Estimating Maximum Resource Load for Resource-Constrained Project Scheduling Problem

Alexander Lazarev
Moscow State University,
National Research University Higher School of Economics,
Moscow Institute for Physics and Technology,
Institute of Control Sciences
65 Profsoyuznaya street,
117997 Moscow, Russia
jobmath@mail.ru

German Tarasov
Institute of Control Sciences
Moscow State University
GSP-1, Leninskie Gory,
100109 Moscow, Russia
gv.tarasov@physics.msu.ru

Dmitry Arkhipov
ISAE-SUPAERO
10, avenue Edouard-Belin,
Toulouse, France,
Institute of Control Sciences
65 Profsoyuznaya street,
117997 Moscow, Russia
miptrafter@gmail.com

Abstract

This paper focuses on estimating maximum resource loads in Resource-Constrained Project Scheduling Problem (RCPSP). We examine a variant of vector sum problem with fractions: considering preemptions allowed, determine what part of each of n jobs should be accomplished in order to minimize sum of squares of non-consumed amounts of resources, taking into account the resource constraints along with minimizing the number of preemptions. We prove that in case of 2 resources, the optimal solution contains only 2 or less preemptions, and present two polynomial algorithms of finding such solution with time complexities of $O(n^2)$ operations. We also present an investigation of the general case of arbitrary number of resources.

1 Introduction

Resource-Constrained Project Scheduling Problem (RCPSP) is one of the core problems of scheduling theory. It has been widely explored throughout the last years. RCPSP is NP-hard in the strong sense [Garey et al., 1975], which means no exact approach to it may find a solution in polynomial time. The exact approaches to RCPSP

In: Yu. G. Evtushenko, M. Yu. Khachay, O. V. Khamisov, Yu. A. Kochetov, V.U. Malkova, M.A. Posypkin (eds.): Proceedings of the OPTIMA-2017 Conference, Petrovac, Montenegro, 02-Oct-2017, published at http://ceur-ws.org

include, but are not limited to branch-and-bound methods, linear programming, constraint programming and dynamic programming methods. For all of these methods, knowing the lower makespan bounds allows to vastly reduce search time. In [Arkhipov et al., 2017], a polynomial algorithm of evaluating lower makespan bounds is presented. The algorithm is based on estimating maximum resource load for a given pair of resources, which is the main goal of the present paper.

2 Preliminaries

The problem of estimating maximum resource load can be formulated in the following way. There is a set of jobs (tasks) N and a set of non-renewable resources R, and available amounts $r_j \in \mathbb{R}^+$ of resources $j \in R$. Each task may be either completed, rejected or executed partially. In order to complete task $i \in N$, amount $r_{ij} \in \mathbb{R}^+$ of each resource $j \in R$ is required.

The *completion ratio* x_i defines the executed part of each task $i \in N$. Completion ratio value $x_i = 1$ corresponds to full completion of task $i \in N$, $x_i = 0$ correspond to rejection, and all the other values in $(0, 1)$ correspond to partial execution (preemption). The required resources are consumed in proportion to *completion ratio* of this job $x_i \in [0, 1]$, i.e. amount $r_{ij}x_i$ of each resource $j \in R$ is needed.

The goal is to determine the vector $X = \{x_i\}$ of completion ratios such that the aggregate amount of consumed resources is maximized $\sum_{j \in R} \sum_{i \in N} x_i r_{ij} \to \max$ and the resource constraints $\sum_{i \in N} x_i r_{ij} \le r_j \, \forall j \in R$ are satisfied.

3 Estimating Maximum Resource Load for Two Resources

In this section we will examine the case of two resources. As will be shown later, this problem can be solved in polynomial time. Let us denote the available amounts of resources as A, B and resource consumption amounts as a_i, b_i. We will also associate each task $i \in N$ with a vector $\vec{s_i} = (a_i, b_i)$ in a two-dimensional rectangular Cartesian coordinate system. For the sake of convenience, here and below we will consider $N = \{1, \ldots, n\}$, where n is the total number of tasks.

