Discrete Optimization

An efficient pseudo-polynomial algorithm for finding a lower bound on the makespan for the Resource Constrained Project Scheduling Problem[☆]

Dmitry Arkhipov[a,b,*], Olga Battaïa[a], Alexander Lazarev[b,c,d,e]

[a] ISAE-SUPAERO, Université de Toulouse, Toulouse, France
[b] V.A. Trapeznikov Institute of Control Sciences of Russian Academy of Sciences, Moscow, Russia
[c] Lomonosov State University, Moscow, Russia
[d] National Research University Higher School of Economics, Moscow, Russia
[e] Moscow Physical and Technical Institute (State University), Dolgoprudnyi, Russia

Keywords:
Project scheduling
Scheduling
RCPSP
Lower bounds
Pseudo-polynomial algorithms

A B S T R A C T

Several algorithms for finding a lower bound on the makespan for the Resource Constrained Project Scheduling Problem (RCPSP) were proposed in the literature. However, fast computable lower bounds usually do not provide the best estimations and the methods that obtain better bounds are mainly based on the cooperation between linear and constraint programming and therefore are time-consuming. In this paper, a new pseudo-polynomial algorithm is proposed to find a makespan lower bound for RCPSP with time-dependent resource capacities. Its idea is based on a consecutive evaluation of pairs of resources and their cumulated workload. Using the proposed algorithm, several bounds for the PSPLIB benchmark were improved. The results for industrial applications are also presented where the algorithm could provide good bounds even for very large problem instances.

1. Introduction

The Resource Constrained Project Scheduling Problem (RCPSP) is a well-known problem in scheduling theory. This problem is known to be *NP*-hard in the strong sense (Garey & Johnson, 1975). In this research, we consider a generalized statement of the problem with a time-dependent resource capacity function defined as follows. There is a set of tasks N and a set of renewable resources R. The amount of resource $X \in R$ which can be used by tasks of set N during time slot $[t, t+1)$ is defined by *capacity function* $c_X(t)$. The statement of a constant resource capacity is a particular case of this formulation. For any task $j \in N$, the following parameters are given:

- p_j – processing time;
- a_{jX} – required amount of resource $X \in R$.

Precedence relations between tasks are given by directed acyclic graph $G(N, E)$. If an edge $(i \to j) \in E$ exists, it means that task i must be finished before the starting time of task j.

Further, the following parameters can be calculated for each task taking into account the precedence and resource constraints:

- r_j – release time, the earliest time from which task j can be started;
- D_j – deadline, the latest time for finishing task j if a global deadline T for the project is known.

It should be noted that these task parameters can be also given as initial data and incorporated in the algorithm described below.

The objective is to find a *schedule* with the lowest makespan i.e. with the shortest project duration.

A *schedule* π is feasible for the sets of resources R and tasks N, if for any $j \in N$ *starting time* $S_j(\pi) \geq r_j$ is defined and all precedence and resource capacity constraints are satisfied. The set of all feasible schedules is noted by $\Pi(N, R)$. The objective is to find a feasible schedule with the minimal makespan value i.e.

$$\min_{\pi \in \Pi(N,R)} \max_{j \in N} C_j(\pi),$$

where $C_j(\pi) = S_j(\pi) + p_j \leq D_j$ – the *completion time* of task j, where D_j – deadline of task j.

☆ This research is supported by the Russian Foundation for Basic Research (grant 18-37-00295 mol_a) and the Foundation of ISAE-SUPAERO.

* Corresponding author at: ISAE-SUPAERO, Universitéde Toulouse, Toulouse, France.
E-mail addresses: miptrafter@gmail.com (D. Arkhipov), olga.battaia@isae.fr (O. Battaïa), jobmath@mail.ru (A. Lazarev).

https://doi.org/10.1016/j.ejor.2018.11.005

In the literature, a number of algorithms to calculate lower bounds on the makespan were proposed. Their overview is presented in Section 2, more details can be found in the following comprehensive surveys (Neron et al., 2006) and (Knust, 2015). In this paper, a novel pseudo-polynomial algorithm is developed which extends the relaxation of RCPSP to a Cumulative Scheduling Problem by considering pairs of resources. Our approach uses ”time-tabling” techniques to adjust the capacity function of resources first and then it calculates a lower bound on the makespan by evaluating highest possible resource loads for each time slot.

The rest of the paper is organized as follows. The main algorithm for lower bound calculation is presented in Section 3. The results of the numerical experiments are discussed in Section 4. Concluding remarks and research perspectives are given in Section 5.

2. Previous research

There is a large number of publications devoted to the discussion on lower bounds for RCPSP. Recent comprehensive reviews can be found in Neron et al. (2006) and Knust (2015). The analysis of the computational complexity of some algorithms and the quality of the obtained bounds are discussed in Gafarov, Lazarev, and Werner (2010). The majority of efficient algorithms can be referred to as ”destructive” methods. Such an algorithm starts with a defined project deadline T and tries to find a feasible schedule for it. If a feasible solution does not exist, the deadline is increased (usually by incrementing the deadline with 1 unit of time) and the calculation procedure restarts. The calculation continues until the algorithm cannot reveal any contradiction with the defined deadline or until the end of the allocated calculation time. Similarly, constraint programming methods can be applied as for example in Schutt, Feydy, Stuckey, and Wallace (2011) and Laborie (2003).

Here below are shortly discussed the most effective methods for finding a lower bound on the makespan for RCPSP. For more details on these algorithms, the reader can consult the recent comprehensive reviews (Neron et al., 2006) and Knust (2015).

1. *Disjunctive bounds*

In the study of Baptiste and Pape (2000), each renewable resource is considered as a system of several identical processors with the number of processors equal to the capacity of the resource. The problem is formulated using mixed integer programming and heuristics are used to solve it.

Several algorithms (Applegate & Cook, 1991; Baptiste, Pape, & Nuijten, 1999; Carlier & Pinson, 1989; Carlier & Pinson, 1990; Carlier & Pinson, 1994) aim to facilitate the lower bound calculation by constructing complementary precedence relations with the verification of the assumptions that certain requirements of set A must / cannot be satisfied before / after some considered set of requirements B. These algorithms have polynomial runtime for one iteration, but the number of iterations increases exponentially when the number of requirements A and B increases.

2. *Cumulative bounds*

The algorithm of Carlier and Latapie (1991) is based on the identification of such sets of tasks for which the available amount of resource is insufficient for their parallel execution. Each resource is considered as a system of several identical processors where the number of processors per resource is less than its capacity. In the further studies of Carlier and Pinson (1998) and Carlier and Pinson (2004), it is assumed that each processor can execute more than one task per unit of time and the same task can be completed by several processors.

Polynomial time algorithms were proposed to solve a relaxed problem (Brucker, 2002; Haouari & Gharbi, 2003; Tercinet, Lente, & Neron, 2004) where the planning horizon (between the starting point and the deadline) was split into intervals and each resource was considered as a multiprocessor system with the number of processors equal to the capacity of the resource. Further, the interruption of tasks at the boundaries of intervals was allowed.

The relaxation to so called Cumulative Scheduling Problem was also explored. It is obtained from the initial problem by ignoring all resources except one and replacing the precedence relations by release times and deadlines. The optimal makespan for this problem provides a lower bound for the initial problem. However, the obtained Cumulative Scheduling Problem is also NP-hard in the strong sense. Nevertheless, methods developed for the calculation of a lower bound on the makespan for such a formulation provide a lower bound for the makespan of the initial problem as well (Brucker & Knust, 2000; Carlier & Neron, 2000; Carlier & Neron, 2003; Mingozzi, Maniezzo, Ricciardelli, & Bianco, 1998). Satisfiability tests SAT can also be performed by dividing the planning horizon into intervals and checking the amount of the available resource in each of the considered intervals (Baptiste et al., 1999; Erschler, Lopez, & Thuriot, 1991; Lopez, Erschler, & Esquirol, 1992; Schwindt, 2005).

