptable, and the processing rate of an activity is a continuous, increasing function of the amount of the continuous resource allotted to the activity at a time. Three common payment models – Lump Sum Payment, Payments at Activity Completion times, and payments in Equal Time Intervals are analyzed. Formulations of mathematical programming problems for an optimal continuous resource allocation for each payment model are presented. Applications of two local search metaheuristics – Tabu Search and Simulated Annealing are proposed. The algorithms are compared on a basis of computational experiments. Some conclusions and directions for future research are pointed out.

Keywords. Discrete-continuous project scheduling, discounted cash flows, net present value, payment models, nonlinear programming, metaheuristics, simulated annealing, tabu search.

Mathematics Subject Classification. 68M20, 90B35, 90C30, 90C59.

1. Introduction

In the classical well-known Resource-Constrained Project Scheduling Problem (RCPSP) (see [3, 4, 6, 10, 11, 19, 23] for surveys), as well as in the RCPSP with discounted cash flows (RCPSPDCF) (see [4,5,9,16]), discrete, renewable resources

Received April 4, 2013. Accepted September 17, 2013.

[1] Institute of Computing Science, Poznan University of Technology, Piotrowo 2, 60-965 Poznan, Poland. grzegorz.waligora@cs.put.poznan.pl

are only considered. However, in practice continuous resources can appear very often. They can be allotted to activities in arbitrary amounts from a given interval, *i.e.* in real numbers of units. Discrete-continuous project scheduling problems occur when activities of a project simultaneously require discrete and continuous resources for their execution. In this paper a class of these problems is considered, where the number of discrete resources is arbitrary, and there is one continuous, renewable resource, whose total amount available at a time is limited. Activities are nonpreemptable, and the processing rate of an activity is a continuous, increasing function of the amount of the continuous resource allotted to the activity at a time. A positive cash flow is associated with the execution of each activity, and the objective is to maximize the Net Present Value (NPV) of all cash flows of the project. Three common payment models are considered: Lump-Sum Payment at the completion of the project (LSP), Payments at Activity Completion times (PAC), and payments at Equal Time Intervals (ETI).

Discrete-continuous project scheduling problems with discounted cash flows were considered in [27, 30]. In the former paper, an implementation of the Tabu Search metaheuristic for the problems with the PAC model was presented. A performance analysis of the proposed algorithm was done, on a basis of comparing its results to the results generated by multi-start iterative improvement and random sampling. The results presented in that paper are the only computational results for discrete-continuous project scheduling problems with discounted cash flows up to now. In the latter paper, several analytical results were given, including analyses of other payment models. It was shown in those two papers that for concave processing rate functions of activities local search algorithms, including metaheuristics, can be applied to attack the considered problems. In this paper we continue this research, and propose a comparison between two metaheuristics: Tabu Search and Simulated Annealing. The goal of the paper is, firstly, to compare those earlier results to results obtained by another efficient metaheuristic and, secondly, to extend the analysis by incorporating two other payment models (LSP and ETI). As it is known, metaheuristic approaches have been successfully applied over the years to many project scheduling problems, both in single- (see [17, 18]), and in the multi-mode version (see [8, 33]). We have applied Simulated Annealing for comparison to Tabu Search since our implementations of SA proved to be very efficient for discrete project scheduling, both to minimize the makespan [12] and to maximize the NPV [21].

The paper is organized as follows. Section 2 contains the problem formulation. Section 3 defines the payment models considered in the paper. In Section 4 the methodology for solving the problems is described, and the formulations of the mathematical programming problems finding optimal continuous resource allocations for the considered payment models are presented. Section 5 is devoted to the local search approach, whereas Section 6 to the implementations of metaheuristics. In Section 7 the computational experiment is described, and the analysis of the obtained results is included. Some conclusions and directions for future research are given in Section 8.

2. Problem formulation

The *Discrete-Continuous Resource-Constrained Project Scheduling Problem with Discounted Cash Flows* (DCRCPSPDCF) is defined as follows [30]. Given is a project consisting of n precedence-related, nonpreemptable activities, which require renewable resources of two types: discrete and continuous ones. We assume that R discrete resources are available and r_{il}, $i = 1, 2, \ldots, n$; $l = 1, 2, \ldots, R$, is the (fixed) discrete resource request of activity A_i for resource l. The total number of units of discrete renewable resource l available in each time period is R_l, $l = 1, 2, \ldots, R$. The activities are subject to finish-to-start precedence constraints of the type $A_i \rightarrow A_j$ with zero minimum time lags. The precedence constraints of activity A_i with other activities are defined by two sets: set Pred_i of direct predecessors of activity A_i, and set Succ_i of direct successors of activity A_i. One continuous, renewable resource is available. The availability of the continuous resource over time is constant and equal to 1. The resource can be allotted to activities in (arbitrary) amounts from the interval [0,1]. The amount (unknown in advance) of the continuous resource allotted to activity A_i at time t is denoted by $u_i(t)$, and $\sum\limits_{i=1}^{n} u_i\left(t\right) = 1$ for any t. The processing rate of activity A_i is defined by the resource amount $u_i(t)$ allotted to the activity at time t, which is described by the following equation:

$$\dot{x}_i\left(t\right) = \frac{\mathrm{d}x_i(t)}{\mathrm{d}t} = f_i\left[u_i(t)\right] x_i\left(0\right) = 0, x_i\left(C_i\right) = \tilde{x}_i \tag{2.1}$$

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 continuous resource amount 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.

There is a cash flow CF_i associated with each activity A_i. Thus, each activity of the project is characterized by its processing demand, processing rate function, discrete resource requests, precedence constraints with other activities, and its cash flow. 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 maximize the NPV. The continuous resource allocation is defined by a piecewise continuous, nonnegative vector function $\mathbf{u}\left(t\right) = \left[u_1\left(t\right), u_2\left(t\right), \ldots, u_n(t)\right]$, whose values $\mathbf{u}^* = [u_1^*, u_2^*, \ldots, u_n^*]$ are optimal continuous resource allocations corresponding to NPV^* – the optimal value of the NPV. Following the classification given in [4], the notation of the general DCRCPSPDCF is m,1|*cpm*,*cont*,c_j|*npv*. As we consider positive cash flows in this paper, the notation of the problem can be specified as m, 1|*cpm*, *cont*, c_j^+|*npv*. All the problem parameters are summarized in Table 1.

Table 1. Parameters of the DCRCPSPDCF.