3.1 Formulation of the Problem

Let us reformulate the problem in a formal manner: given a pair of positive real numbers $A, B \in \mathbb{R}^+$ and a list of tasks described by a set $N = \{1, \ldots, n\}$ of indices and set $S = \{\vec{s_i}\} \subset \mathbb{R}^{+2}$, $i \in N$ of vectors $\vec{s_i} = (a_i, b_i)$ on a two-dimensional rectangular Cartesian coordinate system, find vector $\vec{X} = \{x_i\} \in \mathbb{R}^n$ subject to constraints $A - \sum_{i \in N} a_i x_i \ge 0, \quad B - \sum_{i \in N} b_i x_i \ge 0, \quad x_i \in [0, 1]$ with objective function $\sum_{i \in N} a_i x_i + \sum_{i \in N} b_i x_i \to \max$.

Consider the classical "0-1" knapsack problem, which is known to be NP-hard, and can be solved in pseudo-polynomial time [Pisinger, 1997]. If boolean decision variables are replaced with real decision variables (knapsack with fractions), the problem becomes solvable in polynomial time, as was shown by Dantzig [Dantzig, 1957]. The problem of estimating resource loads can also be regarded as a variant of subset sum (or vector sum) problem, which is essentially a special case of the knapsack problem such that the cost of any item is equal to the weight of this item. Therefore, one-dimensional subset sum problem with fractions is also solvable in polynomial time. In [Baburin et al., 2007], a variant of vector sum problem is considered, and polynomial algorithms are presented for the case of arbitrary dimensionality. However, the problem considered in [Baburin et al., 2007] differs from the one considered in our article in that it does not involve any resource constraints, and the decision variables are boolean. All the problems mentioned above are related to the vector partitioning problem, which has been covered in [Onn et al., 1999]. Thus, the formulation of estimating resource loads problem takes the form of two-dimensional variant of subset sum problem with fractions. One can assume that since the one-dimensional subset sum problem with fractions is solvable in polynomial time, a polynomial approach to the two-dimensional variant should exist as well, on which we will elaborate further.

As was mentioned before, the objective of the problem is to find a list of completion ratios that maximizes resource consumption, taking the resource constraints into account. Obviously, if the amounts of available resources exceed or are equal to their maximum possible consumption ($\sum_{i=1}^{n} a_i \le A, \quad \sum_{i=1}^{n} b_i \le B$), the solution would be $X = \{x_i\} | \forall \, i \, x_i = 1$, i.e. each task should be fully completed. However, this is rarely the case.

3.2 Geometric Approach to Determining Problem Solvability

Consider two permutations γ_{up} and γ_{down} of vectors s_i acquired by sorting these vectors according to non-increase and non-decrease of the ratio $\frac{b_i}{a_i}$. Let us note the order of vectors according to these permutations as $s_i^{\gamma_{up}}$ and

coordinates of these vectors as $a_i^{\gamma_{up}}$ and so forth. For γ_{up}, the following inequality must be satisfied

$$\forall i \in \overline{1, n-1} \quad \frac{b_i^{\gamma_{up}}}{a_i^{\gamma_{up}}} \geq \frac{b_{i+1}^{\gamma_{up}}}{a_{i+1}^{\gamma_{up}}}$$

For γ_{down} the sign of this inequality is reversed. The division operation here is rather pro forma: obviously, the case of dividing by zero must be handled with a separate conditional operator. If, for any two vectors, ratios of their components $\left(\frac{b_i}{a_i}\right)$ happen to be equal, the order of these vectors in permutations γ_{up} and γ_{down} for now is considered to be of no importance.

Note: obviously, sorting changes order of the vectors, which means the original indexation like $s_i, x_i, i \in N$ would not make any sense. However, we will continue to use this style of indexation just for convenience's sake. When implementing the algorithms, it is useful to keep the original index stored as an object's (vector's) property.

Let us place vector $s_1^{\gamma_{up}}$ starting at the origin of our coordinate system, then place the second vector $s_2^{\gamma_{up}}$ starting at the end of first one, and so forth. We will name the resultant polygonal line as γ_{up}, too. Let us also do the same operations with elements of γ_{down}, resulting in polygonal line γ_{down}. Clearly, any permutation of n vectors corresponds to some polygonal curve built with vectors $\vec{s_i}$, and, if provided a set of coefficients $\vec{X} = \{x_i\}$, together this permutation and this set of coefficients correspond to a polygonal curve built with vectors $(x_i\vec{s_i})$.