3. *Methods based on Constraint Programming*

Among the studies using different techniques of Constraint Programming, for example Schutt et al. (2011) and Laborie (2003), the techniques developed in Nuijten (1994) and Caseau and Laburthe (1996) are based on reducing the time intervals calculated for each task due to the analysis of the available amount of each resource for a chosen set of tasks. Such algorithms are time consuming because of the large number of possible sets under consideration.

4. *Algorithms based on the exploration of multi-resource constraints*

Such algorithms (Baptiste & Demassey, 2004; Garaix, Artigues, & Demassey, 2005; Laborie, 2005) are based on the research of ”critical sets” (MCS - minimum critical set, FS - forbidden set) i.e. sets of tasks that cannot be performed simultaneously because of resource capacity constraints, while any subset of such a critical set does not violate resource constraints and can be performed simultaneously.

5. *Linear programming relaxations*

A lower bound can also be obtained by a relaxation of the initial problem to a linear programming problem (Christofides, Alvarez-Valdes, & Tamarit, 1987; Pritsker, Watters, & Wolfe., 1969).

A detailed analysis of these algorithms can be found in recent surveys (Neron et al., 2006 and Knust, 2015). The conclusion of this analysis is that fast algorithms usually do not provide the best lower bounds and the methods that obtain better bounds are mainly based on the cooperation between linear and constraint programming and therefore are time-consuming. In this paper, a new pseudo-polynomial algorithm is proposed which aims to provide good lower bounds in acceptable computational time.

3. A novel algorithm for finding a lower bound on the makespan

The considered decision version of RCPSP is formulated as follows:

Problem 1. Given set of tasks N, set of resources R and deadline (time horizon) T, does any feasible schedule $\pi \in \Pi(N, R)$ exist with a makespan inferior or equal to T, i.e.

$$\max_{j \in N} C_j(\pi) \leq T. \tag{1}$$

Without loss of generality, it is assumed that the project can be started at time $t = 0$. We introduce two dummy tasks $0, n+1 \in N$ which represent the start and the end of the project, i.e. $r_0 = r_{n+1} = 0,\ p_0 = p_{n+1} = 0,\ D_0 = D_{n+1} = T$ and for any $j \in N \setminus \{0, n+1\}$ precedences $0 \to j$ and $j \to n+1$ exist.

The general scheme of finding a lower bound on the makespan is based on the four following procedures:

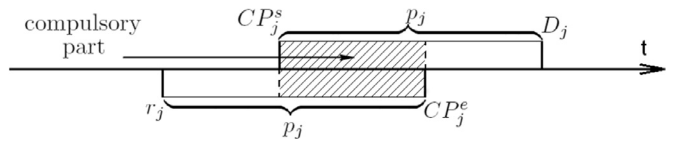

Fig. 1. Compulsory part of a time interval of task j.

Procedure 1. Pre-processing. This procedure updates the release times and deadlines for tasks under condition that the makespan is inferior or equal to T. If during these calculations one of the existing constraints cannot be satisfied, there is no feasible solution with such a bound on the makespan.
Procedure 2. This procedure calculates an upper bound on the resource consumption by set of tasks N during time interval $[0, t+1)$ considering all pairs of resources X, Y. Precedence constraints are replaced by release times and deadlines. In this way, also precedence constraints with time lags can be taken into account.
Procedure 3. Procedure 2 is applied for original precedence graph $G(N, E)$ and the graph with reversed precedence relations $\overline{G}(N, \overline{E})$. The objective is to compare, for any resource $X \in R$, the sum of upper bounds on its amount consumed in intervals $[0, t)$ and $(t, T]$ with the total amount of resource required for all tasks $\sum_{j \in N} a_{jX} p_j$. If the latter is lower than the former, the considered problem is considered infeasible.
Procedure 4. Finally, the binary search part changes time horizon T and then the calculation is restarted.

In the following, each part of the algorithm is discussed in details.

3.1. Procedure 1: pre-processing

We denote the *length of a longest path from* $i \in N$ to $j \in N$ by P_{ij} if there is a path from i to j in graph $G(N, E)$. The calculation of $P_{ij} \geq 0$ for all pairs of tasks $i, j \in N$ having a path from i to j in G can be done using Dijkstra's algorithm (Dijkstra, 1959).

Let us consider all pairs of tasks i, $j \in N$ such that $P_{ij} \geq 0$ and update release times and deadlines using formulae

$$r_j := \max\{r_j, r_i + P_{ij}\},$$

$$D_i := \min\{D_i, D_j - P_{ij}\}.$$

If for any $j \in N$, holds $D_j - r_j < p_j$, then inequality (1) is violated and the algorithm terminates. Otherwise, the compulsory part of the time interval (between the release time and deadline) is calculated for each task $j \in N$ $[CP_j^s, CP_j^e)$ using formulae $CP_j^s = D_j - p_j$, $CP_j^e = r_j + p_j$ (Fig. 1). This idea was formulated in Lahrichi (1982).

If $CP_j^s < CP_j^e$, then under any schedule π, which satisfies given release dates and deadlines, task j consumes exactly a_{jX} of resource $X \in R$ at each moment of time $t \in [CP_j^s, CP_j^e)$. Therefore, the amount of resource $X \in R$ that can be used by other tasks at each moment of time $t \in [CP_j^s, CP_j^e)$ is not more than $c_X(t) - a_{jX}$. This idea leads us to replace capacity function $c_X(t)$ by function $c'_X(t)$ representing the amount of resource $X \in R$ which can be used to perform non-compulsory parts of tasks (Fig. 2)

$$c'_X(t) = c_X(t) - \sum_{j \in N | t \in [CP_j^s, CP_j^e)} a_{jX}.$$

If for any $X \in R$ and $t = 0, \ldots, T-1$ inequality $c'_X(t) < 0$ holds, there is no feasible schedule which satisfies deadline T. The number of breakpoints of function $c'_X(t)$ is not superior to $2n + m$,

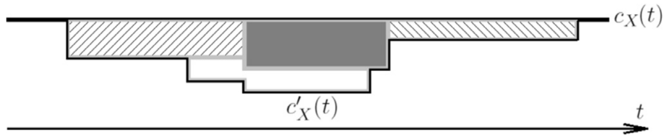

Fig. 2. Amount of resource $X \in R$ which can be used to perform non-compulsory parts of tasks.

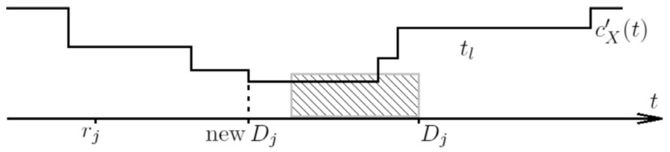

Fig. 3. Update of deadlines.

where m is the number of breakpoints of $c_X(t)$ in horizon T. The complexity of $c'_X(t)$ calculation for all $X \in R$ can be estimated by $O((n+m)r)$ operations, where $n = |N|$ and $r = |R|$. Note that the calculation of $c'_X(t)$ is similar to the Resource profile calculation presented in Fox (1990); Pape (1988).

The idea of the following algorithm is close to sweep algorithms presented in Beldiceanu and Carlsson (2001, 2002); Letort, Beldiceanu, and Carlsson (2012), the difference lies in the utilization of function $c'_X(t)$ which is actively used and dynamically changed in our algorithm.

For each task $j \in N$ and resource $X \in R$, its demand in resource X is compared with the availability of resource X i.e. the value of capacity function $c'_X(t)$ for all m' breakpoints of function $c'_X(t)$ which does not belong to compulsory interval $[CP_j^s, CP_j^e)$. If for any set of breakpoints $t_0, \ldots, t_{m'}$, $c'_X(t) < a_{jX}$ i.e. the amount of resource X is not sufficient to perform task j, the following updates are realized:

- if for any $l \in \{0, \ldots, m'-1\}$ such that $t_l < \max\{CP_j^s, CP_j^e\}$ and $r_j \leq t_{l+1}$ holds $c'_X(t_l) < a_{jX}$, update $r_j := t_{l+1}$;
- if for any $l \in \{1, \ldots, m'\}$ such that $\max\{CP_j^s, CP_j^e\} < t_l$ and $t_{l-1} < D_j$ holds $c'_X(t_{l-1}) < a_{jX}$, update $D_j := t_{l-1}$ (Fig. 3).

