Minimization of the Weighted Total Sparsity of Cosmonaut Training Courses

Alexander Lazarev[1,2,3], Nail Khusnullin[1], Elena Musatova[1(✉)], Denis Yadrentsev[4], Maxim Kharlamov[4], and Konstantin Ponomarev[4]

[1] V.A. Trapeznikov Institute of Control Science of Russian Academy of Sciences, Moscow, Russia
jobmath@mail.ru, nhusnullin@gmail.com, nekolyap@mail.ru
[2] Lomonosov Moscow State University, Moscow, Russia
[3] International Laboratory of Decision Choice and Analysis, National Research University Higher School of Economics, Moscow, Russia
[4] Yu.A. Gagarin Research & Test Cosmonaut Training Center, Star City, Russia
{d.yadrentsev,m.kharlamov,k.ponomarev}@gctc.ru

Abstract. The paper is devoted to a cosmonaut training planning problem, which is some kind of resource-constrained project scheduling problem (RCPSP) with a new goal function. Training of each cosmonaut is divided into special courses. To avoid too sparse courses, we introduce a special objective function—the weighted total sparsity of training courses. This non-regular objective function requires the development of new methods that differ from methods for solving the thoroughly studied RCPSP with the makespan criterion. New heuristic algorithms for solving this problem are proposed. Their efficiency is verified on real-life data. In a reasonable time, the algorithms let us find a solution that is better than the solution found with the help of the solver CPLEX CP Optimizer.

Keywords: Resource-constrained project scheduling problem
Heuristic algorithms · Planning · Priority rule

1 Introduction

Cosmonauts training is a long process of preparing cosmonauts for their space missions. This process includes physical training, medical tests, extra-vehicular activity training, procedure training, as well as training on experiments they will perform during their stay on the space station. For the training process in the Yu.A. Gagarin Research & Test Cosmonaut Training Center (GCTC), high-tech training facilities are used. These include integrated simulators for space crafts and space stations; a water tank for spacewalking training; centrifuges for simulating g-loads during launch and so on. Because of their high cost, the number

This work was supported by the Russian Science Foundation (grant 17-19-01665).

Y. Evtushenko et al. (Eds.): OPTIMA 2018, CCIS 974, pp. 202–215, 2019.
https://doi.org/10.1007/978-3-030-10934-9_15

of these facilities is limited. As several crews are trained at the same time, it is important to share the equipment among cosmonauts so that each of them has enough time to complete the training plan before launching. In this paper, we consider the final stage of cosmonaut training—training of approved crews for a specific space flight on a manned spacecraft. Simultaneously there are several crews training in the GCTC, but this article presents results for the major crew, i.e. the crew whose flight is planned for the nearest future. This crew has priority in the use of simulators and other equipment. Each member of the crew has his own training plan which should be performed with respect to resources and time constraints. Training of each cosmonaut is divided into thematic courses. Each training course is a set of special tasks. A cosmonaut can study several courses simultaneously. The study of the courses should take place in accordance with a curriculum which specifies precedence relations for courses. Resource constraints and precedence relations are supplemented by some constraints that take into account the specifics of training of cosmonauts. In other words, the cosmonauts training problem has the same constraints as the resource-constrained project scheduling problem (RCPSP) has, but it also has some additional restrictions. Note that RCPSP is NP-hard in the strong sense [1]. Different heuristic methods are proposed for solving RCPSP: priority-rule based scheduling methods truncated branch-and-bound, integer programming based heuristics, disjunctive arc concepts, local constraint-based analysis, sampling techniques, evolutionary algorithms and local search techniques (see, for example, [3,4,6,8,11]). In [8] a large number of modern heuristics that have been proposed for RCPSP solving are summarized and categorized. Note that most of these solving methods are designed for objective functions that are different from the one given in this article. As a rule, the objective function in such problems is the project duration (makespan). Another popular criteria are the maximum lateness, the total tardiness, the total flow time [3]. Most of them are regular, i.e. nondecreasing with respect to all finishing times of tasks. In our problem, we use unusual and nonregular objective function. To increase the efficiency of training, it is necessary to establish the correct duration of each course. Duration of a course is a length of a time period between the start of the first task of this course and the end of the last task of this course. If the duration of a course is either too large or too short, then the efficiency of training is reduced. Too fast study of a course is avoided by imposing constraints on the number of tasks of this course per week. To avoid too long duration of a course, we introduce a special objective function—the total weighted sparsity of training courses. It requires the development of new methods that differ from methods for solving the thoroughly studied RCPSP with the makespan criterion.