Since in γ_{up}, γ_{down} vectors are sorted according to non-increase and non-decrease of tangent of the angle between each vector and the abscissa axis of our coordinate system, these permutations also correspond to sorting according to the value of this angle. Thus, the polygonal line γ_{up} is convex upward, and γ_{down} is convex downward, and together these lines form a convex polygon on the coordinate plane. We will name this polygon the *solvability domain* of considered instance of the problem.

Lemma: Any polygonal curve γ' comprised of vectors $s_i \in S$ is located completely within the solvability domain of considered instance of the problem.

Proof: We will view the considered polygonal curves as plots of piece-wise linear functions. For example the curve γ corresponds to function $b_\gamma(a)$. Of course, if some of the vectors has a zero A component ($a_i = 0$), the resulting curve cannot be viewed as a function of A. However, these cases too can be handled by additional condition operators.

Let γ be such a polygonal curve that $\forall \gamma'$, $\forall a \in [0, \sum_{i=1}^n a_i] \quad b_\gamma(a) \geq b_{\gamma'}(a)$. Then γ is identical to γ_{up} (neglecting the cases in which $\exists\, i, j : \frac{b_i^\gamma}{a_i^\gamma} = \frac{b_j^\gamma}{a_j^\gamma}$), i.e. it is convex upward in each of its breaking points. To prove this, suppose the contrary: let γ be convex downward in its j-th breaking point, i.e. $\exists\, j \in \overline{1, n-1} : \frac{b_j^\gamma}{a_j^\gamma} < \frac{b_{j+1}^\gamma}{a_{j+1}^\gamma}$. Consider a polygonal line ψ that differs from γ in that its j-th and $(j+1)$-th vectors are placed in reverse order: $s_j^\gamma = s_{j+1}^\psi$, $s_{j+1}^\gamma = s_j^\psi$, $\forall i \in N \setminus \{j, j+1\}\ s_i^\gamma = s_i^\psi$. Then ψ is convex upward in its j-th breaking point, and $\forall\, a \in (\sum_{i=1}^{j-1} a_i^\gamma, \sum_{i=1}^{j+1} a_i^\gamma) \quad b^\gamma(a) < b^\psi(a)$, which contradicts the supposition that $\forall \gamma'$, $\forall a \in [0, \sum_{i=1}^n a_i]\, b_\gamma(a) \geq b_{\gamma'}(a)$. Thereby γ is identical to γ_{up}. Similarly, by considering another polygonal curve γ, this time with the condition $\forall \gamma'$, $\forall a \in [0, \sum_{i=1}^n a_i] \quad b_\gamma(a) \leq b_{\gamma'}(a)$, we can prove that $\forall\, a \in [0, \sum_{i=1}^n a_i] \quad b_{\gamma_{down}}(a) \leq b_{\gamma'}(a)$. Thereby $\forall\, \gamma'$, $\forall a \in [0, \sum_{i=1}^n a_i] \quad b_{\gamma_{up}}(a) \geq b_{\gamma'}(a) \geq b_{\gamma_{down}}(a)$. ■

Corollary: Obviously, for any polygonal curve γ' composed of vectors $x_i\vec{s_i}$ such that $\vec{X} = \{x_i\}$ does not violate the constraints $x_i \in [0, 1]$

$$\forall\, \gamma',\, \forall \vec{X} \,|\, x_i \in [0, 1]\, \forall i \in N$$

a similar inequality holds true

$$\forall a \in [0, \sum_{i=1}^n a_i x_i] \quad b_{\gamma_{up}}(a) \geq b_{\gamma'}(a) \geq b_{\gamma_{down}}(a)$$

i.e. such polygonal curves are also located within the solvability domain, which leads to the following:
$\forall\, X = \{x_i\} |\, x_i \in [0, 1]\, \forall\, i \in N$ point $(\sum_{i=1}^n a_i x_i,\, \sum_{i=1}^n b_i x_i)$ is located within the solvability domain.

Note: X should not necessarily be a solution to regarded instance of the problem.