If for any task $j \in N$, its release date or deadline is updated, the preprocessing part is restarted with the new values of r_j and D_j. Otherwise, the preprocessing algorithm terminates successfully.

Lemma 1. *The complexity of the preprocessing part is* $O(n^2(n+m)Tr)$ *operations, where n is the number of tasks, m is the highest number of breakpoints of the resource capacity function, T is the time horizon and r is the number of resources.*

Proof. The calculation of P_{ij} for all i, $j \in N$ takes $O(n|E| + n^2 \log n)$ operations, where $|E|$ is the number of edges in graph G. Each iteration of the first round for release and deadline calculation takes $O(n^2)$ operations for checking all paths and $O(n(n+m)r)$ for resource inequalities verification. The number of iterations is no more than nT since each task cannot have more than T release time or deadline updates. Therefore, the total complexity of the preprocessing part can be estimated by $O(n^2(n+m)Tr)$ operations. □

3.2. Inner cycle: relative resource load calculation

Let us consider two resources: X and Y. The earliest possible moment of time when $j \in N$ can start to use resources X, $Y \in R$ is r_j. For any $t \leq p_j$ in time interval $[r_j, r_j + t)$, the amount of resources X and Y consumed by task j cannot be more than $t \cdot a_{jX}$ and $t \cdot a_{jY}$ respectively. If $[CP_j^s, CP_j^e) \neq \emptyset$ task j uses exactly a_{jX} and a_{jY} in each of time slots $[CP_j^s, CP_j^s + 1), \ldots, [CP_j^e - 1, CP_j^e)$.

Let

$$A_{jX}(t) = (\min\{t, CP_j^s, CP_j^e\} - \min\{t, r_j - 1\}) \cdot a_{jX}$$

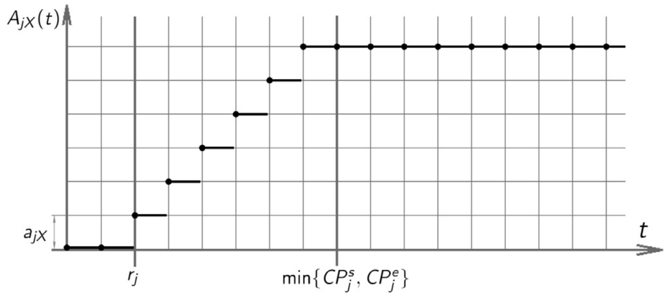

Fig. 4. $A_{jX}(t)$ – the highest possible amount of resource X used by the non-compulsory part of task j in interval $[0, t+1)$.

– be the highest possible amount of resource X used by the non-compulsory part of task j in interval $[0, t+1)$ (Fig. 4).

The inner cycle procedure processes time slot by time slot starting at the moment $t = 0$. In each time slot, the highest possible consumption of resources X and Y by all tasks of set N is estimated by taking into account only non-compulsory parts of tasks. The amount of resource X used by non-compulsory part of task $j \in N$ in interval $[0, t+1)$ is denoted by $u_{jX}(t)$, and the total consumption of resource X by all tasks in interval $[0, t+1)$ is denoted by $U_X(t) = \sum_{j\in N} u_{jX}(t)$.

The main idea behind the developed algorithm is to calculate an upper bound on the possible consumption of resources X and Y taking into account the non-compulsory parts of the tasks that can be assigned to each time interval. In a general case, the demand in resources of all such tasks will be superior to the resource capacity, but not necessarily in the same proportion for resource X as for resource Y. The originality of the proposed approach is to take into account the fixed proportion of usage of different resources by each task. To calculate the upper bound, a linear combination of fractional parts of such tasks that use the highest available amount of both resources is researched. Among different combinations using the totality of the available resources, a geometric algorithm is used to choose the combination for which the validity of the lower bound on the makespan is proven by Theorems 1 and 2. For example, in a general case, this algorithm will prefer the combination of the tasks using both resources to the combination using the tasks requiring only one resource. This is done in order to provide more flexibility in the resource usage for remaining time intervals.

For resources X and Y, the *consumption scheme* φ is defined when non-compulsory used amounts of resources $u_{jX}(t)$ and $u_{jY}(t)$ are known for any task $j \in N$ and time slot $t = 0, \dots, T-1$. The consumption scheme is *valid* if for any task $j \in N$ and time slot $t = 0, \dots, T-1$ the following conditions hold:

$$u_{jX}(t) \le A_{jX}(t),$$

$$u_{jY}(t) \le A_{jY}(t),$$

$$\frac{u_{jX}(t)}{u_{jY}(t)} = \frac{a_{jX}}{a_{jY}}.$$

The first and the second inequalities are associated with the definitions of $A_{jX}(t)$ and $A_{jY}(t)$, respectively. The last equality is very important, since it requires that the proportion of resources X and Y used by task $j \in N$ remains the same in the considered consumption scheme as in any feasible schedule. Note that each feasible schedule with deadline T possesses valid consumption schemes for all resources.

All time slots $t = 0, \dots, T-1$ are considered one by one in an iterative way and for each of them, the following optimization problem is solved:

Problem 2. For each $j \in N$ values $u_{jX}(t-1)$ and $u_{jY}(t-1)$ are given and functions $A_{jX}(t)$, $A_{jX}(t)$ are defined. The objective is to determine $u_{jX}(t) \ge u_{jX}(t-1)$ and $u_{jY}(t) \ge u_{jY}(t-1)$ for all tasks $j \in N$ such that $U_X(t)$ and $U_Y(t)$ reach the highest possible value (since we are interested in an upper bound on resource consumption). The following constraints should be taken into account:

$$\frac{u_{jX}(t) - u_{jX}(t-1)}{u_{jY}(t) - u_{jY}(t-1)} = \frac{a_{jX}}{a_{jY}},$$

$$u_{jX}(t) \le A_{jX}(t), \;\; u_{jY}(t) \le A_{jY}(t),$$

$$\sum_{j\in N} (u_{jX}(t) - u_{jX}(t-1)) \le c'_X(t),$$

$$\sum_{j\in N} (u_{jY}(t) - u_{jY}(t-1)) \le c'_Y(t).$$

If for any time slot there is more than one solution which satisfy these conditions, the solution will be chosen using the following criterion:

$$\min \sum_{j\in N} \sqrt{(u_{jX}(t) - u_{jX}(t-1))^2 + (u_{jY}(t) - u_{jY}(t-1))^2}. \qquad (2)$$

The necessity of this criterion is explained by Theorem 1. This problem can be reformulated in terms of vectors.

Problem 3. For time slot t we have a set of two-dimensional vectors $v_1 = (A_{1X}(t) - u_{1X}(t-1), A_{1Y}(t) - u_{1Y}(t-1)), \dots, v_n = (A_{nX}(t) - u_{nX}(t-1), A_{nY}(t) - u_{nY}(t-1))$ associated with all tasks of set N. The objective is to find a set of coefficients $\{\alpha_1, \dots, \alpha_n\} \in [0, 1]$ such that the linear combination

$$L = \alpha_1 v_1 + \dots + \alpha_n v_n$$

has the highest possible projections on the axes (it corresponds to the highest usage of the resources) and satisfies the inequalities $L_X \le c'_X(t)$ and $L_Y \le c'_Y(t)$. If there is more than one solution, choose the one with the lowest sum of the vectors lengths i.e. (it corresponds to criterion 2):

$$\min \sum_{j\in N} \alpha_j |v_j|. \qquad (3)$$

Lemma 2. *Problems 2 and 3 are equivalent.*

Proof. In problem 2 we have to find $u_{jX}(t)$, $u_{jY}(t)$ such that $A_{jX}(t) \ge u_{jX}(t) \ge u_{jX}(t-1)$, $A_{jY}(t) \ge u_{jY}(t) \ge u_{jY}(t-1)$ and

$$\frac{u_{jX}(t) - u_{jX}(t-1)}{u_{jY}(t) - u_{jY}(t-1)} = \frac{a_{jX}}{a_{jY}}.$$