Symbol	Definition
n	number of activities
$A_i \rightarrow A_j$	precedence constraint between activities A_i and A_j
Pred_i	set of direct predecessors of activity A_i
Succ_i	set of direct successors of activity A_i
R	number of discrete renewable resources
R_l	number of available units of discrete resource l
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
CF_i	cash flow of activity A_i
S_i	starting time of activity A_i
C_i	completion time of activity A_i

3. Payment models

The payment model defines the method of payment going from the client (the owner of the project) to the contractor (the executor) for the execution of the project. The best situation for the client would be a single payment at the end of the project. The contractor, on the other hand, would like to receive the whole payment at the beginning of the project. They have to find a compromise between these two polar approaches. Several payment models have been considered in the project scheduling literature, like, *e.g.*, Lump-Sum Payment at the completion of the project (LSP), Payments at Activity Completion times (PAC), payments at Equal Time Intervals (ETI), or Progress Payments (PP) (see [21,25]). The ETI and PP models belong to the class of periodic payments, where the payment moments do not depend on starting or completion times of activities but they are directly related to time representing the progress of a project. It was shown in [30] that the DCRCPSPDCF under the ETI and PP models has similar properties. For that reason, the PP model will not be explicitly considered in this paper, whereas the ETI model will represent the case of periodic payments. The three models analyzed in the paper are briefly described in the next three subsections.

In financial analyses of a project, time value of money is taken into account by discounting the cash flows. In order to calculate the NPV, a *discount rate* α has to be chosen, which represents the return following from investing in the project. Then $\beta = (1+\alpha)^{-1}$ is the *discount factor* under the assumed discount rate.

3.1. Lump-sum payment

Lump-Sum Payment (LSP) represents the ideal situation for the client since, in this case, the whole payment is paid by the client to the contractor at the moment of the successful completion of the project. The total payment is calculated as the sum of the cash flows of all activities. As a result, the *NPV* is given by the following formula:

$$NPV = \left(\sum_{i=1}^{n} CF_i\right)\beta^{C_{\max}} \tag{3.1}$$

where $C_{\max}$ is the completion time of the project (*i.e.* the project makespan).

It is easy to see that if all cash flows are positive, $\sum_{i=1}^{n} CF_i$ is a constant positive value. Since β is always smaller than 1 ($\beta < 1$), then in this model the maximization of the *NPV* is equivalent to the minimization of $C_{\max}$.

3.2. Payments at activity completion times

Payments at Activity Completion times (PAC) is a very practical payment model, where the client pays the contractor for the completion of each activity of the project. Once an activity is finished, the contractor gets the amount of money equal to the cash flow associated with this activity. In this case, the *NPV* is given by the following formula:

$$NPV = \sum_{i=1}^{n} CF_i \beta^{C_i} \tag{3.2}$$

3.3. Equal time intervals

In the Equal Time Intervals (ETI) model the client makes H payments for the execution of the project. The first H–1 payments are scheduled at equal time intervals over the course of the project, and the last payment is made at its completion. In this case the *NPV* can be given by:

$$NPV = \sum_{h=1}^{H} P_h \beta^{T_h} \tag{3.3}$$

where P_h is the payment at payment point h, $h = 1, 2, \ldots, H$, and T_h is the occurrence time of payment point h. It must be stressed that, since the number of payments is assumed *a priori*, the makespan $C_{\max}$ has to be fixed in order to calculate the payment period T. Then T can be calculated as the smallest integer greater or equal to $C_{\max}/H$, and $T_h = h \cdot T$, $h = 1, 2, \ldots, H - 1$. Obviously, for the last payment point H, $T_H = C_{\max}$. In order to compute the payments P_h at successive payment points, partial cash flows (linearly proportional to activity durations) of all the activities executed in the considered interval (*i.e.* since the last payment point) are summed up. As a result, in general, each payment P_h can be different since it depends on the cash flows of the relevant activities.

4. Methodology

4.1. General methodology

The methodology for solving the DCRCPSPDCF critically depends on the form of the processing rate functions of activities. It was proved in [30] that for processing rate functions of activities fulfilling $f_i(u_i) \leqslant c_i u_i, c_i = f_i(1), i = 1, 2, \ldots, n$ (*i.e.* including all convex functions), in an *NPV*-optimal schedule activities are processed sequentially, each of them using the total available amount of the continuous resource. As a result, the problem reduces to a sequencing problem to find an optimal order of activities in a sequential schedule. Several analytical results concerning this class of functions were given in [30]. The case of convex functions will not be considered in this paper. It was also shown in the latter paper that for concave functions any regular performance measure is optimized by fully parallel precedence- and discrete resource-feasible configuration of all activities, and a special methodology based on feasible sequences has to be applied in this case. The methodology, originally developed in [15] for discrete-continuous machine scheduling, is of crucial importance, and will be briefly recalled below. As it is known, the *NPV* maximization under positive cash flows is a regular performance measure, and, consequently, the abovementioned methodology can be used here.

Let us start with the definition of a *feasible sequence*. Observe that each feasible schedule can be divided into $s \leqslant n$ intervals of lengths M_k, $k = 1, 2, \ldots, s$, defined by completion times of consecutive activities. Let Z_k denote the combination of activities processed in parallel in the kth interval. Thus, a sequence 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;
- nonpreemptability of each activity is guaranteed which means that each activity appears in exactly one or in successive combinations in S;
- precedence constraints between activities are satisfied;
- the number of units of each discrete resource l, $l = 1, 2, \ldots, R$, assigned to all activities in combination Z_k, $k = 1, 2, \ldots, s$, does not exceed R_l.

Example.

Consider an 8-activity project ($n = 8$). Assume that there is one discrete resource ($R = 1$) available in 4 units ($R_1 = 4$). The resource requests of activities are defined by vector $\mathbf{r}_1 = [3,2,1,1,3,1,4,2]$. For simplicity, let us associate each activity with its index, *i.e.* activity A_i will be called shortly activity i. Assume next that the precedence constraints are: $1 \to 2$, $1 \to 3$, $1 \to 4$, $2 \to 5$, $3 \to 7$, $4 \to 6$, $5 \to 7$, $5 \to 8$, and $6 \to 7$.

There are many feasible sequences for this exemplary project. One of them is, *e.g.*:

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

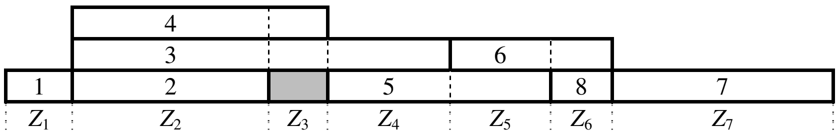

FIGURE 1. Schedule corresponding to feasible sequence S.

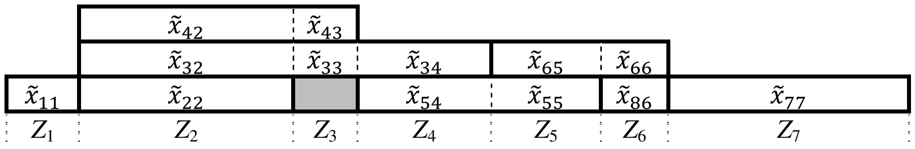

FIGURE 2. Demand division for feasible sequence S.

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 is $Z_1 = \{1\}$. 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 is $Z_2 = \{2,3,4\}$. 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 $A_{7,}$ and this is the end of the schedule. The resulting last combination is $Z_7 = \{7\}$. A schedule corresponding to the considered feasible sequence S is shown in Figure 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 allocated on a basis on that information.

