ISSN 0005-1179 (print), ISSN 1608-3032 (online), Automation and Remote Control, 2023, Vol. 84, No. 9, pp. 1128–1139.

Minimizing the Total Weighted Duration of Courses in a Single Machine Problem with Precedence Constraints

E. G. Musatova[*,a] **and A. A Lazarev**[*,b]

[*] *Trapeznikov Institute of Control Sciences, Russian Academy of Sciences, Moscow, Russia*
e-mail: [a] *nekolyap@mail.ru,* [b] *jobmath@mail.ru*

Received February 8, 2023
Revised June 20, 2023
Accepted July 20, 2023

Abstract—A single machine scheduling problem with a given partial order of jobs is considered. There are subsets of jobs called courses. It is necessary to schedule jobs in such a way that the total weighted duration of all courses is minimal. We consider the case when the initial job and the final one of each course are uniquely determined. The NP-hardness of the problem under consideration is proved. We propose an algorithm for solving the problem, the complexity of which depends polynomially on the total number of jobs, but exponentially on the number of courses, which makes it possible to use it efficiently with a fixed small number of courses and an arbitrary number of jobs.

Keywords: scheduling theory, single machine problem, NP-hard problems, downtime of resources minimization

DOI: 10.25728/arcRAS.2023.90.12.001

1. INTRODUCTION

We consider a set of jobs that need to be executed on one machine and a precedence graph that sets a partial order of jobs. Some of the jobs are combined into subsets, which we will call courses. It is necessary to build a schedule in which the total weighted duration of all courses is minimal. The duration of a course is the length of the time interval between the start of processing the initial job from this course and the end of processing its final one. The article considers the case when the initial and the final jobs of each course are uniquely determined.

Single machine problems are comprehensively investigated in scheduling theory [1, 2]. At the same time, the single machine problem of minimizing the weighted total duration of courses has not been considered before.

The need to minimize the total duration of courses arises in different areas of production, education and services. In [3], a resource-constrained project scheduling problem with such objective function is considered in relation to constructing a schedule for preparing cosmonauts to work at the International Space Station. It is necessary to minimize the length of each course (or an on-board system in the terminology of the Gagarin Cosmonaut Training Centre). If too much time passes from the beginning of a course to the exam, the cosmonauts' skills are considered lost and they have to add additional hours to the preparatory process, which leads to large time and financial losses. In that publication, a heuristic algorithm for solving the problem is proposed.

We can also interpret such a problem as a problem of minimizing the total downtime of resources. Suppose that all jobs of each course requires their own specific resource (for example, processing on additional equipment). This resource is taken on a temporary lease, which begins to be paid

simultaneously with the start of the first job from the course and ends with the completion of the last job from this course. Then the duration of a course can be associated with the total payment for the resource, and the objective function characterizes the total payments for all leased resources. In addition to payments for additional resources, this goal function can be considered as a fee for storage and rental of premises.

For the first time, the question of the duration of courses was considered in [4]. Here, along with the usual concept of "activity", the concept of "hammock activity" is introduced. The duration of a hammock activity is determined by the beginning and the end of some fixed activities. This article discusses project scheduling without resource constraints and provides methods for calculating the duration of hammock activities. Note that in the case when the first and the last jobs of a course are uniquely defined, the concepts of hammock activity and course coincide.

The concept of hammock activity was developed in [5]. The authors of that article consider the problem of minimizing the total cost of several hammock activities in a project both in the presence of resource constraints (Resource-Constrained Hammock Cost Problem, RCHCP) and without them. The cost of a hammock activity means its weighted duration. In the absence of resource constraints, the problem is reduced to a linear programming problem. In case of resource constraints, the formulation of the problem in the form of a mixed integer linear programming problem is proposed. In the dissertation [6] the research of the RCHCP is continued. The author suggests metaheuristics for solving this problem, and also provides an extensive review of publications on problems of the RCHCP type.

Some studies use terms other than "hammock activity" to describe a similar objective function. So in theory of scheduling repetitive jobs (see, for example, [7]), such problems are known as project scheduling problems with work continuity constraints. For example, in [8] the following project scheduling problem with repetitive jobs is considered. There is some basic precedence graph, which is duplicated k times. Some of the repetitive jobs requires additional resources (equipment, teams of workers, etc.), and it is necessary to complete the project by the specified directive deadline with minimizing the duration of these repetitive jobs. Examples of practical applications are given for construction of multi-storey buildings, where identical jobs are performed on each storey, construction of bridges, roads, etc. In [9] the terms used are "minimizing crew idle time" and "minimizing resource idle time". The authors describe a practical use of algorithms developed for such a problem during the construction of the Westerschelde Tunnel in the Netherlands. Crews of workers and freezing machines were selected as resources whose total duration of use had to be minimized.

Thus, if we talk about minimizing the total duration of courses, the major attention in the literature is paid to project scheduling problems either with or without resource constraints. In the first case, we have to deal with an NP-hard problem [5] and the emphasis in such studies is on the development of heuristic algorithms, whereas in the second case polynomial algorithms are built.