Since the problem has a high dimension, the use of exact methods does not allow to find an optimal solution in an acceptable time. Previously, for finding a feasible solution of this problem an approach based on methods of integer linear programming was proposed [10]. However, this approach turned out to be ineffective for high-dimensional problems. In [9] comparison of two approaches for finding a feasible solution to this problem for a medium dimension was presented.

The first approach was based on integer linear programming and the second one was on the basis of constraint programming (CP) [5]. A significant advantage of CP has been shown in that paper. Therefore in this paper, we use a mathematical model in terms of CP and run real-life test problems on the solver CPLEX CP Optimizer [12]. The article has the following structure. Sections 2 and 3 presents a description and mathematical formulation of the problem in terms of CP. Results of computational experiments based on CP Optimizer are given. In Sect. 4 we present a heuristic algorithm, its modifications, and comparison of different approaches on some test problems.

2 Problem Description

A crew consists of three cosmonauts. Each of them has his own individual training plan: a set of courses and a set of tasks in each course. Training schedule for cosmonauts of the crew is subject to the following constraints:

(c1) All tasks must be completed before launching. There exist special time boundaries for some tasks. These boundaries can be both precise and rather extended. In the first case a task has an exact date of its execution and in the second case—some period of time (for example, a winter forest landing training has to be in winter). We will call the tasks with time boundaries by fixed tasks.
(c2) There are precedence relations between some courses. A cosmonaut cannot start some course before finishing the previous course. There are also precedence relations between tasks into each course. For some tasks, there are strict precedence relations, i.e. strict time intervals between tasks. The peculiarity of this problem lies in the fact that fixed tasks are not included in the precedence relations graph.
(c3) At each moment of time the resources must be sufficient to execute all current tasks.
(c4) For some sets of tasks there are constraints on the total volume of tasks per day, or per week for one cosmonaut.
(c5) Each task has to be completed before the end of a working day. Some tasks can be executed only during a definite part of a day.
(c6) Some tasks must be performed by all crew members or by two of them simultaneously.

As it was mentioned in the Introduction, we describe an unusual objective function. Each course into a schedule has two characteristics: a volume and a duration (see Fig. 1). The volume of a course is a sum of durations of all tasks in this course. This value does not depend on a schedule and is known in advance. The duration of a course in some schedule is its volume plus all time intervals between tasks of this course. The volume of a course is a lower bound of its duration. Due to constraints (c4) this bound is not achievable. We define *sparsity* of a course as a ratio of its duration to its volume. Our goal is to minimize the total weighted sparsity which is a weighted sum of sparsities of all courses.

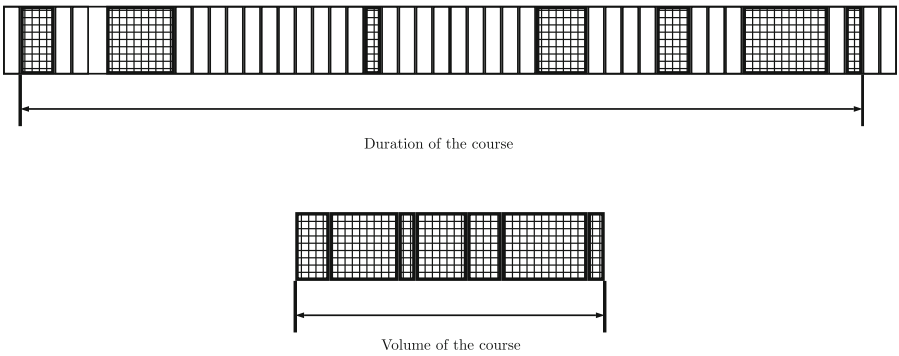

Fig. 1. Characteristics of a course.