Since values $u_{jX}(t-1)$, $u_{jY}(t-1)$, $A_{jX}(t)$ and $A_{jY}(t)$ are given, each pair of values $u_{jX}(t)$, $u_{jY}(t)$ can be associated with a vector $v_j = (u_{jX}(t) - u_{jX}(t-1),\; u_{jY}(t) - u_{jY}(t-1))$ where $u_{jX}(t) = u_{jX}(t-1) + \alpha_j(A_{jX}(t) - u_{jX}(t-1))$ and $u_{jY}(t) = u_{jY}(t-1) + \alpha_j(A_{jY}(t) - u_{jY}(t-1))$, $\alpha_j \in [0,\; 1]$. Therefore linear combination $L = \alpha_1 v_1 + \dots + \alpha_n v_n$ has projections

$$L_X = \sum_{j\in N} \alpha_j (u_{jX}(t) - u_{jX}(t-1)) = U_X(t) - U_X(t-1),$$

$$L_Y = \sum_{j\in N} \alpha_j (u_{jY}(t) - u_{jY}(t-1)) = U_Y(t) - U_Y(t-1)$$

on axes OX and OY respectively. Since $U_X(t-1)$ and $U_Y(t-1)$ are fixed, the highest possible values of $U_X(t)$ and $U_Y(t)$ correspond to the highest values of L_X and L_Y (the highest usage of the resources). If there is more than one linear combination which satisfies the

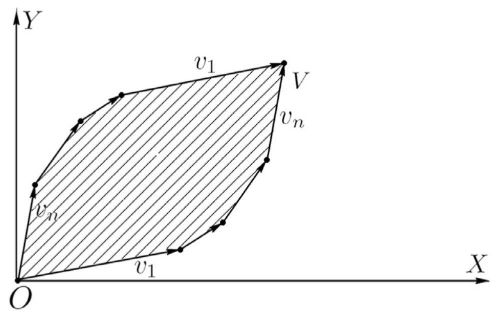

Fig. 5. Polygon construction.

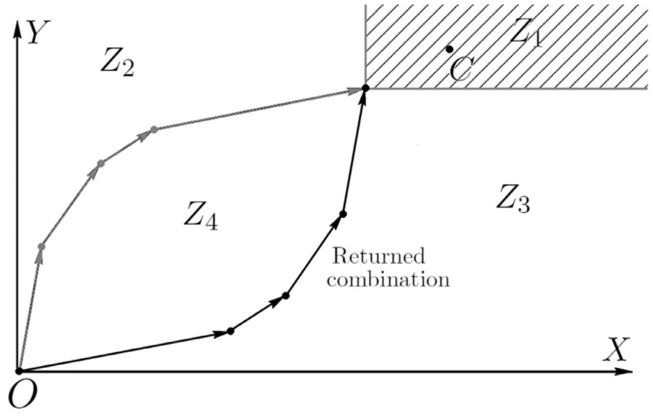

Fig. 6. Geometric algorithm: subcase 2a.

above conditions, the second objective (3) is applied to choose the solution. Note, that

$$\sum_{j\in N}\alpha_j|v_j| = \sum_{j\in N}\sqrt{(u_{jX}(t)-u_{jX}(t-1))^2+(u_{jY}(t)-u_{jY}(t-1))^2},$$

hence (3) is equivalent to (2). □

The following geometric algorithm is designed to solve optimally Problem 3.

1. Construct the convex centrally symmetric polygon of possible linear combinations of vectors $v_1,\ldots,v_n$ with coefficients in [0,1] as follows. Let $OV = v_1+\cdots+v_n$. The upper and lower borders of this polygon are associated with the sequences of vectors placed in descending and ascending orders of tangents of the angle formed with the abscissa axis (Fig. 5). Further, it is assumed that these vectors are already sorted in ascending order of tangents.
2. Consider point $C(c'_X(t), c'_Y(t))$. If C is outside the polygon, three following subcases are possible.
   (a) C belongs to zone Z1, i.e. $C_X \geq V_X$, $C_Y \geq V_Y$. The procedure returns $\alpha_j = 1$ for each $j \in N$ (Fig. 6).
   (b) C belongs to zone Z2, i.e. $C_Y < V_Y$ and the projection of C on the axis of ordinates intersects the polygon. The procedure returns a set of coefficients α_j, such as $\sum_{j\in N}\alpha_j v_j$ corresponds to the rightmost intersection of polygon and $Y = c'_Y(t)$ (Fig. 7).
   (c) C belongs to zone Z3, i.e. $C_X < V_X$ and the projection of C on the axe of abscissa intersects the polygon. The procedure returns a set of coefficients α_j, such as $\sum_{j\in N}\alpha_j v_j$ corresponds to the highest intersection of polygon and $X = c'_X(t)$ (Fig. 8).
3. If point C is inside the polygon (zone Z4), we make a translation of the lower border on vector $OC(c'_X(t), c'_Y(t))$ and find the set of coefficients $\{\beta_1,\ldots,\beta_n\}$ which defines the path from point C to V (dashed line in Fig. 9), the translated lower border and the borders of the initial polygon, i.e. $\beta_1 v_1+\cdots+\beta_n v_n = OV - OC$. Then the procedure returns the set of coefficients $\{1-\beta_1,\ldots,1-\beta_n\}$ that corresponds to a polyline which is shown on Fig. 9.

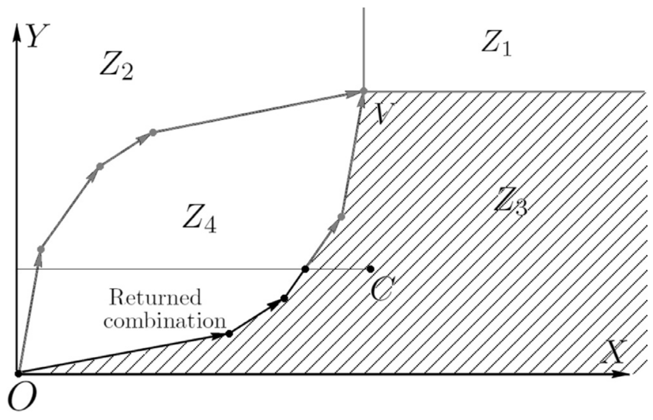

Fig. 7. Geometric algorithm: subcase 2b.

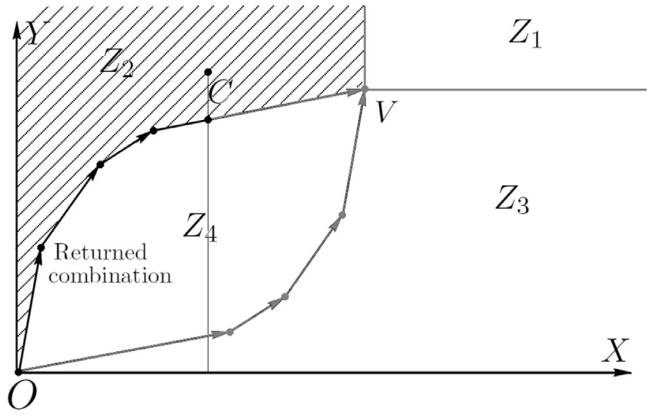

Fig. 8. Geometric algorithm: subcase 2c.

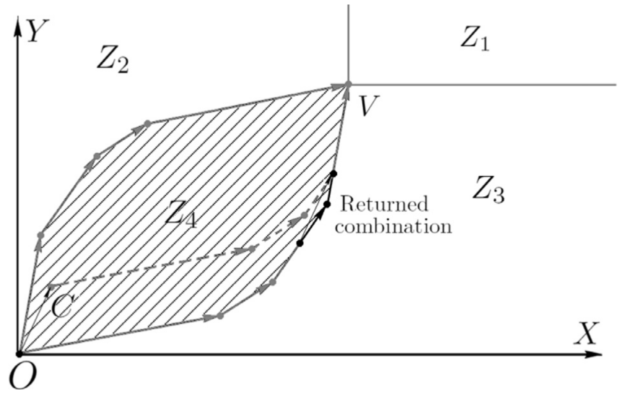

Fig. 9. Geometric algorithm: step 3.