Our article discusses a single machine problem that can be interpreted as a project scheduling problem with single resource available in the amount of one unit at any given time, provided that each job also requires one unit of the resource. It is shown that this problem is NP-hard. We propose an algorithm which allows to find an exact solution in the case of a large number of jobs, but a small fixed number of courses. Section 2 provides a formulation of the problem. Section 3 proves NP-hardness of the problem under consideration, as well as some of its properties. Section 4 is devoted to solving an auxiliary problem, and in section 5 an algorithm for solving the original problem based on solving an auxiliary problem is described and the results of a numerical experiment are presented.

2. PROBLEM STATEMENT

There is a set of jobs $I = \{1, \ldots, n\}$ that need to be executed on one machine. For each job $i \in I$, its processing time is equal to $p_i > 0$. All jobs are available at zero time. The processing of any job cannot be interrupted.

A directed acyclic precedence graph $G(I, E)$ is given, where I is a set of vertices, and E is a set of arcs. We say that for a pair of jobs $i, j \in I$, job i precedes job j, denoting by $i \to j$, if there exists a directed path from vertex i to vertex j in the graph $G(I, E)$. Denote by $A(i)$ the set of all jobs preceding job i and by $D(i)$ the set of jobs preceded by job i. Each job i must be executed after all jobs from set $A(i)$ and before all jobs from $D(i)$.

In addition, there are sets $I_k \subset I$, $|I_k| > 1$, $k \in \{1, \ldots, K\}$, called *courses*. Figure 1 gives an example of a precedence graph for a problem with three courses. Each course I_k, $k \in \{1, \ldots, K\}$, has its own weight $w_k > 0$. Depending on the interpretation, the weight is either a price of a leased resource per unit of time or an index of importance (significance) of the course. For each job $i \in I$, a schedule π determines its processing sequence number on the machine, which we will denote by $\pi(i)$, a start time of processing $S_i(\pi)$ and a completion time of processing $C_i(\pi) = S_i(\pi) + p_i$. We will call a schedule feasible if it does not contradict the precedence relations of jobs and the machine does not serve more than one job at any given time. The problem of minimizing the total weighted duration of all courses implies minimizing the following objective function:

$$H(\pi) = \sum_{k=1}^{K} w_k \left(\max_{i \in I_k} C_i(\pi) - \min_{i \in I_k} S_i(\pi) \right). \tag{1}$$

The article considers the case when the first and the last jobs of each course are uniquely determined, i.e. the following assumption is true.

Assumption 1. For each course I_k, $k \in \{1, \ldots, K\}$, there are jobs i_k^a and i_k^d, such that $i_k^a \in A(j)$ for any $j \in I_k \setminus \{i_k^a\}$ and $i_k^d \in D(j)$ for any $j \in I_k \setminus \{i_k^d\}$.

This condition is often fulfilled in practice. For example, in an educational process, the first lesson is usually introductory, and the last one implies a general knowledge test, while the sequence of

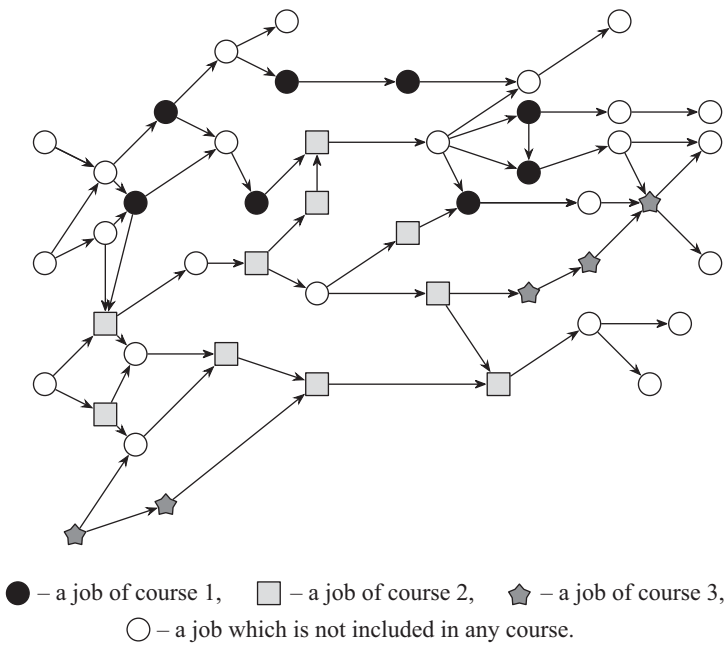

Fig. 1. A precedence graph and courses.

other classes in the course may vary. In this case, the objective function can be written as

$$H(\pi) = \sum_{k=1}^{K} w_k \Big(C_{i_k^d}(\pi) - S_{i_k^a}(\pi) \Big) = \sum_{k=1}^{K} w_k \Big(C_{i_k^d}(\pi) - C_{i_k^a}(\pi) + p_{i_k^a} \Big). \tag{2}$$

We will call i_k^a and i_k^d *extreme* jobs (vertices) of course k, $k \in \{1, \ldots, K\}$. The set of all extreme jobs of courses is denoted by I_{ad}, and their number—by e. Since some jobs from I_{ad} can be extreme in several courses, $e \leqslant 2K$.