To formalize the objective function, input the following notations. Let $I = \{1, 2, 3\}$ be the set of cosmonauts, and B_i be the set of courses of cosmonaut $i \in I$. Each course b has a weight (significance) w_b, which can take one of two values ω_1 or ω_2 ($\omega_1 < \omega_2$) established by an expert. Denote as $J_{i,b}$ the set of tasks of cosmonaut $i \in I$ from course $b \in B_i$. Each task $j \in J_{i,b}$ has duration p_j. $S_{i,b}^{first}(\pi)$ and $C_{i,b}^{last}(\pi)$ are the start time of the first task and the finish time of the last task of course $b \in B_i$ for cosmonaut $i \in I$ respectively in a schedule π. If for some course an order of tasks is not defined or several tasks can be the first (last) then we input dummy start and end tasks. Then the objective function has the following form:

$$F(\pi) = \sum_{i \in I} \sum_{b \in B_i} w_b \frac{C_{i,b}^{last}(\pi) - S_{i,b}^{first}(\pi)}{\sum\limits_{j \in J_{i,b}} p_j}. \tag{1}$$

It is worth noting that such objective function can arise in other areas of planning. For example, instead of an educational process, the following production planning can be considered. There is a set of complex jobs. Each job consists of a number of operations that require the use of different resources (machines). It is necessary to minimize the stay of each job in the production process. Such an objective function can be explained by additional storage costs or other expenses.

Since in the considered problem there is a constraint on makespan (all tasks must be completed before launching), the proof of NP-hardness can be obtained trivially by reduction from the classical RCPSP. The decision version of RCPSP can be transformed into the decision version of the considered problem with zero number of courses, the sparsity of which must be minimized.

3 Problem Statement in Terms of Constraint Programming

The first stage of our investigation of the problem is using an exact method. As previously we have demonstrated the advantage of constraint programming over

integer linear programming for the search of a feasible solution [9], now we try to use CP for minimization of (1). So, formalization of constraints (c1)–(c6) is made in terms of CP and with using of Optimization Programming Language (OPL) [13]. Integer linear programming formulation can be found in [9].

In CP formulation of an optimization problem constraints are not necessarily linear. In addition, the concept of global variables is widely used in CP. This concept allows us to reformulate the group of constraints into "global" one and treat the values of variables as a single entity on the whole planning horizon (or on its part). Another powerful tool of CP modeling is cumulative functions. They allow us to define the "behavior" of variables with help of piecewise linear functions. Due to the fact that IBM ILOG Optimizer and OPL were used for implementation of the model and for solving the problem, we use in the article names of built-in functions and the syntax of OPL.

We will assume that the planning of the cosmonaut training takes place at a certain given time interval, which specifies a planning horizon. We take as a unit of time a half-hour interval. Denote as W a set of all weeks and as D a set of all days on the planning horizon. Set H is a set of all half-hour intervals in a day. Possible moments of the beginning of a task are in a set $T = \{0, 1, \ldots, \mathcal{T}\}$, where $\mathcal{T}$ is the number of half-hour intervals in the planning horizon. Let J be a set of all tasks. For each task $j \in J$ input interval variable s_j with duration p_j:

$$interval \;\; s_j \in [t_1^j..t_2^j] \;\; size \;\; p_j, \tag{2}$$

where $[t_1^j..t_2^j]$ is a subset of T which is defined by some preliminary estimations or by constraints (c1).

We also need auxiliary integer variables $day_j \in \{1, 2, \ldots, |W| \cdot |D|\}$ and $week_j \in \{1, 2, \ldots, |W|\}$, $j \in J$, which are the numbers of the day and of the week respectively on the planning horizon for task j. The relations of variables in terms of CP can be written as follows:

$$day_j == element(days, startOf(s_j)),$$

$$week_j == element(weeks, startOf(s_j)),$$

where $days = (d_1, d_2, \ldots, d_{\mathcal{T}})$ and $weeks = (w_1, w_2, \ldots, w_{\mathcal{T}})$ are vectors whose components are associated with the numbers of days (weeks), corresponding to the respective time intervals from $T = \{0, 1, \ldots, \mathcal{T}\}$. Function $element(a, i)$ returns i-th element of vector a and function $startOf(s_j)$ is used to access the start time of interval variable s_j.