This allows us to formulate the **necessary and sufficient condition** of possibility to consume all the available resources: full consumption of the available resources A and B is possible if and only if point (A, B) is located within the solvability domain. To verify the latter, consider the boundary of the solvability domain, which is a pair polygonal curves γ_{up} and γ_{down}, to be defined by a pair of piece-wise linear functions $b_{\gamma_{up}}(a)$, $b_{\gamma_{down}}(a)$

and check if $b_{\gamma_{up}}(A) \geq B \geq b_{\gamma_{down}}(A)$. For an empty task set $S = \emptyset$ the problem is considered solvable only if $A = 0$, $B = 0$.

3.3 Algorithms.

In this section we'll provide two algorithms of finding the solution that leads to full consumption of available resources; the time complexities are $O(n^2)$.

First we'll examine the problem of full resource consumption, which is to check if $\exists X = \{x_1, \ldots, x_n\} : A - \sum_{i=1}^{n} a_i x_i + B - \sum_{i=1}^{n} b_i x_i = 0$, taking into consideration all the constraints mentioned before. The case in which full resource consumption is not possible will be examined after both of the algorithms (see Auxiliary Algorithm).

3.3.1 Algorithm 1

The first algorithm is aiming at consequently reducing the current instance of full resource consumption problem to a solvable subproblem that has a smaller number of tasks. This continues until the cardinality of current set of tasks S reaches either 2, 1 or 0. Then the remaining problem is identical to solving a simple system of linear equations.

Consider a fixed set of tasks $N = \{1, \ldots, n\}$ associated with vector set $S = \{s_1, \ldots, s_n\}$ and a resource vector (A, B). Suppose this instance of the full resource consumption problem has a solution $\vec{X} = \{x_1, \ldots, x_n\}$.

Notice that if $\exists\, i \in \overline{1,n} : x_i = 0$, then the solution $\vec{X}$ of this problem is identical to the solution $\vec{X'}$ of a subproblem for the task set $S' = S \setminus s_i$ and resource vector (A, B), except that in $\vec{X'}$ the component x_i is skipped: $\forall\, j < i\ x'_j = x_j$, $\forall\, j > i\ x'_j = x_{j+1}$. Similar applies if $\exists\, i \in \overline{1,n} : x_i = 1$, then the solution $\vec{X}$ of this problem is identical to the solution $\vec{X'}$ except that the resource vector in the subproblem would be $(A - a_i, B - b_i)$.

Let us introduce a boolean function $u(s) : S \to \{0, 1\}$. $u(s) = 1$ means vector task s has been marked as used. The first algorithm is as follows.

Algorithm 1

Step 1. Sort the tasks of the original set S by non-increase of ratio $\frac{b_i}{a_i}$. If $a_i = 0$, place the corresponding vector at the end of the list.

Step 2. Check if full resource consumption is possible (check if the point (A, B) lies within the solvability domain). If not – interrupt the algorithm.

Step 3. Mark all the task vectors $s_i \in S$ as not used: $\forall\, s_i \in S\ u(s_i) := 0$.
Cycle 1: until there is an unused task $(\exists\, s_i \in S : \mid u(s_i) = 0)$:
1) Find an unused task vector $s_i \in S \,|\, u(s_i) = 0$.
Mark s_i as used: $u(s_i) := 1$. Form a subset of tasks S'_i by removing task s_i from S: $S'_i = S \setminus s_i$.
2) Check if full resource consumption is possible for S'_i and (A, B). If it is, assign $x_i := 0$, $S := S'_i$.
Repeat the cycle.

Step 4. Cycle 2: until there is an unused task $(\exists\, s_i \in S :\ u(s_i) = 0)$:
1) Select an unused task vector $s_i \in S \,|\, u(s_i) = 0$.
Mark s_i as used: $u(s_i) := 1$. Form a subset of tasks $S'_i = S \setminus s_i$ and a modified resource vector (A', B') where $A' = A - a_i$, $B' = B - b_i$.
2) Check full resource consumption is possible for S'_i and (A', B'). If it is, assign $x_i := 1$, $S := S'_i$.
Repeat the cycle.

Step 5. If set S consists of two tasks j, k – solve the following system

$$\begin{cases} x_j a_j + x_k a_k = A \\ x_j b_j + x_k b_k = B \end{cases}$$

If S consists of only one task j, solve any of the equations of the corresponding system. If S is empty, no action is needed.