The minimization of function (2) exactly coincides with the minimization of the hammock activities cost described in the introduction. In the standard scheduling theory notation [10] this problem can be classified as $1|prec|H$, where 1 means one machine, *prec*—the presence of precedence constraints, and H—the objective function (2).

3. PROBLEM PROPERTIES

Remark 1. Since all jobs in problem $1|prec|H$ are available at the same time and downtime does not improve the value of the objective function, we can only consider schedules without breaks between jobs, with the start of the first job at zero time. Indeed, let there exist an optimal schedule π_1, in which there are machine downtimes or the first job does not start at zero time. Then we can consider the schedule π_2, in which all jobs are performed in the same order as in π_1, but the first job starts from zero time and there are no breaks between jobs. Schedule π_2 is also optimal.

Let's show that even with the same processing time of all jobs, the problem under consideration is NP-hard.

Theorem 1. *Problem* $1|prec, p_i = 1|H$ *is strongly NP-hard.*

Proof. Let's consider the classical single machine problem of minimizing the weighted total flow time $1|prec, p_i = 1| \sum w_i C_i$. The problem is formulated as follows. One machine and a set of jobs $I' = \{1, \ldots, n'\}$ are given, each job i has weight w_i' and processing time $p_i' = 1$, $i \in \{1, \ldots, n'\}$. Let there also be given a directed precedence graph $G'(I', E')$. It is necessary to find a schedule π' that minimizes the objective function $\sum_{i=1}^{n'} w_c' C_i'(\pi')$, where $C_i'(\pi')$ is the completion time of the ith job in the schedule π'.

This problem is strongly NP-hard [11]. Let's reduce it to the following problem $1|prec, p_i = 1|H$. There is a set of jobs $I = \{1, \ldots, n\}$, $n = 2n'$. Each job i has processing time $p_i = 1$. Graph $G(I, E)$ has the following structure. There are $|E'|$ arcs defined by the following rule: if in graph $G'(I', E')$ there is an arc (j, k), then in graph $G(I, E)$ there is also arc (j, k). In addition, there are $n' - 1$ arcs of the form $(i, i+1)$ for $i \in \{n'+1, \ldots, 2n'-1\}$ and $|L|$ arcs of the form $(2n', l)$, where L is the set of root vertices (sources) in graph G', $l \in L$. The structure of graph $G(I, E)$ is shown in Fig. 2.

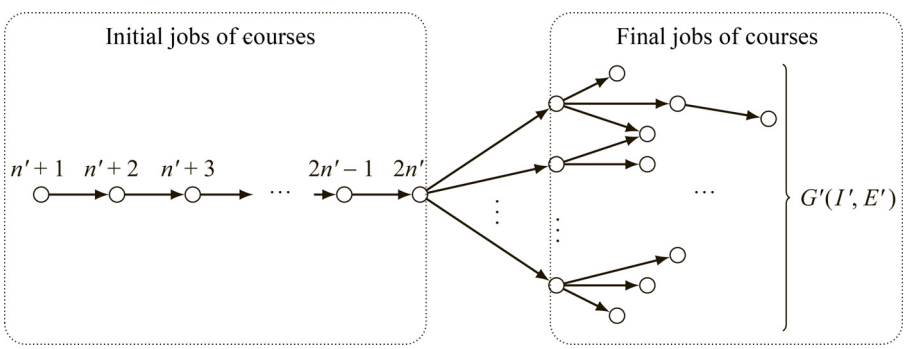

Fig. 2. The structure of graph $G(I, E)$ from the proof of theorem 1.

Let's set the courses as follows: jobs $n'+1$ and 1 belong to the first course, which has weight w'_1, jobs $n'+2$ and 2 belong to the second course, which has weight $w'_2, \ldots,$ jobs $2n'$ and n' refer to the n'th course, which has weight $w'_{n'}$, i.e. each course consists of two jobs, the first of which lies in set $\{n'+1, \ldots, 2n'\}$, and the final—in set I'. Let π be an arbitrary optimal schedule for this problem. The value of the objective function is

$$H(\pi) = \sum_{i=1}^{n'} w'_i \left(C_i(\pi) - C_{i+n'}(\pi) + 1\right) = \sum_{i=1}^{n'} w'_i C_i(\pi) - \sum_{i=1}^{n'} w'_i C_{i+n'}(\pi) + \sum_{i=1}^{n'} w'_i. \tag{3}$$