Let $G_i = (J_i, \Gamma_i)$ be a graph of precedence relations between tasks of cosmonaut $i \in I$. If $(j_1, j_2) \in \Gamma_i$, then the task j_1 has to be completed before the beginning of the task j_2. Then constraint (c2) can be written with help of built-in function $endBeforeStart$ as

$$endBeforeStart(s_{j_1}, s_{j_2}), \;\; \forall (j_1, j_2) \in \Gamma_i, \;\; \forall i \in I. \tag{3}$$

Constraints on resource usage over time can be modeled with constraints on cumulative function expressions. A cumulative function expression is a step

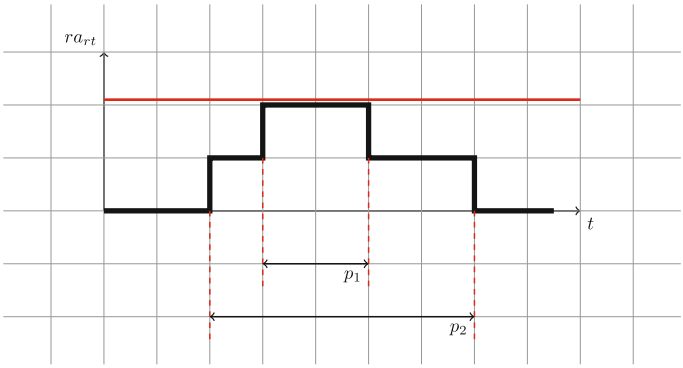

Fig. 2. Cumulative resource usage function.

function that can be incremented or decremented in relation to a fixed time or an interval. It is convenient to use cumulative function expression $pulse(a, b)$, which represents the contribution b to the cumulative function of an individual interval variable or fixed interval a. Figure 2 illustrates cumulative resource usage function for two intervals of length p_1 and p_2.

Let R be a set of renewable resources (instructors, simulators, classrooms, special equipment). Each cosmonaut is also a resource, available in a quantity of 1 during a working day. Denote by ra_{rt} the amount of resource $r \in R$ which is available at time $t \in T$, and by rc_{jr}—the amount of resource $r \in R$ required for the task $j \in J$. Let JR_r be the set of tasks which use resource $r \in R$. Using *step* function for constraint (c3) we get

$$\sum_{j \in JR_r} pulse(s_j, rc_{jr}) \leq ra_{rt}, \ \forall r \in R. \tag{4}$$

For realization of constraints (c4) we use built-in function $count(a, b)$. This function counts how many of the elements in the array given as a are equal to the value given as b. Then, if the number of tasks from some set J_i^A of cosmonaut i should be no more then a_i per week,

$$\sum_{j \in J_i^A} count(week_j, w) \leq a_i, \forall i \in I, \forall w \in W, \tag{5}$$

and, if number of tasks from some set J_i^C of cosmonaut i should be no more then c_i per day,

$$\sum_{j \in J_i^C} count(day_j, d) \leq c_i, \forall i \in I, \forall d \in D. \tag{6}$$

In order to take into account constraints of kind (c5), we use OPL expression $forbidStart(s_j, f)$, which constrains an interval variable s_j to not start where

the associated step function f has a zero value:

$$forbidStart(s_j, f),\ \forall j \in J. \tag{7}$$

Similar constraints can be written not only for the set of all tasks J, but also for arbitrary subsets of it.

The constraint *synchronize* makes interval variables start and end together:

$$synchronize(s_{j_1}, s_{j_2}, s_{j_3}),\ \forall (j_1, j_2, j_3) \in J^{123}. \tag{8}$$

or

$$synchronize(s_{j_1}, s_{j_2}),\ \forall (j_1, j_2) \in J^{12}, \tag{9}$$

where J^{123} and J^{12} are sets of tasks which cosmonauts have to execute together (constraints (c6)).

So, constraints (c1)–(c6) can be formulated with help of (2)–(9). Goal function (1) can be rewritten using functions $StartOf$ and $EndOf$. We used for the computational experiment 5 problems with real-life data provided by the GCTC. Characteristics of the problems (the number of courses, the total number of tasks and the planning horizon in weeks) are presented in Table 1.