Next, for a given feasible sequence S, the processing demand $\tilde{x}_i$ of each activity A_i, $i = 1, 2, \ldots, n$, can be divided into parts $\tilde{x}_{ik} \geqslant 0$ (unknown in advance) corresponding to particular time intervals (combinations), *i.e.* $\tilde{x}_{ik}$ is a part of activity A_i processed in combination Z_k. Such a division of processing demands of activities among successive intervals (combinations) for a given feasible sequence is called a *demand division.* The number of such divisions is, in general, infinite. Figure 2 shows the demand division for the feasible sequence S considered in the above example. Obviously, the sum of all parts of the processing demand of an activity must be equal to its total processing demand, *i.e.* $\tilde{x}_{11} = \tilde{x}_1,\ \tilde{x}_{22} = \tilde{x}_2,\ \tilde{x}_{32} + \tilde{x}_{33} + \tilde{x}_{34} = \tilde{x}_3,$ *etc.*

Now, a nonlinear mathematical programming problem can be formulated which finds an optimal division of processing demands of activities $\tilde{x}_i$, $i = 1, 2, \ldots, n$, among combinations in S, *i.e.* a division that leads to an *NPV*-optimal schedule from among all feasible schedules generated by S. Let M_k^* be the minimal length of the part of the schedule generated by combination $Z_k \in S$, $k = 1, 2, \ldots, s$, 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 $A_i \in Z_k$. In the following subsections the nonlinear mathematical programming problems are presented, originally formulated in [30], each of them finding an *NPV*-optimal demand division (and, in consequence, an optimal continuous resource allocation) for a given feasible sequence and an assumed payment model.

4.2. Lump-sum payment

For the LSP model the *NPV* is given by formula (3.1). In consequence, the following mathematical programming problem can be formulated:

Problem $P_{NPV-LSP}$
maximize

$$NPV = \left(\sum_{i=1}^{n} CF_i\right) \beta^{C_{\max}} \tag{4.1}$$

subject to

$$\sum_{k \in K_i} \tilde{x}_{ik} = \tilde{x}_i\,, \quad i = 1, 2, \ldots, n \tag{4.2}$$

$$\tilde{x}_{ik} \geqslant 0\,,\ i = 1, 2, \ldots, n; k \in K_i \tag{4.3}$$

$$C_{\max} = \sum_{k=1}^{s} M_k^*(\tilde{\mathbf{x}}_k) \tag{4.4}$$

where $M_k^*\left(\tilde{\mathbf{x}}_k\right)$ is the unique positive root of the equation:

$$\sum_{A_i \in Z_k} f_i^{-1}\left(\tilde{x}_{ik}/M_k\right) = 1 \tag{4.5}$$

Constraints (4.2) correspond to the condition of fulfilling the processing demands of all activities, whereas constraints (4.3) ensure that the $\tilde{x}_{ik}$'s are nonnegative. $C_{\max}$ is calculated in (4.4) as the sum of the lengths of all the intervals of the schedule. Equation (4.5) allows to calculate the minimal length of the kth interval following from an optimal continuous allocation [31]. It can be solved analytically for some important cases, *e.g.* linear or power functions. From among them the ones in which eq. (4.5) is an algebraic equation of an order $\leqslant 4$ are of special importance. This is, *e.g.*, the case of power processing functions of the form: $f_i\left(u_i\right) = c_i u_i^{1/a_i}$, $a_i \in \{1, 2, 3, 4\}$, $i = 1, 2, \ldots, n$. Using these functions we can model job processing rates in a variety of practical problems, *e.g.* those arising in multiprocessor scheduling with memory allocation (see [2,32]). Obviously, as stressed in Section 3.1, since $\beta < 1$ then in this case maximizing the *NPV* results in minimizing $C_{\max}$.

4.3. Payments at activity completion times

In this case the *NPV* is given by formula (3.2), and the following mathematical programming problem is formulated:

Problem $P_{NPV-PAC}$
maximize

$$NPV = \sum_{i=1}^{n} CF_i \beta^{C_i} \tag{4.6}$$

subject to

$$\sum_{k \in K_i} \tilde{\mathbf{x}}_{ik} = \tilde{x}_i \,, \quad i = 1, 2, \ldots, n \tag{4.7}$$

$$x_{ik} \geqslant 0 \,, \; i = 1, 2, \ldots, n; k \in K_i \tag{4.8}$$

$$C_i = \sum_{k=1}^{\max\{K_i\}} M_k^*(\tilde{\mathbf{x}}_k) \quad i = 1, 2, \ldots, n \tag{4.9}$$

where $M_k^*(\tilde{\mathbf{x}}_k)$ is the unique positive root of equation (4.5).

Objective function (4.6) maximizes the *NPV* for the PAC model. Constraints (4.7) and (4.8) have already been discussed in Section 4.2. Completion time of each activity is calculated according to equation (4.9) as the sum of the lengths of all intervals from the first one up to the interval in which the considered activity is finished.

4.4. Equal time intervals

Under the *NPV* given by formula (3.3), the following mathematical programming problem arises:

Problem $P_{NPV-ETI}$
maximize

$$NPV = \sum_{h=1}^{H} P_h \beta^{T_h} \tag{4.10}$$

subject to

$$\sum_{k \in K_i} \tilde{x}_{ik} = \tilde{x}_i, i = 1, 2, \ldots, n \tag{4.11}$$

$$\mathbf{x}_{ik} \geqslant 0 \,, \; i = 1, 2, \ldots, n; k \in K_i \tag{4.12}$$

$$C_{\max} = \sum_{k=1}^{s} M_k^*(\tilde{\mathbf{x}}_k) \tag{4.13}$$

$$T = \left\lceil \frac{C_{\max}}{H} \right\rceil \tag{4.14}$$

$$T_h = h \cdot T,\ h = 1, 2, \ldots, H-1; \qquad (4.15)$$
$$T_H = C_{\max}$$

where $M_k^*(\tilde{\mathbf{x}}_k)$ is the unique positive root of equation (4.5), H is the assumed number of payments, T in (4.14) is the length of the interval between payments, T_h in (4.15) is the time point of the hth payment, and P_h is the payment at time point T_h.

Let us now explain how the values of P_h are calculated. After the allocation of the continuous resource the actual starting time S_i and completion time C_i of each activity A_i can be defined. The values of T_h determine the intervals in which the partial cash flows of activities processed in those intervals are summed up. It is easy to find in which such interval a particular activity is started, in which it is processed, and in which it is completed. Then the partial cash flow P_{ih} of activity A_i at time point T_h is calculated as:

a) $P_{ih} = \frac{C_i - T_{h-1}}{C_i - S_i} \cdot CF_i \iff (S_i < T_{h-1} \,\text{and}\, T_{h-1} < C_i < T_h)$;
b) $P_{ih} = \frac{T_h - T_{h-1}}{C_i - S_i} \cdot CF_i \iff (S_i \leqslant T_{h-1} \,\text{and}\, C_i \geqslant T_h)$;
c) $P_{ih} = CF_i \iff (S_i \geqslant T_{h-1} \,\text{and}\, C_i \leqslant T_h)$;
d) $P_{ih} = \frac{T_h - S_i}{C_i - S_i} \cdot CF_i \iff (T_{h-1} < S_i < T_h \,\text{and}\, C_i > T_h)$.

where $h = 1, 2, \ldots, H$, and $T_0 = 0$.