Due to the structure of the precedence graph, job $n'+1$ will be processed first. Then taking into account Remark 1 we have $C_{n'+1}(\pi) = 1$. Since there are arcs $(i, i+1)$ in the graph G for $i \in \{n'+1, \ldots, 2n'-1\}$, and all jobs from $\{1, \ldots, n'\}$ are processed after job $2n'$, the order of jobs $n'+2, \ldots, 2n'$ is known, moreover,

$$C_i(\pi) = i - n', \quad i \in \{n'+2, \ldots, 2n'\}, \tag{4}$$

that is, all these jobs are processed one after another without breaks. Thus, in schedule π, the execution of jobs $n'+1, \ldots, 2n'$ is predetermined and will end at time n'. But then, taking into account (3), schedule π is optimal if and only if (4) is executed and the minimum of the function $\sum_{i=1}^{n'} w'_i C_i(\pi)$ is reached with schedule π, i.e. when in the schedule π jobs $1, \ldots, n'$ are executed starting from the moment n' in a way minimizing their weighted total flow time. As a result, the optimal schedule π' in the problem $1|pres, p_i = 1|\sum w_i C_i$ can be obtained from the optimal schedule π of the described problem $1|prec, p_i = 1|H$. Completion times of the jobs in the problem $1|prec, p_i = 1|\sum w_i C_i$ are as follows:

$$C'_i(\pi') = C_i(\pi) - n', \quad i \in \{1, \ldots, n'\}.$$

The theorem is proven.

Remark 2. Similarly, it is possible to prove NP-hardness in the strong sense of the one-machine problem of minimizing the total (unweighted) duration of courses with different jobs processing times, using a reduction from the NP-hard problem $1|prec|\sum C_i$ to it.

As noted earlier, due to Remark 1 next, we will consider only schedules without breaks between jobs, with the start of the first job at time zero. In this case, for each job $j \in I$ its sequence number $\pi(j)$ in the schedule π uniquely sets the start time and the end of the job. Note also that only completion times of extreme jobs of courses are included in the definition of the objective function (2), moreover, only differences, and not absolute values, are decisive. Since in the absence of breaks the duration of a course is determined by the jobs started after the first job of the course and completed before the completion of the last job of the course, let's rewrite objective function (2) in a different form, without using the job completion times :

$$H(\pi) = \sum_{k=1}^{K} w_k \left(\sum_{j:\pi(i_k^a) \leqslant \pi(j) \leqslant \pi(i_k^d)} p_j \right). \tag{5}$$

Then we can write

$$H(\pi) = \sum_{j=1}^{n} W_j(\pi) p_j, \tag{6}$$

where

$$W_j(\pi) = \sum_{\substack{k \in K: \\ \pi(j) \geqslant \pi(i_k^a)}} w_k - \sum_{\substack{k \in K: \\ \pi(j) > \pi(i_k^d)}} w_k. \tag{7}$$

So the contribution $W_j(\pi)$ of each job $j \in I$ to the objective function depends on the mutual order of extreme jobs of courses and its place among the extreme jobs. In this regard, the following idea of solving the problem arises: for each feasible permutation of extreme jobs of courses, it is necessary to find an optimal order of jobs relative to the extreme jobs. The next section will present a polynomial algorithm for constructing an optimal schedule for a given order of extreme jobs of courses.

4. SOLVING AN AUXILIARY PROBLEM

As it was shown in the previous section, the value of the objective function of the initial problem depends on the mutual order of extreme jobs of courses. Denote by $\Lambda = (\lambda_1, \ldots, \lambda_e)$ an arbitrary permutation of extreme jobs that does not contradict the precedence relations given by graph $G(I, E)$, and introduce a directed acyclic graph $G'(I, E')$ such that $E \subset E'$ and for the set of extreme jobs, the following condition is fulfilled:

$$\lambda_1 \to \lambda_2 \to \ldots \lambda_{e-1} \to \lambda_e. \tag{8}$$

The graph $G'(I, E')$ is obtained from $G(I, E)$ by sequentially connecting extreme vertices $\lambda_1, \ldots, \lambda_e$ by arcs in accordance with the order given by Λ. If such order does not contradict the precedence constraints of the problem, the resulting graph $G'(I, E')$ will be acyclic. Due to the acyclicity of the original graph $G(I, E)$, there is always at least one sequence of extreme jobs Λ that does not contradict the precedence constraints. Then in any schedule π for the graph $G'(I, E')$ we will have

$$\pi(\lambda_1) < \pi(\lambda_2) < \ldots < \pi(\lambda_{e-1}) < \pi(\lambda_e).$$

The problem is to minimize function (6)–(7) relative to the new graph $G'(I, E')$. We will denote the auxiliary problem by P_Λ.

For all jobs that are not extreme, it is necessary to determine their places in the sequence $\lambda_1, \lambda_2, \ldots, \lambda_e$. For each job, there are no more than $e + 1$ options (the job is processed before λ_1, between λ_1 and λ_2, etc.). Say that job j is placed in *cell* q, $q \in \{1, \ldots, e-1\}$ if it is executed after extreme job λ_q and before extreme job λ_{q+1}. Assume that $q = 0$ if job j is executed before λ_1, and $q = e$ if j is executed after λ_e.