Table 1. Characteristics of test problems.

N	Courses	Tasks	Planning horizon, weeks
1	10	202	5
2	18	383	15
3	37	838	40
4	48	1214	60
5	11	193	6

The graphs of precedence relations between all possible courses for test problems are presented on Fig. 3. Each cosmonaut has a set of courses which is a subset of all possible courses. Each graph G_i is obtained from the graph of precedence relations between courses and from graphs of precedence relations between tasks in the courses.

As we use the exact method, its runtime can be too much for a high-dimension problem. Therefore the runtime was restricted by some value. In Table 2 numerical results for solving problem (1)–(9) with help of ILOG CP Optimizer are presented.

We use the following notations: t is a time restriction (the solver stopped its work after this time), F is the found value of the objective function, *Solutions* is a number of solutions which the solver found out for time t, *Branches* is a number of branches, *Av.Sparsity* is an average sparsity of courses, which is defined as

$$\Phi(\pi) = \frac{1}{\sum\limits_{i \in I} |B_i|} \sum_{i \in I} \Big(\sum_{b \in B_i} \frac{C_{i,b}^{last}(\pi) - S_{i,b}^{first}(\pi)}{\sum\limits_{j \in J_{i,b}} p_j} \Big),$$

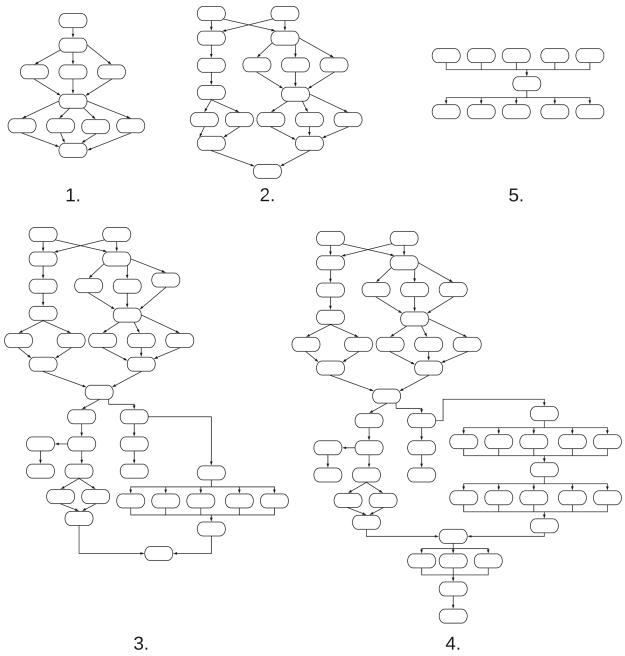

Fig. 3. Graphs of precedence relations between courses for test problems.

where $|B_i|$ is the number of courses of cosmonaut i. The average sparsity is a handy characteristic of a solution, which shows how many times the average length of the course is greater than its minimum possible length (volume). The calculations were performed on a workstation with an Intel Xeon processor E5-2673, 2.4 GHz, and 15 Gb of RAM, solver IBM ILOG CPLEX CP 12.6.2 was used. In these problems we have $\omega_1 = 1$ and $\omega_2 = 5$.

The solver could not find an optimal solution for 3 h. In the problem of the highest dimension (problem 4) the value of the objective function is not improved after 30 min. It is worth noting that problem 5, despite its low dimension, has also proved to be very difficult: the average sparsity is very high and the value of the objective function almost is not improved. This can be explained by the structure of the precedence relations graph (see Fig. 3).

In the next section heuristics algorithms will be presented for solving the problem.

4 Heuristic Algorithms

Despite a wide variety of heuristic methods for solving RCPSP, the priority rule-based scheduling is still one of the most important solution techniques. This can be explained by the simplicity of its implementation, its speed, and efficiency. A priority rule-based scheduling heuristic is made up of two parts, a schedule

Table 2. Testing the CP model.