In case a) activity A_i starts before interval $\langle T_{h-1}; T_h\rangle$ and finishes within it. In case b) activity A_i is processed over the whole interval but may be started earlier and/or completed later. Case c) concerns a situation where the entire activity is executed within interval $\langle T_{h-1}; T_h\rangle$. Finally, in case d) activity A_i starts within interval $\langle T_{h-1}; T_h\rangle$ but finishes beyond it. It is easy to see that the special case when $S_i = T_{h-1} \,\text{and}\, C_i = T_h$ can be classified both as case b) and as case c), however, both the situations lead to the same result $P_{ih} = CF_i$ since the entire activity A_i is processed within the considered interval.

5. Local search

As described in Section 4.1, the solution of Problem P_{NPV-X} (X $\in$ {LSP, PAC, ETI}), allows to find an optimal continuous resource allocation for a given feasible sequence an assumed payment model. Consequently, the DCRCPSPDCF can be decomposed 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 may 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, *e.g.* processing demands of activities, 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]. In general, in the continuous part of the problem there appears a nonlinear objective function of continuous variables, whose values may be irrational numbers. In consequence, the complexity of the DCRCPSPDCF, 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 DCRCPSPDCF into two subproblems (as discussed above) is of huge importance. Firstly, 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. Secondly, notice that an optimal solution can be found by solving Problem P_{NPV-X} for all feasible sequences, and choosing the one with the maximum NPV. This full enumeration approach can be applied for small problem sizes, and guarantees finding an optimal schedule. However, 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. In [27] the Tabu Search metaheuristic for the PAC model was examined. In this paper we propose a comparison between Tabu Search and Simulated Annealing for three different payment models.

In this section we present the basic features of the proposed local search approach, *i.e.*, representation of a feasible solution, calculation of the objective function, generation of a starting solution, neighbourhood generation mechanism, and stop criterion.

5.1. Solution representation

A feasible solution for a local search algorithm corresponds to a feasible sequence, and is represented by two n-element lists. The first one is a precedence-feasible permutation of activities, in which each activity has to occur after all its predecessors and before all its successors. This structure is called the *Sequence of Activity Starts* (SAS), and defines the order in which activities are started. The second one is also a precedence-feasible permutation of activities, and defines the order in which activities should be completed in order to fulfill the discrete resource constraints. This list is called the *Sequence of Activity Completions* (SAC). Let us stress that SAC does not define the actual order in which activities will be finished. It is used when introducing a new activity to a combination violates the discrete resource constraints and, in order to maintain feasibility of the sequence, some activities must be finished before this new one is started. In such a case, activities are finished according to the SAC list. This two-list representation was previously used for the Tabu Search algorithm presented in [27].

A pair of lists (SAS, SAC) is then transformed to a feasible sequence S according to the rule:

- in the first combination Z_1 successive activities without predecessors from SAS are inserted into this combination as long as the discrete-resource constraints are satisfied;
- in every next combination Z_k, $k = 2, \ldots, s$, one successive activity A_i from SAS is added, and first – all predecessors of activity A_i are removed from Z_k, and second – if adding activity A_i to Z_k has caused a violation of the discrete-resource constraints, then successive activities are removed from Z_k in the order defined by SAC until the discrete-resource constraints are satisfied (in an extreme case, all activities previously occurring in Z_k may be removed, and $Z_k = \{i\}$).

5.2. Objective function

The objective function for a feasible solution is defined as the maximum NPV for the corresponding feasible sequence and the assumed payment model, obtained as a solution of the relevant nonlinear mathematical programming problem.

5.3. Starting solution

Initial SAS and SAC lists are generated by setting activities on both the lists in an ascending order.

5.4. Neighbourhood

Neighbours of the current solution are generated by swapping activities on lists SAS and SAC. More precisely, an activity swap operator is used (already applied in [27]), which swaps two activities on the list that may be swapped without violating the precedence constraints. This operator can be applied to both the SAS and the SAC list.

5.5. Stop criterion

The stop criterion has been defined as an assumed number of visited solutions, *i.e.* an assumed number of the objective function calculations, in order to assure a comparable computational effort for each search algorithm.

6. Metaheuristics

In this section we describe the implementations of the two applied metaheuristics: Tabu Search and Simulated Annealing.

6.1. Parameter settings

Let us first discuss briefly a few details concerning the implementations.

As mentioned in the Introduction, in [27] a Tabu Search algorithm was proposed for the PAC model. During that research some preliminary experiments were carried out concerning the influence of the length of the tabu list on the performance of the algorithm. After those experiments a decision was made about this important parameter, since its setting resulted in the average best performance over the assumed numbers of activities. Since the aim of this work is to compare that TS implementation with another efficient metaheuristic, we have kept the setting undertaken there.

On the other hand, our SA algorithm was very carefully tuned for the experiments concerning discrete project scheduling problems, described in [12, 21], with similar numbers of activities. Apart from the problem specific decisions, we have used the same cooling scheme with the same parameter settings in this work, since both the mentioned implementations appeared to be very efficient for the considered classes of problems. Below, in Sections 6.2 and 6.3, we address the implementation issues of both the metaheuristics in more details.

6.2. Tabu search

Tabu Search (TS) is a metastrategy based on neighbourhood search with overcoming local optimality. Unlike Simulated Annealing, TS works in a deterministic way, trying to model human memory processes. Memory is implemented by the implicit recording of previously visited solutions, using simple but effective data structures. To this end, a tabu list of moves which have been performed in the recent past of search is created, and these moves are forbidden for a certain number of iterations. This helps to avoid cycling, and also serves to promote a diversified search over the set of feasible solutions. A comprehensive report of the basic concepts and developments of TS was given in [7]. Implementations of the Tabu Search metaheuristic have shown their good efficiency for several project scheduling problems, including discrete-continuous ones, we have dealt with in the past (see [13, 14, 21, 22, 27, 28]).

Neighbourhood

In the case of TS all possible activity swaps are performed on both the lists SAS and SAC. As a result, a set of neighbouring solutions is created, each of them differing from the current solution in two positions on SAS, or in two positions on SAC.

A move is represented by the following three attributes:

$$(\textit{position, activity removed, activity inserted})$$

where position denotes the position of the replaced activity on the merged list [SAS, SAC], *i.e.* positions from 1 to n concern SAS, positions from $n+1$ to $2n$ concern SAC. From among two activities involved in the swapping operation, the smaller value of position is taken, *i.e.* concerning the activity occurring closer to front of the list.