The following lemma is required to prove the correctness of the geometric algorithm.

Lemma 3. *Let two sets of vectors $A = \{v_1^A,\ldots,v_l^A\}$ and $B = \{v_1^B,\ldots,v_k^B\}$ such that $\sum_{j=1}^{l} v_j^A = \sum_{j=1}^{k} v_j^B$ and the polygon associated with A be totally included into the polygon associated with B (Fig. 10). Then the total length of vectors of set A is not superior to the total*

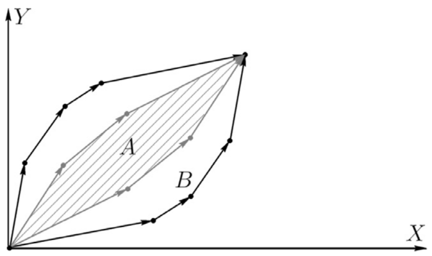

Fig. 10. Lemma 3. Polygon associated with A be totally included into the polygon associated with B.

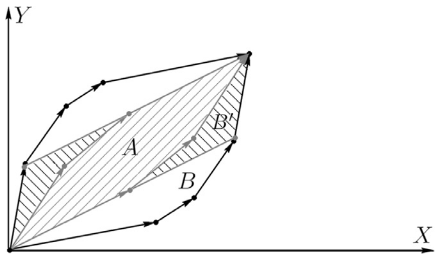

Fig. 11. Proof of Lemma 3.

length of vectors of set B, i.e.

$$\sum_{j=1}^{l} |v_j^A| \le \sum_{j=1}^{k} |v_j^B|.$$

If $A \neq B$, then the inequality is strict.

Proof. Let us compare polygons A and B vector by vector. If we find a difference on vector v_i, let us do an additional construction as on Fig. 11, extending vector v_i to the intersection with the polygon B. Then, let us make a centrally symmetric construction for upper border vectors to obtain new polygon B', such as $A \subset B' \subset B$. Then, let us replace polygon B by polygon B'. Obviously, such a change reduces the perimeter. Repeating it no more times than the number of edges of polygon A, we obtain two identical polygons. □

Lemma 4. *The proposed geometric algorithm finds an optimal solution for Problem 3 in $O(n^2)$ operations.*

Proof. Let us show that in each case algorithm finds the set of vectors which sum has the greatest possible projections on the axes (which correspond to the highest possible consumptions of the available resources). If point C is outside the polygon, then the coefficients associated either with the initial set of vectors (2a) or with the intersection of line $Y = C_Y$ with a lower border line of the polygon (2b) or with the intersection of line $X = C_X$ with an upper border line of the polygon (2c) is returned. All this points have the highest possible coordinates among all points of the polygon, which coordinates are not higher than C_X and C_Y. If C lays inside the polygon, then algorithm returns the set of vectors which sum has the coordinates (C_X, C_Y).

Now let us show that the obtained set of vectors $\{\alpha_1 v_1, \dots, \alpha_n v_n\}$ has the shortest total length among all those with the maximal sum of the coordinates. In the cases where point C is located in zones Z1, Z2 and Z3, there is only one solution which satisfies this condition. When C lies in zone Z4, the algorithm finds the set of vectors $\{\beta_1 v_1, \dots, \beta_n v_n\}$ that corresponds to polyline CV with the longest possible length. Finally, the algorithm returns the set of coefficients $\{\alpha_1, \dots, \alpha_n\} = \{(1-\beta_1), \dots, (1-\beta_n)\}$. Since

$$\sum_{j\in N} |\alpha_j v_j| = \sum_{j\in N} |v_j| - \sum_{j\in N} |\beta_j v_j|,$$

set of vectors $\{\alpha_1 v_1, \dots, \alpha_n v_n\}$ has the shortest possible sum of vector lengths. Therefore obtained solution is optimal with respect to criterion 3. Thus Lemma 4 is verified.

The greatest number of operations is required for the case when C lies in zone $Z4$. In this case, a lower border translation is required, the intersection point with the polygon border line can be found in $O(n^2)$ operations. □

The following lemmas should be proved ahead Theorem 1.

Lemma 5. *Let us have a set of two-dimensional vectors $A = \{v_1, \dots, v_m\}$ placed in tangents ascending order and a point V which belongs to the polygon associated with A. Suppose that $A' = \{\alpha_1 v_1, \dots, \alpha_m v_m\}$ is a set of vectors, such that $\forall j = 1, \dots, m : \alpha_j \in [0, 1]$, $\sum_{j=1}^{m} \alpha_m v_m = OV$ and the sum of vector lengths $\sum_{j=1}^{m} \alpha_m |v_m|$ is minimal. Then, for any set of vectors $B = \{\beta_1 v_1, \dots, \beta_m v_m\}$, such that $\beta \in [0, 1]$ and*

$$\sum_{\beta_j v_j \in B} \beta_j v_j = \sum_{\alpha_j v_j \in A'} \alpha_j v_j = OV,$$

the polygon associated with A' belongs to the polygon associated with B.

Proof. Let us assume the contrary. Suppose that there is a set of vectors B which satisfies the Lemma's conditions but the polygon associated with A' does not belong to the polygon associated with B. Lemma 3 implies that the polygon associated with B cannot fully belong to the polygon associated with A'. Therefore, we have to deal only with the situation where the considered polygons are intersected. Hence, polygons' lower border lines have at least four intersection points including O and V. Let us take a look at two consecutive intersection points K and L, such that lower border segment $KL^{A'}$ lies under KL^B. Since both polylines $OV^{A'}$ and OV^B consist of vectors placed in the ascending order of tangents, the vectors which constitute polyline KL^B cannot belong to the set of vectors which constitute $OK^{A'}$ and $LV^{A'}$. Hence, we can replace $KL^{A'}$ by LB^B and thus decrease the perimeter of the polygon associated with A'. This violates the assumption that the sum of vectors lengths of A' is minimal. Lemma 5 is proved. □

Lemma 6. *Suppose that there are two sets of two-dimensional vectors $A = \{v_1^A, \dots, v_m^A\}$ and $B = \{v_1^B, \dots, v_k^B\}$ such that $\sum_{j\in A} v_j^A = \sum_{j\in B} v_j^B$ and the polygon associated with set A is totally included in the polygon associated with set B. Therefore, there is a set of coefficients $\alpha_1^1, \dots, \alpha_k^1, \dots, \alpha_1^m, \dots, \alpha_k^m \in [0, 1]$, which satisfies the following:*

$$\sum_{j=1}^{m} \alpha_1^j = 1,$$

$$\dots$$

$$\sum_{j=1}^{m} \alpha_k^j = 1,$$

$$\sum_{i=1}^{k} \alpha_i^1 v_k^B = v_1^A,$$

$$\dots$$

$$\sum_{i=1}^{k} \alpha_i^m v_k^B = v_m^A.$$

Proof. Let us find coefficients $\alpha_1^1, \dots, \alpha_k^1$ explicitly, using the graphic approach described in Figs. 12–14. The polygon associated with A is totally included in the polygon associated with B', which is included in the polygon associated with B.

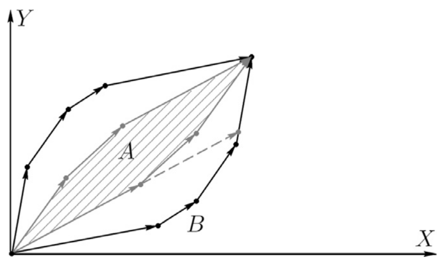

Fig. 12. Lemma 6.