N	*t*	*F*	*Solutions*	*Branches*	*Av.Sparsity*
1	1 min	58.89	1	21 205	2.05
	5 min	57.51	5	78 030	2.03
	15 min	54.36	7	202 587	2.03
	30 min	53.99	10	389 910	2.01
	1 h	53.27	13	465 358	1.99
	3 h	51.27	17	533 425	1.87
2	1 min	106.98	2	36 579	2.41
	5 min	106.92	4	95 403	2.41
	15 min	97.97	8	155 712	2.22
	30 min	97.97	9	258 495	2.22
	1 h	96.87	13	507 082	2.03
	3 h	96.71	14	653 121	2.01
3	1 min	488.86	2	14 785	4.88
	5 min	487.86	4	45 673	4.76
	15 min	487.74	5	130 437	4.75
	30 min	487.21	7	243 897	4.7
	1 h	482.26	13	612 342	4.34
	3 h	302.21	23	756 934	3.09
4	1 min	692.95	1	13 635	5.07
	5 min	691.29	3	43 801	5.06
	15 min	690.60	7	66 075	5.01
	30 min	690.07	8	93 085	5.01
	1 h	690.07	8	142 191	5.01
	3 h	690.07	8	304 989	5.01
5	1 min	197.51	3	25 011	4.99
	5 min	197.25	16	105 672	4.98
	15 min	196.41	24	238 235	4.98
	30 min	195.87	25	439 793	4.93
	1 h	195.83	32	828 890	4.91
	3 h	195.83	39	1 906 453	4.91

generation scheme and a priority rule. Two best known and the oldest schedule generation and the parallel scheduling schemes (see [7]). These schemes generate a feasible schedule by extending a partial schedule step by step. In each step, a generation scheme defines the decision set, i.e. set of all schedulable tasks. Then a used to choose one (or more in parallel case) task from the decision set is scheduled.

In this article, some kind of scheduling heuristics with a serial generation scheme is presented. Firstly we define a priority rule which takes into account objective function (1). Then a schedule generation scheme is presented.

4.1 The Priority Rule

Each task $j \in J_{i,b}$ has a weight $w_b \in \{\omega_1, \omega_2\}$ which is defined by the weight of the course $b \in B_i$. Let $A = \{1, 2, 3\}$ be a set of possible priorities of a task. Assume that we already have a partial schedule, i.e. a schedule where only a task has been assigned a finish time. This subset of tasks will be called a *scheduled* set. We say that a task $j \in J_{i,b}$ from a decision set has priority $\alpha_j = 1$, if there are no any tasks from course b in the scheduled set. In other words, $\alpha_j = 1$, if j is the first task of a course. Otherwise task j has priority $\alpha_j = 2$ in case of $w_b = \omega_1$ and $\alpha_j = 3$ in case of $w_b = \omega_2$. Thus, the main priority is given to tasks from the already started courses with the high weight. So, for each task $j \in J_{i,b}$, $b \in B_i, i \in I$, we have

$$\alpha_j = \begin{cases} 1, \text{ if } j \text{ is the first task of course } b, \\ 2, \text{ if } j \text{ is not the first task of course } b \text{ and } w_b = \omega_1, \\ 3, \text{ otherwise.} \end{cases} \tag{10}$$

4.2 Schedule Generation Scheme

To present the proposed algorithm, define the following functions:

- $GetTask(w, d, i, start)$ returns a schedulable task of the maximal possible priority and its start moment for cosmonaut i. The depth-first search is used. This function checks whether it is possible to schedule a task in the day d of the week w at moment $start$ within the constraints (c1)–(c5). If there are no any schedulable tasks for this moment, the function returns the next possible time moment of this day or the empty set (if there are no any schedulable tasks in this day). As constraint (c6) is connected with several cosmonauts, this function does not check, whether the respective synchronous task of another cosmonaut can be scheduled at the same time.
- $task.IsShared$ returns $TRUE$, if the $task$ must be performed simultaneously with some task of another cosmonaut.
- $Release(start, task)$ adds the $task$ to a partial schedule at time moment $start$.
- $DealContract(task.SharedTasks, start, w, d)$ is a function, which defines the nearest possible common moment in day d of week w (starting from moment $start$) for the $task$ and for all its synchronous tasks from the set $task.SharedTasks$.