Consider for each extreme job $\lambda_i \in I_{ad}$ the sets $A(\lambda_i)$ and $D(\lambda_i)$ in the graph $G'(I, E')$. Note several obvious statements that will be used later and the proof of which follows directly from (8).

Lemma 1. *a) If $j \in A(\lambda_i)$, then $j \in A(\lambda_k)$ for all $k \geqslant i$.*

b) If $j \in D(\lambda_i)$, then $j \in D(\lambda_k)$ for all $k \leqslant i$.

c) If $j \notin D(\lambda_i)$, then $j \notin D(\lambda_k)$ for all $k \geqslant i$.

d) If $j \notin A(\lambda_i)$, then $j \notin A(\lambda_k)$ for all $k \leqslant i$.

To determine boundaries of the possible location of the non-extreme jobs by cells in the row of extreme jobs, we introduce the following notation:

$$q_1(j) = \begin{cases} 0, & \text{if } j \notin D(\lambda_1), \\ \max\{g \in \{1, \ldots, e\} \mid j \in D(\lambda_g)\}, & \text{otherwise;} \end{cases}$$

$$q_2(j) = \begin{cases} e, & \text{if } j \notin A(\lambda_e), \\ \min\{g \in \{1, \ldots, e\} \mid j \in A(\lambda_g)\} - 1, & \text{otherwise.} \end{cases}$$

Lemma 2. *For each job $j \in I \setminus I_{ad}$ the inequality $q_1(j) \leqslant q_2(j)$ is satisfied.*

Proof. If either $j \notin D(\lambda_1)$ or $j \notin A(\lambda_e)$, the statement is obvious. For a proof in the other cases, we assume the opposite. Let $q_1(j) > q_2(j)$. By definition of $q_1(j)$ we have $j \in D(\lambda_{q_1(j)})$. On the other hand, by definition of $q_2(j)$ we have $j \in A(\lambda_{q_2(j)+1})$. But then by lemma 1 we get $j \in A(\lambda_k)$ for all $k \geqslant q_2(j) + 1$, which means $j \in A(\lambda_{q_1(j)})$. The resulting contradiction proves the lemma.

Lemma 3. *If in a feasible solution of problem P_λ job j is placed into cell q, then $q_1(j) \leqslant q \leqslant q_2(j)$.*

Proof. Assume the opposite. Let's suppose that in some feasible schedule job j is placed into cell q, for which either $q < q_1(j)$ or $q > q_2(j)$ is holds. Let $q < q_1(j)$. Then $q_1(j) > 0$ and by definition $j \in D(\lambda_{q_1(j)})$, which means that job j cannot be executed before $\lambda_{q_1(j)}$, which contradicts the choice of cell q. Let $q > q_2(j)$. Then $q_2(j) < e$ and by definition $j \in A(\lambda_{q_2(j)+1})$, which means that the job cannot be executed after $\lambda_{q_2(j)+1}$, which contradicts the choice of cell q. The lemma is proven.

For each cell $q \in \{0, \ldots, e\}$ let's introduce its price $f(q)$ according to the following rule:

$$f(q) = \sum_{\substack{k \in K: \\ x(i_k^a) \leqslant q}} w_k - \sum_{\substack{k \in K: \\ x(i_k^d) \leqslant q}} w_k,$$

where $x(i_k^a)$ and $x(i_k^d)$ are the numbers of extreme jobs of course k in permutation $\Lambda = (\lambda_1, \ldots, \lambda_e)$. This value determines the "contribution" of a job in the original objective function (6) if this job is placed into cell q. Denote by $q^*(j)$ the first number from $q_1(j)$ to $q_2(j)$ for which the minimum of f is reached:

$$q^*(j) = \min \left\{ t \mid f(t) = \min_{q_1(j) \leqslant q \leqslant q_2(j)} f(q) \right\}. \tag{9}$$

We will call $q^*(j)$ the optimal cell for job j, $j \in I$. The following lemma shows that if one job should precede another one in a schedule, then its optimal cell is not greater than the optimal cell for another job.

Lemma 4. *If $j \to g$ in graph $G'(I, E')$ for two non-extreme jobs j and g, then*

a) $q_1(j) \leqslant q_1(g)$;

b) $q_2(j) \leqslant q_2(g)$;

c) $q^*(j) \leqslant q^*(g)$.