Assume that activity A_2 in position 3 is swapped with activity A_6 in position 8 (position number $< n$, thus swapping is performed on SAS), then the attributes of the performed move are (3,2,6) which means that in position 3 activity A_2 has been replaced by activity A_6. In consequence, the reverse move should forbid activity A_2 to go back into position 3, and has the form of the couple (3,2).

Tabu list management

The tabu list is managed by the Tabu Navigation Method (TNM) [24]. The tabu list (TL) is a queue of a given length. Whenever a move is performed, its reverse is added to the TL in order to avoid going back to a solution already visited, and, at the same time, the oldest existing move is removed from the front of the TL (according to the FIFO policy). Each move existing on the TL is tabu. The TL length has been set at 7, as it has been justified in Section 6.1.

Aspiration critrion

In the case of the TNM, it may happen that a move existing on the TL does not allow to reach a solution that has not been visited yet. In order to avoid a situation where a good solution is overlooked, an aspiration criterion has been applied which allows the algorithm to move to a tabu solution (perform a tabu move) if this solution is better than the best found so far. If this is the case, the tabu move performed is removed from the TL (the oldest move stays on the TL), and the reverse move is added to the end of the TL. Obviously, performing such a move cannot lead to a solution visited in the past, because in this case the obtained objective function value would already have been known.

6.3. Simulated annealing

Simulated Annealing (SA) is a well-known local search metaheuristic which belongs to a class of the threshold algorithms, and can be viewed as a special case of the First Fit Strategy, where the next solution is accepted with a certain probability. Its idea was originally used to simulate a physical annealing process, and was applied to combinatorial optimization for the first time in the 1980's. Most adaptations of the SA algorithm use its homogeneous version [26]. As mentioned in Section 6.1, the implementation of the Simulated Annealing algorithm described below is based on the ones previously applied to discrete multi-mode project scheduling problems presented in [12, 21].

Neighbour solution

A neighbour of the current solution is generated by randomly choosing one of the lists SAS and SAC, and performing the activity swap on two randomly selected activities on the chosen list.

Cooling scheme

The adaptive cooling scheme, known as polynomial-time [1], is used to control the cooling process of the SA algorithm. The only exception introduced into this implementation of SA is the stop criterion, set at a fixed number of visited solutions.

The initial value T_0 of the control parameter is calculated during the initialization phase from the following equation:

$$T_0 = \frac{\bar{\Delta f}^{(+)}}{\ln\left(\frac{m_2}{m_2 \cdot \chi_0 - m_1(1-\chi_0)}\right)} \tag{6.1}$$

where χ_0 is the initial acceptance ratio (the assumed proportion between transitions accepted and all the transitions generated for T_0); m_1 is the number of cost-nondecreasing transitions and m_2 is the number of cost-decreasing transitions from among m_0 ($m_0 = m_1 + m_2$) trial transitions generated to determine the initial value T_0 of the control parameter. $\bar{\Delta f}^{(+)}$ is the average difference in cost over the m_2 cost-decreasing transitions. We have assumed that $\chi_0 = 0.95$ and $m_0 = 50$.

The next value T_{k+1} of the control parameter calculated during the cooling process depends on the mean and the standard deviation σ_{T_k} for the values of the objective function for the kth Markov chain. The value T_{k+1} is calculated using the following formula:

$$T_{k+1} = \frac{T_k}{1 + \frac{T_k \cdot \ln(1+\delta)}{3 \cdot \sigma_{T_k}}}\,,\ k = 0,\, 1,\, \ldots \tag{6.2}$$

where δ is a real number denominating the distance parameter. Usually, smaller values of this parameter lead to better solutions but also increase the computation time. We have assumed that $\delta = 0.5$.

The length of the Markov chain in the homogeneous version of SA determines the number of transitions for a given value of the control parameter. It is assumed that this value is constant, and depends on the problem size. The length of kth Markov chain is calculated according to the rule:

$$L_k = L = n\,(n/2)\,,\ k = 0, 1, \ldots \tag{6.3}$$

where n is the number of activities.

7. Computational experiment

In this section we present the results of a computational experiment concerning the implementations of the two metaheuristics for the considered DCRCPSPDCF. The implementations were coded and compiled in C++ and ran on an SGI Altix 3700 machine with 128 64-bit Intel Itanium2 processors and 768 GFlops overall computing power, installed in the Poznan Supercomputing and Networking Center.

As mentioned before, for each solution visited in the solution space, the corresponding *NPV*-optimal schedule was found by solving the suitable mathematical programming problem. In this step specially adapted solver CFSQP 2.5 – "A C Code for Solving (Large Scale) Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying All Inequality Constraints" [20] was applied. The solver stopped when the absolute difference in consecutive values of the objective function was less than or equal to 10^{-3}. As mentioned in Section 5.5, in order to ensure a comparable computational effort devoted to both the algorithms, the stop criterion was defined as a number of solutions visited. This number was set at 1000. The experiment was carried out for processing rate functions of activities of the form $f_i = u_i^{1/a_i}$, where the values of a_i from the set $\{1,2\}$were generated randomly with equal probability. Processing demands of activities were generated as integers from the interval [1,100], whereas their cash flows as integers from the interval [1,1000], each time with a uniform distribution. Precedence constraints between activities were also set randomly. The average density of the precedence graph was 0.5.

The experiment is divided in two parts.

In the first experiment a comparison to optimal solutions is made. To this end, instances with optimal solutions known, used for the research presented in [27], were applied again. These are 50 instances for $n = 10$ activities, one discrete resource with $R_1 = 5$ available resource units, and the PAC model. Additionally, a second group of 50 instances with the same parameters but for $n = 8$ activities have been generated and solved to optimality by the full enumeration method. In these two cases the results produced by SA and TS are compared to optimal solutions. Finding optimal solutions for a larger number of activities would require an unreasonable computational effort, and is, in fact, unrealistic in practice.

The second experiment was performed for $n = 12$, 15, and 20 activities. In this experiment three different payment models described in Section 3.1–3.3 were examined. Also one discrete resource was considered ($R = 1$) but various numbers of its available units were tested: $R_1 \in \{2,5,10\}$. The values of r_{i1} (discrete resource requests of activities) were generated randomly as integers from the interval [0, $R_1/2$]. Also different values of the discount rate α were assumed – $\alpha \in\{0.01, 0.02, 0.05\}$. For each set of problem parameters (*i.e.* set of values of n, R_1, α), 100 instances were generated, characterized by the values of $\tilde{x}_i$, a_i, r_{i1}, CF_i, $i = 1, 2, \ldots, n$. For the periodic payment model (ETI) also different values of the H parameter were tested. The number of payments H was assumed to

Table 2. Results of the experiment – comparison to optimal solutions.

		SA			TS		
n	α	#	ARD [%]	MRD [%]	#	ARD [%]	MRD [%]
8	0.01	44	0.07	0.41	31	0.17	0.83
	0.02	43	0.08	0.51	31	0.15	0.79
	0.05	41	0.10	0.59	35	0.10	0.71
10	0.01	39	0.09	0.56	30	0.18	0.92
	0.02	40	0.10	0.64	32	0.16	0.81
	0.05	38	0.10	0.69	37	0.11	0.77

be equal to 2, 3, and 5. In the second experiment optimal solutions are not known, thus, the relative solution is the best one found.