Let enumerate cosmonauts of the crew in the decreasing order of their training loads. We take the first unplanned time moment from the planning horizon and try to schedule for cosmonaut 1 a task of the highest possible priority. Then we turn to schedule for cosmonaut 2 and so on. For details see Algorithm 1.

Algorithm 1.

```
for w in W, d in D do
    repeat
        for i in I do
            (start, task) = GetTask(w, d, i, start)
            if task == null then
                continue
            end if
            if not task.IsShared then
                Release(start, task)
                continue
            end if
            DealContract(task.SharedTasks, start, w, d)
        end for
    until not isAnyonePlanned()
end for
```

We used two variants of function *DealContract* for Algorithm 1. In the first variant, this function only checks whether it is possible to establish all tasks from *task.SharedTasks* at a given time moment *start*. If it is possible, it adds all shared tasks to a partial schedule. In the second variant, the function tries to find out the possible time moment if the shared tasks cannot be performed at a time moment *start* (see Algorithm 2). In both variants of *DealContract* and in *GetTask* a function $TryToDeal(task, w, d, start)$ is used, which check whether it is possible to schedule the *task* in day d of week w at time moment *start* within the constraints (c1)–(c5). So, this function is the main brick of the algorithm and in the future we will measure the complexity of the algorithm as the number of calls to this function.

4.3 Computational Results

At the first stage of the computational experiment, we solved 5 practical previously declared problems. Results of using Algorithm 1 with two variants of function *DealContract* described above are presented in Table 3. The following notations were used: F is the value of the objective function, $Av.Sparsity$ is the average sparsity of courses, $TryToDeal\ count$ is the number of calls to the function $TryToDeal$.

It is easy to see that for small problems 1 and 2 the heuristic algorithm with both variants of the function *DealContract* received identical results. The problem of the highest dimension is the only problem in which the second variant of the algorithm turned out to be better than the first one. For all problems using the heuristic algorithms significantly improved the value of the objective function in comparison with the value obtained with the help of the CP solver (see Table 2).

At the second stage of the experiment, we generated a series of 500 different problems for each graph of precedence relations between courses from Fig. 3.

Algorithm 2. *DealContract*$(task.SharedTasks, start, w, d)$

```
for task in SharedTasks do
    if not task.AllParents.IsSchedulled then
        return
    end if
end for
repeat
    for task in SharedTasks do
        (isDealed, start) = TryToDeal(task, w, d, start)
        if not isDealed then
            return
        end if
        task.Start = start
    end for
until SharedTasks.AllStartsAreEqual()
for task in SharedTasks do
    Release(start, task)
end for
```

Table 3. Testing the heuristic algorithms on real-life problems.

N	The first variant			The second variant		
	F	$Av.Sparsity$	$TryToDeal\ count$	F	$Av.Sparsity$	$TryToDeal\ count$
1	36.204	1.248	466	36.204	1.248	466
2	79.225	1.650	1048	79.225	1.650	1048
3	251.394	2.646	2353	254.020	2.673	2241
4	346.805	2.774	3629	270.258	2.162	3766
5	77.475	2.671	534	83.612	2.883	567

Duration p_j for each task of a problem was chosen as an arbitrary integer from the interval $[1; 8]$. Results of this experiment are presented in Table 4. Here N is a number of graph from Fig. 3, which is used for generation of a series of problems; $F_1 < F_2$ ($F_1 > F_2$) is a number of problems from a series, in which the function value is strictly better for variant 1 (variant 2) of the algorithm; $T_1 < T_2$ ($T_1 > T_2$) is a number of problems, in which the makespan is strictly better for variant 1 (variant 2) of the algorithm; Avg, Max and Min are the average, the minimal and the maximal sparsities respectively for a series of problems.

The experiment shows that for the first two series of problems two approaches give the same results. For the problems of high dimensions (series 3 and 4) almost in half of the cases, the best results were given by one approach, in half - by another. The second variant of the function *DealContract* has a slight advantage in the number of problems with the best value of the objective function for large-scale problems. So, both variants can be used for such problems together. The runtime of algorithms did not exceed 2 min. If to say about the makespan, the

Table 4. Testing the heuristic algorithms on series of test problems.