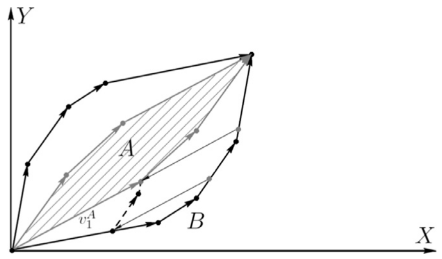

Fig. 13. Finding $v_1^A = \sum_{i=1}^k \alpha_i^1 v_k^B$.

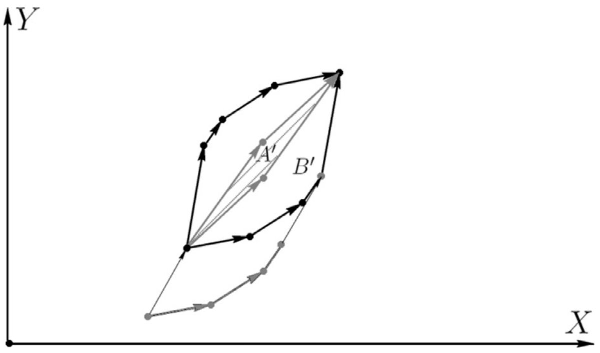

Fig. 14. New polygons A' and B'.

Therefore the polygons associated with sets $A' = A \setminus \{v_1^A\}$ and $B' = \{v_1^B - \alpha_1^1 v_1^B, \ldots, v_k^B - \alpha_k^1 v_k^B\}$ satisfy the initial conditions of Lemma 6. We can iterate this procedure to find all required sets of coefficients which correspond to all vectors of set A. □

The presented geometric algorithm considers the time slots one by one in an iterative way. At each step, an optimal solution for Problem 2 is found for each pair of resources X and Y. Let $U_{X|Y}(t)$ and $U_{Y|X}(t)$ be respectively the amounts of resources X and Y used by set of tasks N in time interval $[0, t+1)$. The following theorem proves that $U_{X|Y}(t)$ and $U_{Y|X}(t)$ provide upper bounds on the consumption of resources X and Y during time interval $[0, t+1)$.

Theorem 1. *Under any valid consumption scheme, the amount of resources X and Y consumed in interval* $[0, t+1)$ *is not more than*

$$U_{X|Y}(t) + \sum_{t'=0}^{t} (c_X(t') - c'_X(t'))$$

and

$$U_{Y|X}(t) + \sum_{t'=0}^{t} (c_Y(t') - c'_Y(t'))$$

respectively.

Proof. Assume the contrary. Suppose that there is a consumption scheme φ^* which violates the initial assumption and uses more than $U_{X|Y}(t) + \sum_{t'=0}^{t}(c_X(t) - c'_X(t))$ resource X in time interval $[0, t+1)$. If there is more than one of such schemes, consider

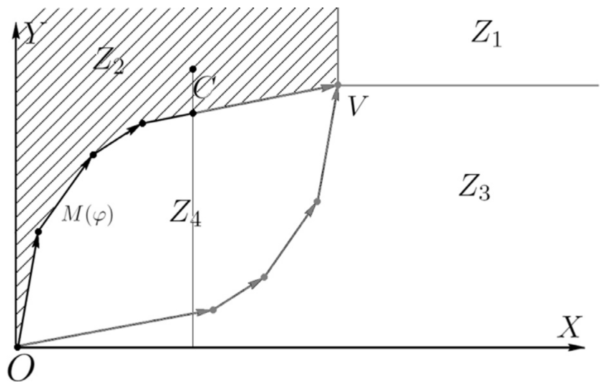

Fig. 15. The highest possible consumption subject to $C \in Z_2$.

the one which uses the highest total amount of resources X and Y in time interval $[0, t+1)$. Let $u^*_{jX}(t)$ be the amount of resource X used by task j under consumption scheme φ^* in time interval $[0, t+1)$.

Let us consequently consider the resource consumption at $u^*_{jX}(t')$ for $t' = 0, \ldots, t$. Suppose t' is the first moment of time which satisfies $u^*_{jX}(t') \neq u_{jX}(t')$ or $u^*_{jY}(t') \neq u_{jY}(t')$ for some $j \in N$. We consider polygon OV associated with the set of vectors $v_j = (A_{jX}(t') - u_{jX}(t'-1), A_{jY}(t') - u_{jY}(t'-1))$ corresponding to the consumptions of resources by each task $j \in N$. The vertex of OV with the highest coordinates X and Y is denoted by V. We also consider point $C(c'_X(t), c'_Y(t))$.

There are four possible cases of positioning C in zones Z_1, Z_2, Z_3, Z_4 in relation to polygon OV.

1. $C \in Z_1$. Each resource cannot be totally used, i.e. $(\sum_{j\in N} v_j)_X < c'_X(t')$ and $(\sum_{j\in N} v_j)_Y < c'_Y(t')$. In this case, for any $j \in N$, the following conditions hold $u_{jX}(t') = A_{jX}(t')$ and $u_{jY}(t') = A_{jY}(t')$. Therefore, we can change the consumption during time slot $[t', t'+1)$ for φ^* using the full amounts of resources as well as under φ without violation of any assumption.
2. $C \in Z_2$. Resource Y cannot be totally used. If under φ^* not full amount of resource X is used, we can use it by one of task $j \in N$ which $a_{jX} > 0$ and $u_{jX}(t') < A_{jX}(t')$. Such a change will not decrease the values of functions $U^*_{jY}(t)$ and $U^*_{jX}(t)$ for any t. Therefore, we can consider only the case where $U^*_{jX}(t') - U^*_{jX}(t'-1) = c'_X(t')$. Note that the highest resource consumption can be achieved only by using a linear combination of vectors with the highest possible ratio $\frac{a_{jY}}{a_{jX}}$ (Fig. 15). Hence, if there is a difference in resource consumption between φ and φ^*, then there is a task $j \in N$ such that $u_{jY}(t') > u^*_{jY}(t')$ and there is a task $i \in N$ which holds $u_{iY}(t') < u^*_{iY}(t')$ and $\frac{a_{jY}}{a_{jX}} < \frac{a_{iY}}{a_{iX}}$. This means that we can replace the part of task i used in time slot $[t', t'+1)$ by j under φ^* without violation of any constraint. Such a change will not decrease the values of functions $U^*_{jY}(t)$ and $U^*_{jX}(t)$ for any t. Let us apply the same changes until $u_{jY}(t') = u^*_{jY}(t')$ does not hold for any $j \in N$.
3. $C \in Z_3$. This case is similar to the previous one. We only need to swap resources X and Y in the description of case 2.
4. $C \in Z_4$. This means that under φ the full capacities of both resources can be achieved.
   Suppose that under φ^* full capacities of resources X and Y are achieved in time slot $[t', t'+1)$. According to Lemma 4 we obtain that the polygon related to the resource consumption in $[t', t'+1)$ under φ has the lowest perimeter of all polygons related to the highest consumption of resources X and Y. Lemma 5 implies that it is totally included in the polygon

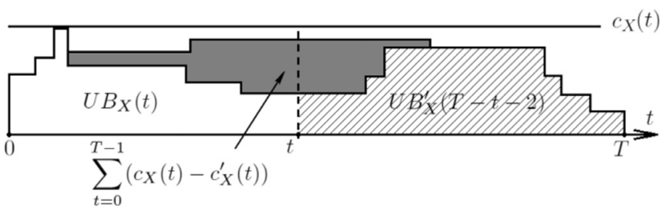

Fig. 16. Master algorithm.

related to φ^*. Therefore, we can change resource consumption under φ^* in time slot $[t', t'+1)$ to the consumption used under φ by taking required parts of $\alpha_j v_j$ from the future timeslots of interval $[t'+1, T)$. Lemma 6 implies that it is possible to replace correctly all parts of vectors $v_1, \ldots, v_n$ used for this procedure by linear combinations of vectors $v_1^*, \ldots, v_n^*$, which were used in φ^* previously. Thus, we can make a change in φ^* without violating the conditions of the Theorem and we obtain equal consumptions of φ^* and φ for time slot $[t', t'+1)$ without increasing or decreasing any amount of resources X and Y being used in any time slot $[t'+1, t'+2), \ldots, [t, t+1)$. For the case where full capacities of resources X and Y are not achieved together in time slot $[t', t'+1)$ under φ^*, the consumption scheme φ^* is modified similarly.