Proof. a) Since $j \to g$, then $g \in D(j)$. If $q_1(j) = 0$, then the statement is obvious. If $q_1(j) > 0$, then $j \in D(\lambda_{q_1(j)})$. It means that $g \in D(\lambda_{q_1(j)})$. Then by definition of $q_1(g)$ we get $q_1(g) \geqslant q_1(j)$.

b) Since $j \to g$, then $j \in A(g)$. If $q_2(g) = e$, then the statement is obvious. If $q_2(g) < e$, then $g \in A(\lambda_{q_2(g)})$. It means that $j \in A(\lambda_{q_2(g)})$. Then by definition of $q_2(j)$ we get $q_2(j) \leqslant q_2(g)$.

c) Let $q^*(j) > q^*(g)$. Taking into account a) and b), we obtain

$$q_1(j) \leqslant q_1(g) \leqslant q^*(g) < q^*(j) \leqslant q_2(j) \leqslant q_2(g).$$

This means that both cells $q^*(j)$ and $q^*(g)$ are available for jobs j and g. This contradicts cell selection rule (9). Indeed, if $f(q^*(j)) = f(q^*(g))$, then cell $q^*(g)$ should be selected for both jobs as the earlier one. If $f(q^*(j)) \neq f(q^*(g))$, then the cell with the minimum value of f should be selected for both jobs. The lemma is proven.

For each cell $q \in \{0, \ldots, e\}$ we introduce a set of jobs I_q for which this cell is optimal:

$$I_q = \{j \in I \setminus I_{ad} \ : \ q^*(j) = q\}, \ q \in \{0, \ldots, e\}.$$

Let $E_q \subset E$ be the set of arcs connecting the vertices of I_q, $q \in \{0, \ldots, e\}$. Denote by $\bar{\pi}(I_q, E_q)$ an arbitrary topological sorting of the graph $G_q(I_q, E_q)$, i.e. some permutation of jobs from I_q satisfying the partial order given by the set of arcs E_q. Due to acyclicity of the original graph $G(I, E)$, a topological sorting of any of its subgraphs $G_q(I_q, E_q)$ exists. The following theorem shows that, by ordering jobs in each cell separately, we can get an optimal schedule for problem P_Λ as follows: first we need to complete all jobs from set I_0, then complete the extreme job λ_1, then—all jobs from set I_1, the extreme job λ_2, etc.

Theorem 2. *The schedule* $\pi^\Lambda = (\bar\pi(I_0, E_0), \lambda_1, \bar\pi(I_1, E_1), \lambda_2, \ldots, \lambda_e, \bar\pi(I_e, E_e))$ *is an optimal solution of problem* P_Λ.

Proof. The schedule π^Λ is feasible in problem P_Λ. Indeed, consider any two jobs $i, j \in I \setminus I_{ad}$ such that $i \to j$. If these jobs are in the same cell, then the precedence constraint is satisfied due to the construction of the topological sorting of all jobs from this cell. If i and j are in different cells, then by virtue of lemma 4 we have $q^*(i) < q^*(j)$, which means that in schedule π^Λ job i will be executed before job j. The precedence constraints between extreme jobs and all other jobs are satisfied by virtue of the rule of constructing optimal cells.

The optimality of the solution follows from the definition of an optimal cell. Indeed,

$$\sum_{j \in I\setminus I_{ad}} f(q^*(j))p_j = \sum_{j \in I\setminus I_{ad}} \min_{q_1(j)\leqslant q\leqslant q_2(j)} \left(\sum_{\substack{k \in K:\\ x(i_k^a)\leqslant q}} w_k - \sum_{\substack{k \in K:\\ x(i_k^d)\leqslant q}} w_k \right) p_j = \min_{\pi} \sum_{j \in I\setminus I_{ad}} W_j(\pi)p_j.$$

Since for the given order Λ, the contribution to the objective function of extreme jobs is fixed and equal to

$$\sum_{j \in I_{ad}} \left(\sum_{\substack{k \in K:\\ x(j)\geqslant x(i_k^a)}} w_k - \sum_{\substack{k \in K:\\ x(j)> x(i_k^d)}} w_k \right) p_j,$$

this means that the schedule π^Λ, which corresponds to the distribution of jobs across cells, delivers the minimum of objective function (6)–(7). The theorem is proven.

Thus, solving problem P_Λ can be reduced to calculating the optimal cell for each job and ordering jobs in each cell separately. A general scheme of finding a solution to the auxiliary problem is described by Algorithm 1. Let's evaluate the complexity of this approach. It is necessary to construct sets $A(\lambda)$, $D(\lambda)$ for each extreme job λ, which in total will require $O(nK)$ operations. Next, for each job j, it is necessary to define the boundaries $q_1(j)$, $q_2(j)$ and the optimal cell $q^*(j)$ that will required $O(nK)$ operations. Building partial schedules in each cell needs no more than $O(n + |E|)$ operations [12].