The results of the experiment are presented in Tables 2–5. For each algorithm the following numbers are shown:

- # – the number of instances for which the algorithm found a solution equal to the best solution known;
- ARD – the average relative deviation from the best solution known;
- MRD – the maximal relative deviation from the best solution known.

where in Table 2 the best solution known is an optimal solution, whereas in Tables 3–5 it is the best solution found by either of the algorithms tested.

The average CPU times (in seconds) for the considered problem sizes and payment models are given in Table 6. The presented values are times needed by the relevant algorithm to solve one instance of the corresponding problem (*i.e.* to visit 1000 feasible solutions). As can be seen, the computational times of the compared algorithms are very similar, since the overwhelming majority of the computational effort was devoted to calculating the objective function, *i.e.* finding a solution of the appropriate nonlinear programming problem by the CFSQP solver. However, it can also be noticed that SA is slightly faster.

On the basis of the obtained results, the following observations can be made.

As to the comparison to optimal solutions for 8 and 10 activities, the results obtained by both the metaheuristics are very promising, although it is visible that SA performs better. However, 30–40 instances out of 50 solved to optimality, keeping the average relative deviation from optimum below 0.1% for SA and 0.2% for TS, may suggest that both the proposed implementations can be quite effective for the considered problems.

Analyzing all the obtained results, we can draw the following conclusions:

a) SA performs better for smaller number of activities. Along with the growth of the problem size, TS becomes more and more competitive, and starts to achieve some advantage for the problems with 20 activities. This is an interesting observation which suggests that the TS implementation is rather predisposed for larger DCRCPSPDCF problems.

TABLE 3. Results of the experiment for the LSP model.

			SA			TS		
n	R_1	α	#	ARD [%]	MRD [%]	#	ARD [%]	MRD [%]
12	2	0.01	80	0.04	0.10	23	0.14	0.37
		0.02	78	0.04	0.11	24	0.12	0.35
		0.05	76	0.06	0.15	28	0.11	0.34
	5	0.01	81	0.03	0.11	25	0.15	0.38
		0.02	79	0.05	0.11	26	0.11	0.35
		0.05	78	0.06	0.14	28	0.10	0.35
	10	0.01	81	0.04	0.09	24	0.16	0.35
		0.02	80	0.05	0.12	27	0.12	0.32
		0.05	76	0.06	0.14	27	0.11	0.29
15	2	0.01	71	0.06	0.18	39	0.11	0.29
		0.02	68	0.06	0.19	41	0.09	0.28
		0.05	67	0.08	0.21	42	0.08	0.24
	5	0.01	70	0.05	0.19	40	0.10	0.28
		0.02	70	0.06	0.19	42	0.08	0.27
		0.05	65	0.07	0.25	43	0.07	0.25
	10	0.01	69	0.04	0.20	38	0.10	0.28
		0.02	68	0.05	0.21	40	0.07	0.26
		0.05	64	0.07	0.22	41	0.07	0.22
20	2	0.01	58	0.07	0.24	50	0.08	0.25
		0.02	57	0.08	0.25	51	0.08	0.25
		0.05	52	0.10	0.27	54	0.07	0.24
	5	0.01	56	0.07	0.26	51	0.09	0.28
		0.02	55	0.09	0.28	53	0.07	0.27
		0.05	51	0.11	0.29	53	0.06	0.25
	10	0.01	59	0.09	0.24	52	0.10	0.25
		0.02	57	0.10	0.25	53	0.08	0.24
		0.05	53	0.12	0.26	56	0.07	0.22

b) The results produced by TS get better with the growth of the discount rate, whereas SA is better for smaller values of α. A similar observation was made in [21] for the MRCPSPDCF (Multi-Mode Resource-Constrained Project Scheduling Problem with Discounted Cash Flows). Increasing the discount rate results in bigger differences in the objective function between different solutions, which is advantageous for Tabu Search since, as known, TS does not prefer a flat objective function.

c) SA is significantly better for the LSP model. This results can be explained by the fact that, as mentioned before, under positive cash flows the NPV maximization is equivalent to the minimization of $C_{\max}$. Since our SA implementation is known to be one of the most effective for the makespan minimization project scheduling problems, it has also produced very good results for the LSP model. For the PAC model both the algorithms are rather comparable, taking into account remarks a) and b) pointed above. The same concerns, in fact, the ETI model, however, in this case another interesting regularity can be

TABLE 4. Results of the experiment for the PAC model.

			SA			TS		
n	R_1	α	#	ARD [%]	MRD [%]	#	ARD [%]	MRD [%]
12	2	0.01	71	0.05	0.12	33	0.12	0.34
		0.02	71	0.05	0.14	34	0.11	0.32
		0.05	68	0.06	0.18	37	0.11	0.30
	5	0.01	70	0.04	0.13	34	0.13	0.35
		0.02	69	0.06	0.13	36	0.10	0.35
		0.05	65	0.07	0.17	39	0.09	0.33
	10	0.01	69	0.05	0.11	31	0.12	0.31
		0.02	69	0.06	0.15	32	0.10	0.30
		0.05	61	0.07	0.16	37	0.08	0.27
15	2	0.01	55	0.07	0.21	54	0.09	0.26
		0.02	53	0.07	0.22	54	0.07	0.22
		0.05	52	0.08	0.25	58	0.06	0.21
	5	0.01	54	0.06	0.21	55	0.07	0.24
		0.02	54	0.06	0.20	56	0.07	0.24
		0.05	49	0.08	0.28	59	0.06	0.22
	10	0.01	52	0.05	0.22	54	0.08	0.23
		0.02	51	0.06	0.23	55	0.07	0.21
		0.05	48	0.08	0.26	57	0.05	0.20
20	2	0.01	38	0.09	0.30	69	0.05	0.17
		0.02	35	0.09	0.30	69	0.05	0.16
		0.05	31	0.11	0.33	72	0.04	0.13
	5	0.01	37	0.08	0.29	68	0.06	0.15
		0.02	32	0.09	0.31	69	0.05	0.15
		0.05	31	0.12	0.32	70	0.05	0.12
	10	0.01	35	0.10	0.28	67	0.06	0.14
		0.02	33	0.11	0.29	69	0.06	0.11
		0.05	32	0.12	0.32	74	0.04	0.09

noticed. We can see that the growth of the number of payments H improves the performance of TS, whereas SA is better when the payments are made less often. This fact confirms the rule that the more the problem approaches its makespan minimization version (by decreasing the number of payments), the better results are produced by SA. Theoretically, for $H = 1$ we would obtain the LSP model in which SA achieves the most significant advantage.

d) Remarks a) – c) hold generally for all values of R_1. We can state that the number of available discrete resource units does not have any vital impact on the results when the number of activities is fixed. The explanation seems to be quite simple. Both the metaheuristics search over the set of feasible sequences, in which the discrete resource constraints have to be already fulfilled. Thus, parameter R_1 is only taken into account during the phase of transformation a pair of the list (SAS, SAC) into a feasible sequence which transformation is identical for both the algorithms and does not make any important difference.