Depending on the case we face in time slot $[t', t'+1)$ we apply the procedure which does not decrease the values of functions $U_X(t)$ or $U_Y(t)$. After having been proceeded with all time slots, we obtain $u_{jX}(t) = u^*_{jX}(t)$ and $u_{jY}(t) = u^*_{jY}(t)$ for any $j \in N$ and $t = 0, \ldots, T-1$. □

3.3. Main cycle: master algorithm

The master part of our algorithm uses Procedure 2 for $G(N, E)$ and the graph with reversed precedence relations $\overline{G}(N, \overline{E})$ to compare, for any resource $X \in R$, a sum of upper bounds on its possible consumed amount in intervals $[0, t)$ and $(t, T]$ with the total amount of resource required for all tasks $\sum_{j \in N} a_{jX} p_j$. If the latter is lower that the former, the considered problem is considered infeasible for time horizon T.

Then, this verification is made for all moments of times $t = 0, 1, 2, \ldots, T-1$ for all pairs of resources $X, Y \in R$, for which functions $U_{X|Y}(t)$ and $U_{Y|X}(t)$ are calculated. Each feasible schedule defines a valid consumption scheme. Theorem 1 implies that for each resource $X \in R$ and any t, an upper bound of the consumption of resource X by tasks in non-compulsory parts of time interval $[0, t+1)$ under any valid consumption scheme can be estimated by function

$$UB_X(t) = \min_{Y \in R} U_{X|Y}(t).$$

Further, the same procedures, including preprocessing, are applied to set of tasks N but for the graph with reversed precedence relations $\overline{G}(N, \overline{E})$, the values of functions $U'_{X|Y}(t)$ are calculated. As a result, for each resource $X \in R$ we obtain a function

$$UB'_X(t) = \min_{Y \in R} U'_{X|Y}(t)$$

which is an upper bound on the consumption of resource X in non-compulsory parts of time interval $(T-t-1, T]$ for the tasks of set N.

After that, the algorithm repeats the same cycle on all moments of time $t = 1, \ldots, T-1$ to check if for any resource $X \in R$ a sum of upper bounds on its available capacity in intervals $[0, t+1)$ and $(t+1, T]$ (Fig. 16) is not lower than the sum of the demands in this resource by all tasks, i.e.

$$\sum_{j \in N} a_{jX} p_j \geq UB_X(t) + UB'_X(T-t-2) + \sum_{t=0}^{T-1} (c_X(t) - c'_X(t)).$$

If this condition is violated, then the problem is infeasible for time horizon T.

Theorem 2. *Suppose that the master algorithm was used for set of tasks N, set of resources R and time horizon T. If for any $X \in R$ and $t = 0, \ldots, T-1$, the following inequality does not hold:*

$$\sum_{j \in N} a_{jX} p_j \leq UB_X(t) + UB'_X(T-t-2) + \sum_{t=0}^{T-1} (c_X(t) - c'_X(t)), \qquad (4)$$

then there is no feasible schedule with makespan inferior or equal to T.

Proof: According to Theorem 1 we obtain that for any feasible schedule $\pi \in \Pi(N, R)$, the amount of resource X used by tasks in time interval $[0, t+1)$ does not exceed $UB_X(t) + \sum_{t'=0}^{t} (c_X(t') - c'_X(t'))$. The amount of resource X used in non-compulsory parts of interval $(t+1, T]$ for tasks does not exceed $UB'_X(T-t-2) + \sum_{t'=t+1}^{T-1} (c_X(t') - c'_X(t'))$. Therefore, taking into account compulsory parts for any feasible schedule π, the amount of resource X used in horizon $[0, T]$ does not exceed

$$UB_X(t) + UB'_X(T-t-2) + \sum_{t=0}^{T-1} (c_X(t) - c'_X(t)).$$

If inequality (4) is violated, then for each feasible schedule $\pi \in \Pi(N, R)$ the amount of resource X required for processing all tasks of set N cannot be used during time interval $[0, T]$. This proves the statement of the Theorem.

3.4. Binary search

In this part, a simple binary search is used to find the highest possible value of the time horizon T which satisfies the conditions in Theorem 2.

Theorem 3. *The developed algorithm finds a lower bound on the makespan in $O(n^2 r(n+m+r) T \log T)$ operations, where n is the number of tasks, T is the time horizon, r is the number of resources, m is the highest number of breakpoints of the capacity function of one resource.*

Proof. The number of bi-section search iterations can be estimated by $O(\log T)$ operations. At each iteration, the preprocessing takes $O(n^2(n+m)rT)$ operations. The master part takes $O(n^2 T)$ operations for each pair of resources. Number of pairs of resources is $O(r^2)$. Therefore, the total complexity of the algorithm can be estimated by $O(n^2 r(n+m+r) T \log T)$ operations. □

4. Numerical experiments

The algorithm was implemented in C++. Two series of numerical experiments were carried out using Intel Core i7 2.8 gigahertz CPU with 16 gigabytes RAM. In the first one, the algorithm was tested on the well-known PSPLIB benchmark (Kolisch & Sprecher, 1997). In the second one, the algorithm was applied to large-scaled RCPSP instances based on real data provided by Kuznetsov Design Bureau. The results of the tests are presented in Tables 1 and 2, respectively.

The first series of tests was performed for the problem instances from PSPLIB benchmark. The objective was to compare the results provided by our approach with the best known lower bounds (BKLB), presented at PSPLIB website (consulted in July 2017). The results are given in Table 1). They show that for 66% of

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, *CPU time* – highest computation time of one instance.

Tasks	Instances	NW BKLB %	MIN R BKLB %	AVG R BKLB %	Bounds improved	CPU time (seconds)
30	445	66,5	68,5	96,3	0	1
60	450	71,4	77,3	97,6	0	1
90	445	75,3	83,3	98,8	0	2
120	600	54,7	84,7	98,3	4	5

Table 2
tasks – number of tasks, *precedences* – number of precedence relations, *critical path* – critical path-based lower bound, *obtained LB* – obtained lower bound, *R CP %* – ratio of the obtained lower bound to critical path, *CPU time* – time of algorithm performance.

Tasks	Precedences	Critical path	Obtained LB	R CP %	CPU time
444	829	830	887	106,9	5 minutes 30 seconds
888	1658	830	1130	136,1	13 minutes 25 seconds
1332	2487	830	1482	178,6	30 minutes 17 seconds
1776	3316	830	1844	222,1	64 minutes 59 seconds
2220	4145	830	2208	266,0	108 minutes 30 seconds

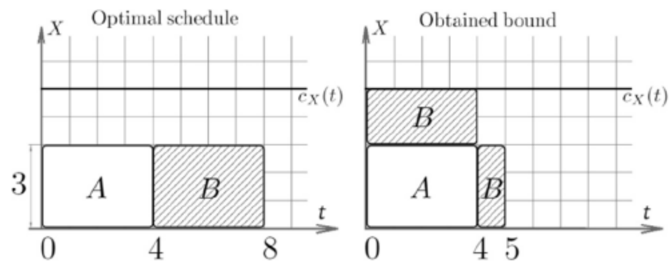

Fig. 17. Bad instance.

the instances the bound calculated with our algorithm is not worse than the best known lower bound. For all instances, our value of lower bound is not more than 31, 5% worse than the best known lower bound, average deviation is about 2%. Moreover, for 4 instances the best known lower bound was improved. It should be noted that the computational time was very short.

The second series of tests was realized on real industrial data in order to evaluate the possibility to apply the presented approach to real large-scaled instances. The tested instances were created by an arbitrary combination of different projects each of which consisting of 444 tasks and 829 precedence relations. There were 46 different resources required to process each project. Numerical results and a comparison with the project's critical path value are reported in Table 2.