Algorithm 1 Procedure $Solv(\Lambda)$

$\pi^\Lambda := ()$
for all $q \in \{0, 1, \ldots, e\}$ **do**
    $I_q := \emptyset$
end for
Generate graph $G'(I, E')$ by permutation $\Lambda = (\lambda_1, \ldots, \lambda_e)$
for all $j \in I \setminus I_{ad}$ **do**
    Calculate $q^*(j)$
    $I_{q^*(j)} := I_{q^*(j)} \cup \{j\}$
end for
Build $\bar\pi(I_0, E_0)$
$\pi^\Lambda := \bar\pi(I_0, E_0)$
for all $q \in \{1, \ldots, e\}$ **do**
    Build $\bar\pi(I_q, E_q)$
    $\pi^\Lambda := \pi^\Lambda \cup (\lambda_q, \bar\pi(I_q, E_q))$
end for
Return π^Λ

5. AN ALGORITHM FOR SOLVING PROBLEM $1|prec|H$

Denote by B the set of all possible permutations of extreme jobs Λ that do not contradict the precedence constraints of the original problem. If the number of courses in the problem is small, or due to the structure of graph $G(I, E)$, the mutual order of extreme jobs does not allow a large number of options, an efficient search for a solution to the problem is possible. It is based on iterating through permutations of extreme jobs and solving an auxiliary problem for each permutation. Thus, the scheme of solving the problem can be represented as Algorithm 2, where H^*_Λ is the optimal value of the objective function in the auxiliary problem P_Λ, and H^*, π^* are the optimal value and optimal the schedule of the original problem $1|prec|H$, respectively. Note that in Algorithm 2 there is no need to find a schedule for each permutation Λ under consideration, since to calculate the value of H^*_Λ, it is enough to know the optimal cells for each job.

Algorithm 2 Solving problem $1|prec|H$

```
H* := +∞
for all Λ ∈ B do
    Calculate H*_Λ
    if H*_Λ < H* then
        H* := H*_Λ
        Λ* := Λ
    end if
end for
π* := Solv(Λ*)
Return π*
```

The algorithm for solving the problem has the complexity $O(|B|(nK+|E|))$, where n is the total number of jobs, K is the number of courses, $|E|$ is the number of edges in the precedence graph and $|B|$ is the number of feasible permutations of the extreme jobs of the courses. The largest contribution to the complexity is given by the value $|B|$. The maximum possible value of $|B|$ is $\frac{(2K)!}{2^K}$, when all possible permutations of extreme jobs of courses are considered without taking into account their precedence relations. However, in the case of, for example, dense precedence graphs, the value $|B|$ may be acceptable for using Algorithm 2 even with a large number of courses.

During computational experiments, the proposed algorithm was compared with the optimization solver IBM ILOG CPLEX 22.1.0.0 [13]. To apply this solver, the following formulation of the problem was used in the form of an integer linear programming problem:

$$\sum_{k\in K}\sum_{j\in I, j\neq i^a_k} w_k p_j x_{i^a_k,j} + \sum_{k\in K}\sum_{j\in I, j\neq i^d_k} w_k p_j x_{j,i^d_k}$$
$$+ \sum_{k\in K}\left(p_{i^a_k} + p_{i^d_k} - \sum_{i\in I} p_i\right) \to \min,$$
$$x_{i,j} + x_{j,i} = 1 \ \forall i, j \in I;$$
$$x_{i,j} + x_{j,k} + x_{k,i} \geqslant 1 \ \forall i, j, k \in I;$$
$$x_{i,j} = 1 \ \forall (i, j) \in E;$$
$$x_{i,j} \in \{0, 1\} \ \forall i, j \in I,$$

where variable $x_{i,j}$, $i \neq j \in I$, takes the value 1 if job i is executed before job j, and the value 0 otherwise. Such variables and constraints are standard for integer formulations of single machine problems with precedence constraints (see, for example, [14]).

Table 1. Test results for sparse graphs

n	K	$\|E\|$	$\|B\|$	Algorithm 2	CPLEX
100	3	481	6	0.007	13.218
	5	467	4299	5.124	13.436
	7	525	26 244	35.191	12.774
200	3	1864	6	0.034	106.282
	5	1942	150	0.516	105.152
	7	1990	870	3.169	102.967
300	3	4550	5	0.050	426.563
	5	4423	10	0.069	404.386
	7	4410	950	6.834	396.179
400	3	7765	1	0.022	>10 min
	5	7662	4	0.051	>10 min
	7	7746	280	3.362	>10 min
500	3	12 241	2	0.037	>10 min
	5	12 118	2	0.065	>10 min
	7	12 194	96	1.854	>10 min

Table 2. Test results for dense graphs

n	K	$\|E\|$	—B—	Algorithm 2	CPLEX
100	3	2514	2	0.009	11.144
	5	2541	4	0.054	10.717
	7	2515	3	0.025	10.777
	9	2487	8	0.048	10.668
200	3	10 103	1	0.028	94.987
	5	10 194	2	0.071	95.790
	7	10 199	2	0.043	99.734
	9	10 123	4	0.165	94.775
300	3	22 846	1	0.034	386.899
	5	22 943	1	0.039	398.339
	7	22 933	6	0.225	378.841
	9	22 845	4	0.156	400.153
400	3	40 862	1	0.062	¿10 min
	5	40 692	2	0.135	¿10 min
	7	40 779	1	0.094	¿10 min
	9	40 806	2	0.147	¿10 min
500	3	63 657	1	0.090	¿10 min
	5	63 424	1	0.098	¿10 min
	7	63 676	1	0.136	¿10 min
	9	63 802	3	0.316	¿10 min

The calculations were performed on a personal computer (Intel Core i7-7700K, 4.2 GHz, 32.0 GB), the algorithm was implemented in Python using NetworkX library for working with graphs. In Tables 1 and 2 the results of solving randomly generated problems are given. Random integers from range [1; 10] were chosen for weights of courses and processing times of jobs. Random vertices of graphs were chosen as extreme jobs of courses in such a way that precedence constraints between the first and the last vertices of each course were not violated. The running time of the algorithm and the CPLEX solver was limited to 10 minutes.