End.

Proposition: Time complexity of Algorithm 1 is at most $O(n^2)$, where n is the number of tasks in the initial task set.

Proof: Examining each step of the Algorithm 1 individually, one can make sure that the most time-consuming steps are Step 3 and Step 4, at which the solvability of a selected subproblem is tested, taking $O(n)$ operations for each subproblem instance. In total at most n subproblems are examined during both steps, taking $O(n^2)$ or less operations. Thus, the time complexity of Algorithm 1 is $O(n^2)$. ■

3.3.2 Algorithm 2

The second algorithm is aimed at finding a pair of indices j, k such that solution α, β of the system

$$\begin{cases} \sum\limits_{i=1}^{k-1} a_i + \beta a_k = A + \sum\limits_{i=1}^{j-1} a_i + (1-\alpha)a_j \\ \sum\limits_{i=1}^{k-1} b_i + \beta b_k = B + \sum\limits_{i=1}^{j-1} b_i + (1-\alpha)b_j \end{cases}$$

satisfies the constraints $0 \le \alpha, \beta \le 1$ (the indexation is provided as if the vectors s_i were already sorted). Then, the solution of the whole problem is constructed from this pair of indices j, k and coefficients α, β.

Let us modify the system:

$$\begin{cases} \alpha a_j + \sum\limits_{i=j+1}^{k-1} a_i + \beta a_k = A \\ \alpha b_j + \sum\limits_{i=j+1}^{k-1} b_i + \beta b_k = B \end{cases} \quad (1)$$

Examining each pair j, k, which would take at least $O(n^2)$ time, is not necessary: note that for each fixed j, the coefficients β form a monotoneous non-increasing sequence depending on k (we'll provide a way of finding such pair of α, β even if (1) has more than one solution).

Suppose that $j \in N$ is fixed, and $k_1, k_2 \in N$ are such that $k_1 < k_2$. Let α_1, β_1 be the solution of (1) for the pair of indices j, k_1, and α_2, β_2 the solution of (1) for j, k_2. Then, since all the variables and coefficients in (1) are non-negative, $\beta_1 \ge \beta_2$. Therefore, a binary search can be implemented to slightly speed up the algorithm. The algorithm is constructed as follows.

Algorithm 2

Step 1. Sort the tasks of set S by non-increase of ratio $\frac{b_i}{a_i}$. If $a_i = 0$, place the corresponding vector at the end of the list.

Step 2. Check if the problem of full resource consumption is solvable (check if the point (A, B) lies within the solvability domain). If the problem is unsolvable – interrupt the algorithm.

Step 3. Cycle:

For each $j, k \in N$ calculate and memorize values of sums $\sum\limits_{i=j+1}^{k-1} a_i$, $\sum\limits_{i=j+1}^{k-1} b_i$.

Step 4. Set $j = 1$. **Cycle**:

For a fixed $j \in N$, try to find (via binary search) the index k such that the solution α, β of (1) meets the constraint $0 \le \beta \le 1$. When solving (1), use the values of sums calculated previously.

It may happen that for some pairs j, k system (1) has more than one solution, i.e. its determinant turns out to be 0. In such cases, we simply assign one of the variables α, β equal to 0 and solve the remaining equation to find the other variable.

If the obtained values α, β fit within the constraints $0 \le \alpha, \beta \le 1$, then **interrupt the cycle**. Otherwise, set $j := j + 1$ and **repeat the cycle**.

Note: the situation in which none of the constraints were met and j reaches n is impossible, since it would mean that full resource consumption is not possible and the algorithm must have been interrupted at Step 2.

Step 5. Assign
$\forall\, i < j:\ x_i := 0;$
$x_j := \alpha;$
$\forall\, i \,|\, j < i < k:\ x_i = 1;$
$x_k := \beta;$
$\forall\, i > k:\ x_i = 0;$

End.

Proposition: Time complexity of Algorithm 2 is $O(n^2)$, where n is the number of tasks in the initial task set.