Table 5. Results of the experiment for the ETI model.

				SA			TS		
n	R_1	α	H	#	ARD [%]	MRD [%]	#	ARD [%]	MRD [%]
12	2	0.01	2	75	0.06	0.13	34	0.13	0.35
			3	74	0.07	0.14	36	0.12	0.32
			5	71	0.07	0.16	37	0.12	0.31
		0.02	2	74	0.06	0.15	35	0.11	0.33
			3	74	0.08	0.16	37	0.10	0.33
			5	72	0.09	0.18	37	0.10	0.30
		0.05	2	70	0.07	0.17	38	0.11	0.31
			3	70	0.08	0.17	39	0.11	0.30
			5	67	0.08	0.19	40	0.10	0.30
	5	0.01	2	70	0.05	0.14	35	0.12	0.34
			3	70	0.07	0.16	38	0.11	0.32
			5	66	0.08	0.18	39	0.10	0.30
		0.02	2	68	0.06	0.13	36	0.10	0.34
			3	67	0.07	0.15	38	0.09	0.33
			5	66	0.09	0.16	40	0.09	0.29
		0.05	2	65	0.07	0.18	38	0.08	0.33
			3	64	0.07	0.19	39	0.08	0.31
			5	61	0.09	0.19	41	0.08	0.30
	10	0.01	2	67	0.05	0.12	34	0.12	0.32
			3	65	0.06	0.14	37	0.11	0.29
			5	64	0.08	0.15	41	0.10	0.29
		0.02	2	67	0.07	0.16	38	0.09	0.30
			3	66	0.08	0.17	40	0.09	0.28
			5	63	0.09	0.18	41	0.08	0.27
		0.05	2	62	0.07	0.17	39	0.09	0.28
			3	62	0.07	0.19	41	0.08	0.27
			5	60	0.08	0.20	42	0.08	0.26
15	2	0.01	2	56	0.08	0.22	51	0.09	0.25
			3	55	0.09	0.23	52	0.09	0.24
			5	55	0.10	0.25	54	0.08	0.22
		0.02	2	54	0.08	0.21	55	0.08	0.21
			3	54	0.08	0.22	55	0.08	0.21
			5	52	0.09	0.24	56	0.07	0.20
		0.05	2	53	0.09	0.24	56	0.07	0.21
			3	51	0.10	0.25	56	0.07	0.20
			5	51	0.10	0.25	57	0.07	0.20
	5	0.01	2	55	0.07	0.20	51	0.08	0.25
			3	54	0.08	0.22	52	0.08	0.24
			5	52	0.09	0.24	53	0.07	0.23
		0.02	2	55	0.08	0.23	52	0.08	0.23
			3	54	0.09	0.24	54	0.07	0.23
			5	51	0.10	0.25	54	0.07	0.22
		0.05	2	51	0.09	0.28	55	0.07	0.22
			3	50	0.10	0.28	55	0.06	0.21
			5	50	0.11	0.29	57	0.06	0.20

TABLE 5. Continued.

				SA			TS		
n	R_1	α	H	#	ARD [%]	MRD [%]	#	ARD [%]	MRD [%]
	10	0.01	2	53	0.06	0.21	52	0.08	0.24
			3	52	0.08	0.22	54	0.08	0.22
			5	51	0.10	0.24	54	0.07	0.21
		0.02	2	50	0.08	0.24	53	0.07	0.22
			3	50	0.09	0.24	55	0.06	0.22
			5	49	0.09	0.25	56	0.06	0.20
		0.05	2	49	0.08	0.27	55	0.06	0.20
			3	48	0.09	0.28	55	0.06	0.20
			5	48	0.10	0.29	57	0.05	0.19
20	2	0.01	2	37	0.10	0.31	70	0.06	0.18
			3	36	0.11	0.33	71	0.06	0.17
			5	34	0.11	0.34	73	0.05	0.15
		0.02	2	34	0.10	0.32	70	0.05	0.15
			3	34	0.10	0.34	71	0.05	0.15
			5	32	0.12	0.34	73	0.05	0.14
		0.05	2	32	0.12	0.33	72	0.04	0.13
			3	30	0.12	0.34	73	0.04	0.13
			5	30	0.13	0.35	75	0.04	0.12
	5	0.01	2	38	0.09	0.29	69	0.06	0.16
			3	37	0.10	0.30	70	0.05	0.15
			5	35	0.11	0.32	72	0.05	0.14
		0.02	2	33	0.09	0.33	69	0.06	0.16
			3	32	0.11	0.34	71	0.05	0.15
			5	31	0.11	0.34	72	0.04	0.15
		0.05	2	30	0.13	0.35	72	0.05	0.13
			3	30	0.13	0.35	72	0.05	0.13
			5	29	0.14	0.36	73	0.04	0.12
	10	0.01	2	36	0.11	0.30	68	0.06	0.15
			3	35	0.12	0.32	69	0.06	0.14
			5	33	0.13	0.34	72	0.05	0.13
		0.02	2	34	0.11	0.29	68	0.06	0.10
			3	33	0.11	0.30	70	0.05	0.10
			5	32	0.13	0.32	71	0.05	0.09
		0.05	2	31	0.13	0.32	75	0.04	0.09
			3	31	0.13	0.34	75	0.04	0.09
			5	30	0.14	0.36	76	0.04	0.08

TABLE 6. CPU times [s].

	SA			TS		
n	LSP	PAC	ETI	LSP	PAC	ETI
8	998.47	1011.82	1013.24	1007.33	1016.51	1019.16
10	1592.44	1609.90	1613.88	1600.05	1612.72	1618.03
12	2989.89	2995.16	3001.93	2993.17	2998.84	3002.06
15	6027.51	6034.05	6036.22	6026.62	6033.06	6035.78
20	16 235.18	16 271.53	16 302.07	16 223.54	16 259.23	16 288.45

e) Finally, let us stress that the differences between results obtained by both the metaheuristics are very small because of the optimal continuous resource allocation done at the stage of calculating the objective function value for a given feasible sequence. The specificity of discrete-continuous problems is such that solving optimally the continuous part can make many solutions of the discrete part close in terms of quality, although they are different feasible sequences. Those differences in results will become more significant when heuristic procedures for allocating the continuous resources are applied as it was done for problems with makespan minimization in [13, 28, 29], and is planned for *NPV*-maximization in the future research.

8. Conclusions

In this paper discrete–continuous project scheduling problems with positive discounted cash flows and the maximization of the *NPV* have been considered. Three common payment models – Lump Sum Payment (LSP), Payments at Activity Completion times (PAC), and payments in Equal Time Intervals (ETI) have been analyzed. Formulations of mathematical programming problems for an optimal continuous resource allocation for each payment model have been presented. Applications of two local search metaheuristics – Tabu Search and Simulated Annealing have been proposed. The algorithms have been compared on a basis of some computational experiments.