Notations n, K, $|E|$, $|B|$ used in the tables coincide with the notations adopted earlier in the article, and columns “Algorithm 2” and “CPLEX” indicate the time of solving problems in seconds by the algorithm proposed in the article and by CPLEX, respectively.

We can see from Table 1 that running time of the algorithm depends on the cardinality of set B more than on the total number of jobs. So, the algorithm gives an exact solution to the problems of high dimension in a fraction of a second, if the number of feasible permutations of extreme jobs is small. In the case of large value of $|B|$ (see, for example, the problem with $n = 100$, $K = 7$), the running time of the algorithm increases significantly. CPLEX, on the contrary, is insensitive to changes of $|B|$ and K, but with an increase of n, its running time increases greatly.

In Table 2 results of solving problems with denser graphs are presented. Here, as expected, the number of feasible permutations of extreme jobs is less, so the algorithm found solutions in all problems in less than a second.

Thus, the results of the computational experiment confirm the theoretical estimation of the complexity of the developed algorithm and show that the algorithm can be efficiently applied to problems with a small set B, which corresponds to the case of either problems with dense precedence graphs, or problems with a small number of courses, or problems in which the positions of extreme jobs are fixed relative to each other. In these cases, the algorithm allows us to quickly solve high-dimensional problems.

6. CONCLUSION

The article considers the single machine problem with precedence constraints, in which it is necessary to minimize the total weighted duration of courses (some subsets of jobs). The NP-hardness of the problem under consideration is proved. An exact algorithm for its solving is proposed. This algorithm depends polynomially on the total number of jobs and allows solving problems efficiently, if there is a small number of options for the relative location of extreme jobs of courses. The direction of further research may concern a general formulation of the problem, when extreme jobs of courses are not clearly defined. Resources constrained project scheduling problem with the considered goal function can also be considered.

REFERENCES

1. Brucker, P., *Scheduling algorithms*, Springer: Heidelberg, 2007.
2. Lazarev, A.A., *Teoriya raspisanii. Metody i algoritmy* (Theory of Schedules. Methods and Algorithms), Moscow: ICS RAS, 2019.
3. Lazarev, A., Khusnullin, N., Musatova, E., Yadrentsev, D., Kharlamov, M., and Ponomarev K., Minimization of the weighted total sparsity of cosmonaut training courses, *OPTIMA 2018. Communications in Computer and Information Science*, 2019, pp. 202–215.
4. Harhalakis, G., Special features of precedence network charts, *Eur. J. Oper. Res.*, 1990, vol. 49, no. 1, pp. 50–59.
5. Csébfalvi, A.B. and Csébfalvi, G., Hammock activities in project scheduling, *Proceedings of the Sixteenth Annual Conference of POMS*, 2005.
6. Eliezer, O., A new bi-objective hybrid metaheuristic algorithm for the resource-constrained hammock cost problem (RCHCP), *Doctoral Dissertation*, Pécs, 2011.
7. El-Rayes, K. and Moselhi, O., Resource-driven scheduling of repetitive activities, *Construction Management and Economics*, 1998, vol. 16, no. 4, pp. 433–446.
8. Vanhoucke, M., Work continuity constraints in project scheduling, *Working Paper 04/265*, Belgium: Ghent University, Faculty of Economics and Business Administration, 2004.
9. Vanhoucke, M. and Van Osselaer, K., Work continuity in a real-life schedule: the Westerschelde Tunne, *Working Paper 04/271*, Belgium: Ghent University, Faculty of Economics and Business Administration, 2005.

10. Graham, R.L., Lawler, E.L., Lenstra, J.K., and Rinnooy Kan, A.H.G., Optimization and approximation in deterministic sequencing and scheduling: a survey, *Annals of Discrete Mathematics*, 1979, vol. 5, pp. 287–326.

11. Lenstra, J.K. and Rinnooy Kan, A.H.G., Complexity of scheduling under precedence constraints, *Oper. Res.*, 1978, vol. 26, no. 1, pp. 22–35.

12. Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C., *Introduction to algorithms*, MIT press, 2022.

13. IBM ILOG CPLEX Optimization Studio,
URL: https://www.ibm.com/products/ilog-cplex-optimization-studio.

14. Potts, C.N., An algorithm for the single machine sequencing problem with precedence constraint, *Combinatorial Optimization II. Mathematical Programming Studies*, Springer: Berlin, Heidelberg, 1980, vol. 13, pp. 78–87.

This paper was recommended for publication by P.Yu. Chebotarev, a member of the Editorial Board