Proof: Examining each step of the Algorithm 2 individually, one can make sure that the most time-consuming step is Step 3, which can be done in $O(n^2)$. Thus, the time complexity of Algorithm 2 is $O(n^2)$. ■

3.3.3 Auxiliary Algorithm

To account for cases in which full resource consumption is not possible (e.g., $\sum i = 1^n a_i x_i < A$), another algorithm is needed.

Auxiliary Algorithm

Step 1. Sort vectors s_i according to non-decrease of ratio $\frac{b_i}{a_i}$. If $a_i = 0$, place the corresponding vector at the end of the list.

Step 2. Assign $a := 0,\ b := 0,\ i := 1$

Step 3. Cycle: while $(a + a_i \le A)\&(b + b_i \le B)\&(i < n)$ do $(x_i := 1,\ a := a + a_i,\ b := b + b_i,\ i := i + 1)$.

Step 4. If $(i < n)$, assign $x_i := \min\left(\frac{B-b}{b_i},\ \frac{A-a}{a_i}\right),\ i := i+1$ (only the minimum of these two values will meet the constraint $0 \le x_i \le 1$).
Cycle: while $(i \le n)$ do $(x_i := 0,\ i := i + 1)$.

End.

Proposition: Time complexity of Auxiliary Algorithm is at most $O(n \log n)$, where n is the number of tasks in the initial task set.

Proof: Examining each step of the Auxiliary Algorithm individually, one can make sure that the most time-consuming step is Step 1, at which sorting the tasks takes $O(n \log n)$ time using Hoare's quicksort. Thus, the time complexity of Auxiliary Algorithm is $O(n \log n)$. ■

3.3.4 Main Algorithm

The algorithm to solve the problem of estimating maximum resource load for two resources (the main algorithm) consists of either Algorithm 1 or Algorithm 2 supplemented with the Auxiliary Algorithm. Overall time complexity, therefore, can be either $O(n^2)$ in case $\sum_{i=1}^{n} a_i x_i < A$, $\sum_{i=1}^{n} b_i x_i < B$ or $O(n \log n)$ otherwise.

4 Estimating Maximum Resource Load for Arbitrary Number of Resources

The problem is close to a linear relaxation of multi-dimensional knapsack problem, with resource constraints being regarded as knapsack capacity constraints. The multi-dimensional knapsack problem has been widely described in [Kellerer et al., 2004].

We'll call *complete solution* such a solution of the problem that allows for full resource consumption. We'll provide a polynomial algorithm that allows to find a complete solution (if such exists and is unique) for the case $r \ge n$.

Let us denote the matrix of required resource amounts r_{ij} as R, vector column of sought-for completion ratios x_i as X, and the vector column of available resource amounts a_i as A. Finding a complete solution of the original problem is then equivalent to solving a system of linear algebraic equations $RX = A$ subject to constraints $0 \le x_i \le 1$, $i = 1, \ldots, n$.

We'll describe the algorithm first and then discuss the ideas behind it.

Algorithm for the case $r \geq n$

Step 1. Check if $r \geq n$. If not – interrupt the algorithm.
Use the Gauss-Jordan elimination method on the augmented matrix $(R|A)$ to obtain the reduced row echelon form R^* of matrix R and the transformed right-hand side A^*.
Check if $\operatorname{rank} R^* = \operatorname{rank}(R^*|A^*)$. If not – interrupt the algorithm.

Step 2. Check if $\operatorname{rank} R = \operatorname{rank} R^* = n$. If not, interrupt the algorithm.
Since these matrices are if reduced row echelon form, to calculate ranks of these matrices one needs simply to find the numbers of non-zero rows.
For all $i = 1, \ldots, n$ assign $x_i := a_i^*$.

Step 3. Check if all the completion ratios obtained x_i fit within the constraints $0 \leq x_i \leq 1$. If not – interrupt the algorithm.

End.

Since Gauss-Jordan method uses only elementary row operations, the rank of the matrix is not affected by it: $\operatorname{rank} R = \operatorname{rank} R^*$. Steps 1 and 2 are based upon the Rouche-Capelli (Kronecker-Capelli) theorem. According to it, the system $RX = A$ is consistent and has a unique solution if and only if $\operatorname{rank} R = \operatorname{rank}(R|A) = n$, which is equivalent to $\operatorname{rank} R^* = \operatorname{rank}(R^*|A^*) = n$. This condition also implies that $r \geq n$, which is best to be ascertained at the very beginning of the algorithm in order to save computation time. If the solution to system $RX = A$ is unique, then the augmented matrix $(R^*|A^*)$ produced by Gauss-Jordan method defines a set of n equalities $\forall\, i \in 1, \ldots, n\, x_i = a_i^*$. Step 3 is needed to ascertain that the solution provided by Gauss-Jordan method satisfies the resource constraints.