The results show that, generally, SA performs better for smaller problems and smaller values of the discount rate, whereas TS is more efficient for bigger numbers of activities as well as for larger discount rates. Moreover, for periodic payments TS gets better when the number of payments grows, whereas SA prefers when payments are made more rarely, and becomes most effective for one payment at the end (LSP model).

The future research can be carried out in three directions. Firstly, further improvements of the proposed metaheuristics are certainly possible and/or implementing other (also hybrid) metaheuristic approaches. Secondly, generalizing the considered problem can be done in several ways, *e.g.* by incorporating negative cash flows which would make the problem more general on one hand, but also much more computationally complex on the other. Finally, heuristic procedures for allocating the continuous resource should be developed, in order to shorten the computational times, as well as to analyze larger problem instances.

Acknowledgements. This research was partially supported by the Polish National Science Centre.

References

[1] E.H.L. Aarts and J.H.M. Korst, *Simulated annealing and Boltzmann machines: A stochastic approach to combinatorial Optimization and Neural Computing.*, Wiley, Chichester (1989).

[2] L.A. Belady and C.J. Kuehner, Dynamic space sharing in computer systems. *Commun. ACM* **12** (1968) 282–288.

[3] P. Brucker, A. Drexl, R. Möhring, K. Neumann and E. Pesch, Resource-constrained project scheduling: notation, classification, models and methods. *Eur. J. Oper. Res.* **112** (1999) 3–41.

[4] E.L. Demeulemeester and W.S. Herroelen, *Project Scheduling – A Reseach Handbook*. Kluwer, Boston (2002).

[5] L.E. Drezet, (2008) RCPSP with financial costs, in C. Artigues, S. Demassey and E. Néron, *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*, ISTE-Wiley, London (2002) 213–226.

[6] S.E. Elmaghraby, Activity nets: a guided tour through some recent developments. *Eur. J. Oper. Res.* **82** (1995) 383–408.

[7] F. Glover and M. Laguna, *Tabu Search*. Kluwer, Norwell (1997).

[8] S. Hartmann and D. Briskorn, A survey of variants and extensions of the resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **207** (2010) 1–14.

[9] W.S. Herroelen, P. Van Dommelen and E.L. Demeulemeester, Project network models with discounted cash flows: a guided tour through recent developments. *Eur. J. Oper. Res.* **100** (1997) 97–121.

[10] W.S. Herroelen, B. De Reyck and E.L. Demeulemeester, Resource-constrained project scheduling: a survey of recent developments. *Comput. Oper. Res.* **25** (1998) 279–302.

[11] O. Icmeli, S.S. Erengüç and C.J. Zappe, Project scheduling problems: a survey. *Inter. J. Oper. Production Manag.* **13** (1993) 80–91.

[12] J. Józefowska, M. Mika, R. Różycki, Waligóra, G. and Węglarz, J. Simulated annealing for multi-mode resource-constrained project scheduling problem. *Annal. Oper. Res.* **102** (2001) 137–155.

[13] J. Józefowska, M. Mika, R. Różycki, G. Waligóra and J. Węglarz, A heuristic approach to allocating the continuous resource in discrete-continuous scheduling problems to minimize the makespan. *J. Schedul.* **5** (2002) 487–499.

[14] J. Józefowska, G. Waligóra and J. Węglarz, (2002) Tabu list management methods for a discrete-continuous scheduling problem, *Eur. J. Oper. Res.* **137** 288–302.

[15] J. Józefowska and J. Węglarz, (1998) On a methodology for discrete-continuous scheduling, *Eur. J. Oper. Res.* **107** 338–353.

[16] A. Kimms, *Mathematical Programming and Financial Objectives for Scheduling Projects*. Kluwer, Dordrecht (2012).

[17] R. Kolisch and S. Hartmann, (2000) Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **127** (2001) 394–407.

[18] R. Kolisch and S. Hartmann, Experimental investigation of heuristics for resource-constrained project scheduling: An update. *Eur. J. Oper. Res.* **174** (2006) 23–37.

[19] R. Kolisch and R. Padman, An integrated survey of deterministic project scheduling. *OMEGA Int. J. Manag. Sci.* **29** (2001) 249–272.

[20] C. Lawrence, J.L. Zhou and Tits A.L. Users guide for CFSQP Version 2.5, http://www.aemdesign.com/download-cfsqp/cfsqp-manual.pdf (1997) (Accessed 2nd April 2013).

[21] M. Mika, G. Waligóra and J. Węglarz, Simulated annealing and tabu search for multi-mode resource-constrained project scheduling with positive discounted cash flows and different payment models. *Eur. J. Oper. Res.* **164** (2005) 639–668.

[22] M. Mika, G. Waligóra and J. Węglarz, Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times. *Eur. J. Oper. Res.* **187** (2008) 1238–1250.

[23] L. Özdamar and G. Ulusoy, A survey on the resource-constrained project scheduling problem. *IIE Trans.* **27** (1995) 574–586.

[24] J. Skorin-Kapov, Tabu search applied to the quadratic assignment problem. *ORSA J. Comput.* **2** (1990) 33–45.

[25] G. Ulusoy, F. Sivrikaya-Şerifoğlu and Ş. Şahin, Four payment models for the multi-mode resource constrained project scheduling problem with discounted cash flows. *Annal. Oper. Res.* **102** (2001) 237–261.

[26] P.J.M. Van Laarhoven and E.H.L. Aarts. *Simulated Annealing: Theory Appl.*, Reidel, Dordrecht (1987).

[27] G. Waligóra, Discrete-continuous project scheduling with discounted cash flows – a tabu search approach. *Comput. Oper. Res.* **35** (2008) 2141–2153.

[28] G. Waligóra, Tabu search for discrete-continuous scheduling problems with heuristic continuous resource allocation. *Eur. J. Oper. Res.* **193** (2009) 849–856.

[29] G. Waligóra, Heuristic approaches to discrete-continuous project scheduling problems to minimize the makespan, *Comput. Optim. Appl.* **48** (2011) 399–421.

[30] G. Waligóra, (2011) Discrete-continuous project scheduling with discounted cash inflows and various payment models – a review of recent results. *Annal. Oper. Res.* `DOI: 10.1007/s10479-011-1014-0`.

[31] J. Węglarz Time-optimal control of resource allocation in a complex of operations framework. *IEEE Trans. Systems, Man and Cybernetics* **6** (1976) 783–788.

[32] J. Węglarz, Multiprocessor scheduling with memory allocation – a deterministic approach. *IEEE Trans. Comput.* **29** (1980) 703–709.

[33] J. Węglarz, J. Józefowska, M. Mika and Waligóra and G. Project scheduling with finite or infinite number of activity processing modes – a survey, *Eur. J. Oper. Res.* **208** (2011) 177–205.