N	$F_1 < F_2$	$F_1 > F_2$	$T_1 < T_2$	$T_1 > T_2$	The first variant			The second variant		
					Avg	Max	Min	Avg	Max	Min
1	3	1	4	0	1.51	1.87	1.30	1.51	1.87	1.30
2	0	3	0	3	2.07	3.20	1.67	2.07	3.20	1.67
3	230	264	227	267	2.72	3.88	1.97	2.72	4.53	1.97
4	245	253	272	226	2.74	4.45	1.96	2.74	5.06	1.92
5	350	149	384	115	2.92	4.38	2.18	2.92	4.65	2.02

trend is the same, but in some cases, the better value of the objective function is achieved due to the worse value of the makespan. This is clearly seen from Table 5, which contains summary data for all series of problems. This tendency is a distinctive feature of this problem. Indeed, if we did not have a constraint on the planning horizon and other constraints, the optimal value of the objective function would be achieved in the schedule where courses pass one after another without intersections. Therefore, it is necessary to maintain a balance between the value of the objective function and the makespan.

Table 5. The relations between the values of the objective function and the planning horizons in the series of problems.

	$T_1 < T_2$	$T_1 > T_2$	$T_1 = T_2$
$F_1 < F_2$	761	67	0
$F_1 > F_2$	126	544	0
$F_1 = F_2$	0	0	1002

5 Conclusion

A new real-life problem statement for cosmonaut training planning with an unusual objective function (the weighted total sparsity) is presented. A heuristic algorithm with two variants of realization is proposed for solving this problem. Comparison of the algorithm with an exact method is carried out. Numerical testing of algorithms on real data, provided by the GCTC, demonstrated the efficiency of the approach. For all problems, the heuristic algorithm found schedules in which the value of the objective function is significantly better than the value obtained with a help of the CPLEX CP Optimizer in 3 h. Comparison of the two variants of the heuristic algorithm showed that it is sensible to apply both variants to high-dimensional problems since they give different solutions. In this case, the best solution can be obtained by both the first and the second variant.

The directions of further research may include obtaining more flexible priority rules, lower bounds for the given objective function and developing approximate algorithms.

References

1. Artigues, C., Demassey, S., Neron, E. (eds.): Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications. Wiley-ISTE, Hoboken-London (2008)
2. Bartusch, M., Mohring, R.H., Radermache, F.J.: Scheduling project networks with resource constraints and time windows. Ann. Oper. Res. **16**, 201–240 (1988)
3. Brucker, P., Drexl, A., Mohring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: notation, classification, models, and methods. Eur. J. Oper. Res. **112**, 3–41 (1999)
4. Debels, D., Vanhoucke, M.: A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. Oper. Res. **55**(3), 457–469 (2007)
5. Dechter, R.: Constraint Processing. Morgan Kaufmann Publishers, San Francisco (2003)
6. Homberger, J.: A multi-agent system for the decentralized resource-constrained multi-project scheduling problem. Int. Trans. Oper. Res. **14**, 565–589 (2007)
7. Kolisch, R.: Serial and project scheduling methods revisited: theory and computation. Eur. J. Oper. Res. **90**, 320–333 (1996)
8. Kolisch, R., Hartmann, S.: Experimental investigation of heuristics for resource-constrained project scheduling: an update. Eur. J. Oper. Res. **174**(1), 23–37 (2006)
9. Lazarev, A.A., et al.: Mathematical modeling of the astronaut training scheduling. UBS **63**, 129–154 (2016)
10. Musatova, E., Lazarev, A., Ponomarev, K., Yadrentsev, D., Bronnikov, S., Khusnullin, N.: A mathematical model for the astronaut training scheduling problem. IFAC PapersOnLine **49**(12), 221–225 (2016)
11. Valls, V., Quintanilla, M.S., Ballestin, F.: Resource-constrained project scheduling: a critical activity reordering heuristic. Eur. J. Oper. Res. **149**(2), 282–301 (2003)
12. IBM Homepage. https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-cp-optimizer. Accessed 1 July 2018
13. IBM Homepage. https://www.ibm.com/analytics/data-science/prescriptive-analytics/optimization-modeling. Accessed 1 July 2018