4.1. Bad instances

Very bad algorithm performance ($LB \approx \frac{C^*_{\max}}{2}$) can be achieved on the instances where under the optimal schedule the resource capacity is poorly used. A simple example of such a "bad instance" is presented in Fig. 17. For this instance, the obtained lower bound is equal to

$$LB = \frac{C^*_{\max}}{2} + 1 = 5,$$

where the optimal makespan value is equal to $C^*_{\max} = 8$.

5. Conclusion and future perspectives

In this paper, a novel pseudo polynomial algorithm for finding a lower bound on the makespan for RCPSP was presented. This algorithm can be applied to a generalized statement of RCPSP with discrete time where each resource capacity is defined by a non-negative arbitrary function. The main idea of the lower bound calculation is based on a consecutive evaluation of pairs of resources and their cumulated workload. Numerical experiments show that this algorithm provides good results for classical benchmark instances and it can be used for calculating lower bounds for large-scaled problem instances in reasonable time.

In future extension of the algorithm, more generalizations, such as time-dependent resource consumptions of tasks and non-renewable resources will be considered. Another part of the work to be done concerns the theoretical estimation of the quality of the obtained bound and the analysis of bad instances.

References

Applegate, D., & Cook, W. (1991). A computational study of job-shop scheduling. *ORSA Journal on Computing, 3*(2), 149–156.
Baptiste, P., & Demassey, S. (2004). Tight LP bounds for resource constrained project scheduling. *OR Spectrum, 26*, 251–262.
Baptiste, P., & Pape, C. L. (2000). Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints, 5*, 119–139.
Baptiste, P., Pape, C. L., & Nuijten, W. (1999). Satisfiability tests and timebound adjustments for cumulative scheduling problems. *Annals of Operations Research, 92*, 305–333.
Beldiceanu, N., & Carlsson, M. (2001). Sweep as a generic pruning technique applied to the non-overlapping rectangles constraint. In *Proceedings of the seventh international conference on principles and practice of constraint programming* (pp. 377–391).
Beldiceanu, N., & Carlsson, M. (2002). A new multi-resource cumulatives constraint with negative heights. In *Proceedings of the eighth international conference on principles and practice of constraint programming* (pp. 63–79).
Brucker, P. (2002). *Scheduling algorithms*. Berlin: Springer Verlag.
Brucker, P., & Knust, S. (2000). A linear programming and constraint propagation based lower bound for the RCPSP. *European Journal of Operational Research, 127*, 355–362.
Carlier, J., & Latapie, B. (1991). Une methode arborescente pour resoudre les problemes cumulatifs. *RAIRO-Recherche Operationnelle, 25*, 311–340.
Carlier, J., & Neron, E. (2000). A new LP based lower bound for the cumulative scheduling problem. *European Journal of Operational Research, 127*(2), 363–382.
Carlier, J., & Neron, E. (2003). On linear lower bounds for the resource constrained project scheduling problem. *European Journal of Operational Research, 149*, 314–324.
Carlier, J., & Pinson, E. (1989). An algorithm for solving the job-shop problem. *Management Science, 35*, 164–176.
Carlier, J., & Pinson, E. (1990). A practical use of Jackson's preemptive schedule for solving the job-shop problem. *Annals of Operations Research, 26*, 269–287.
Carlier, J., & Pinson, E. (1994). Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research, 78*, 146–161.
Carlier, J., & Pinson, E. (1998). Jackson's pseudo preemptive schedule for the $pm/r_i, q_i/c_{max}$ scheduling problem. *Annals of Operations Research, 83*, 41–48.

Carlier, J., & Pinson, E. (2004). Jackson's pseudo preemptive schedule and cumulative scheduling problems. *Discrete Applied Mathematics, 145*, 80–94.

Caseau, Y., & Laburthe, F. (1996). Cumulative scheduling with task intervals. In *Proceedings of the joint international conference and symposium on logic programming* (pp. 363–377). The MIT Press, Cambridge.

Christofides, N., Alvarez-Valdes, R., & Tamarit, J. M. (1987). Project scheduling with resource constraints: a branch and bound approach. *European Journal of Operational Research, 29*(3), 262–273.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik, 1*, 269–271.

Erschler, J., Lopez, P., & Thuriot, C. (1991). Raisonnement temporel sous contraintes de ressources et problemes d'ordonnancement. *Revue d'Intelligence Artificielle, 5*, 7–32.

Fox, B. R. (1990). Non-chronological scheduling. In *Proceedings of AI, Simulation and Planning in High Autonomy Systems* (pp. 72–77).

Gafarov, E., Lazarev, A., & Werner, F. (2010). On Lower and Upper Bounds for the Resource-Constrained Project Scheduling Problem. Technical report. Otto-von-Guericke Universitaet., 27.

Garaix, T., Artigues, C., & Demassey, S. (2005). Bornes inferieures et superieures pour le RCPSP. In *Proceedings of 6ieme congres de la societe francaise de recherche operationnelle et d'aide a la decision,Tours, France* (pp. 219–240).

Garey, M., & Johnson, D. (1975). Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing, 4*(4), 397–411.

Haouari, M., & Gharbi, A. (2003). An improved max-flow-based lower bound for minimizing maximum lateness on identical parallel machines. *Operations Research Letters, 31*, 49–52.

Knust, S. (2015). Lower bounds on the minimum project duration. In C. Schwindt, & J. Zimmermann (Eds.), *Handbook on project management and scheduling* (pp. 43–56). Springer. Edited by, Vol. 1, Chap. 3.

Kolisch, R., & Sprecher, A. (1997). PSPLIB - a project scheduling problem library: OR software - ORSEP operations research software exchange program. *European Journal of Operational Research, 96*(1), 205–216. http://www.om-db.wi.tum.de/psplib/.

Laborie, P. (2003). Algorithms for propagation of resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence, 143*, 151–188.

Laborie, P. (2005). Complete MCS-based search: Application to resource constrained project scheduling. In *Proceedings of the international joint conference on artificial intelligence*.

Lahrichi, A. (1982). Ordonnancements. la notion de "parties obligatoires" et son application aux problemes cumulatifs. *RAIRO-Operations Research, 16*, 241–262.

Letort, A., Beldiceanu, N., & Carlsson, M. (2012). A scalable sweep algorithm for the cumulative constraint. In *Proceedings of the 18th international conference on principles and practice of constraint programming* (pp. 439–454).

Lopez, P., Erschler, J., & Esquirol, P. (1992). Ordonnancement de taches sous contraintes: une approche energetique. *Automatique-productique informatique industrielle, 26*(5–6), 453–481.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., & Bianco, L. (1998). An exact algorithm for the multiple resource-constrained project scheduling problem based on a new mathematical formulation. *Management Science, 44*, 714–729.

Neron, E., Artigues, C., Baptiste, P., Carlier, J., Damay, J., Demassey, S., & Laborie, P. (2006). Lower bounds for resource constrained project scheduling problem. In J. Jozefowska, & J. Weglarz (Eds.), *Perspectives in modern project scheduling* (pp. 167–204). Springer. Edited by, Chap. 7.

Nuijten, W. (1994). *Time and resource constrained scheduling: A constraint satisfaction approach.*. Eindhoven University of Technology. (Ph.D. dissertation).

Pape, C. L. (1988). Des systemes d'ordonnancement flexibles et opportunistes. (Ph.D. dissertation)Universite Paris XI.

Pritsker, A. A., Watters, L. J., & Wolfe. , P. M. (1969). Multi-project scheduling with limited resources: a zero-one programming approach. *Management Science, 16*, 93–108.

Schutt, A., Feydy, T., Stuckey, P., & Wallace, M. (2011). Explaining the cumulative propagator. *Constraints, 16*(3), 250–282.

Schwindt, C. (2005). *Resource allocation in project management*. Springer Verlag..

Tercinet, P., Lente, C., & Neron, E. (2004). Mixed satisfiability tests for multiprocessor scheduling with release dates and deadlines. *Operations Research Letters, 32*(11), 326–330.