Proposition: Time complexity of the provided algorithm is at most $O(rn^2)$, where n is the number of tasks in the initial task set and r is the number of resources.

Proof: Examining each step of the algorithm individually, one can make sure that the most time-consuming step is Step 1, at which Gauss-Jordan elimination method is implemented to obtain the reduced row-echelon matrix R^*. The Gauss-Jordan elimination method requires n pivots, each of which requires $O(nr)$ calculations [Nemhauser et al., 1999]. Thus, the time complexity of the algorithm is $O(n^2)$. ■

Note that the algorithm does not provide a solution for any instance of the problem. However, it does demonstrate that the problem is solvable polynomially in at least some of all possible cases.

5 Conclusion

In this paper we've examined the problem of estimating maximum resource load for Resource-Constrained Project Scheduling Problem. We've discussed in detail the case of two resources and presented two polynomial algorithms for solving this case, with time complexity of at most $O(n^2)$. The former allows for relatively easy further modifications, while the latter is easier to implement and was used in designing a pseudopolynomial RCPSP lower bound estimation algorithm. A series of tests was done using PSPLIB benchmark to compare the designed pseudopolynomial algorithm with the best known lower bound estimation algorithms (Tab. 1).

Tasks	Instances	NW BKLB %	MIN R BKLB %	AVG R BKLB %	Bounds improved
30	445	66,5	68,5	96,3	0
60	450	71,4	77,3	97,6	0
90	445	75,3	83,3	98,8	0
120	600	54,7	84,7	98,3	5

Table 1: *Tasks* – number of tasks, *Instances* – number of tested instances, *NW BKLB* – percent of instances, where the obtained lower bound is not worse than the best one, *MIN R BKLB* – minimal ratio of the obtained value to the best known lower bound, *AVG R BKLB* – average ratio of the obtained value to the best known lower bound, *Bounds improved* – number of improved bounds.

The results demonstrate that for 66% of instances the lower bound provided by our algorithm is not worse than the best known value. For all instances, our value of lower bound is at most $31,5\%$ worse than the best known lower bound. Moreover, for 5 instances the best known lower bound was improved. We have also discussed the general case of arbitrary number of resources and described how it can be solved in $O(rn^2)$ if $r \geq n$, full resource consumption is possible and the solution to the corresponding linear equation system is unique.

Acknowledgements

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

References

[Arkhipov et al., 2017] Arkhipov, D., Battaia, O., & Cegarra, J. (2017). Resource load-based makespan estimation algorithm for resource-constrained project scheduling problem. *Liste des articles accepts ROADEF 2017 [73]*, ROADEF 2017 Conference, Metz, France.

[Baburin et al., 2007] Baburin, A. E., & Pyatkin, A. V. (2007). Polynomial algorithms for solving the vector sum problem. *Journal of Applied and Industrial Mathematics, 1(3)*, 268-272.

[Dantzig, 1957] Dantzig, G. (1957). Discrete variable extremum problems. *Operations Research, 5*, 266-277.

[Garey et al., 1975] Garey, M. R., & Johnson, D. S. (1975). Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing 4(4)*, 397-411.

[Kellerer et al., 2004] Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack Problems*. Springer-Verlag Berlin Heidelberg GmbH.

[Nemhauser et al., 1999] Nemhauser, G., & Wolsey, L. (1999). *Integer and Combinatorial Optimization*. New York: John Wiley & Sons, Inc.

[Onn et al., 1999] Onn., S., & Schulman, L. (1999). The Vector Partition Problem for Convex Objective Functions. *Mathematics of Operations Research 26(3)*, 583-590.

[Pisinger, 1997] Pisinger, D. (1997). A minimal algorithm for the 0-1 knapsack problem. *Operations Research, 45(5)*, 758-767.