RAIRO Operations Research
Will be set by the publisher

A NEW *ANY-ORDER* SCHEDULE GENERATION SCHEME FOR RESOURCE-CONSTRAINED PROJECT SCHEDULING

Cyril Briand[1]

Abstract. In this paper, a new schedule generation scheme for resource-constrained project scheduling problems is proposed. Given a project scheduling problem and a priority rule, a schedule generation scheme determines a single feasible solution by inserting one by one each activity, according to their priority, inside a partial schedule. The paper proposes a generation scheme that differs from the classic ones in the fact that it allows to consider the activities in any order, whether their predecessors have already been scheduled or not. Moreover, activity insertion is performed so that delaying some already scheduled activities is allowed. The paper shows that this strategy remains polynomial and often gives better results than more classic ones. Moreover, it is also interesting in the fact that some priority rules, which are quite poor when used with classic schedule generation schemes, become very competitive with the proposed schedule generation scheme.

Keywords: project scheduling, schedule generation scheme, activity insertion

Mathematics Subject Classification. 90B35

1. Introduction

The paper focuses on project scheduling problems with resource and precedence constraints. This problem class is denoted as $PS|prec|C_{\max}$ in the literature (according to Brucker's notation [4]). A set of n interrelated activities has to be processed without preemption in order to achieve the project. Precedence constraints stipulate that an activity cannot start before all its predecessors have been completed. In order to carry out an activity, several limited-capacity resources are used. During the processing window $[s_j, s_j + p_j]$ of an activity j (s_j being the starting time of the activity and p_j its duration), $r_{j,k}$ resource units of resource k are used. A schedule is resource feasible if, at any time t, for any resource k, the sum $\sum r_{j,k}$ of the activities j in progress at time t never exceeds the maximum resource capacity R_k. A schedule is said *feasible* if it is both precedence-feasible (i.e. it satisfies the precedence constraints) and resource-feasible. The objective is to find a feasible schedule that minimizes the total project duration $C_{\max}$.

This optimization problem is known to be NP-hard [5]. Exact solution procedures have been proposed which are quite efficient on medium-sized problem instances [6, 7]. A large collection of heuristics (Genetic Algorithms, Tabu Search, multi-pass methods, ...) is also available to find good solutions for larger problem instances within a reasonable amount of time. For a description and a

[1] Université de Toulouse, LAAS-CNRS, 7 avenue du Colonel Roche, 31077 Toulouse, France

comparison of those heuristics it is recommended to refer to the Hartmann and Kolisch's survey [8], as well as to its recent update [9].

A frequent ingredient used as a core component of most of these heuristics is a Schedule Generation Scheme (SGS). Combined with a priority rule, a SGS determines for a given problem a single feasible solution. Since a SGS can be called very often in the same procedure, it has to run very fast in particular on large problem instances. Two classic SGSs are mainly referred in the literature : the *serial-SGS* and the *parallel-SGS*.

The serial-SGS consists of n iterations. At each iteration, an activity is selected according to its priority and inserted inside a partial schedule at the earliest (respecting the precedence and resource constraints), while keeping unchanged the starting time of the already scheduled activities. Only an *eligible* activity can be selected at each iteration. An activity is eligible if all its predecessors have already been scheduled. The activity priorities are determined according to a given rule (minimum Latest Starting Time, Earliest Starting Time, ...).

A parallel-SGS is time oriented and requires, at most, n iterations. At each iteration, several eligible activities can be scheduled. A time t_k is associated with one iteration k: t_k equals the earliest finishing time of the activities in progress at time t_{k-1}. The activities having their earliest starting time lower or equals to t_k are scheduled at time t_k, one by one, with respect to the priority order. If starting activity i at time t_k avoids the possibility to start activity j at the same time, then j is considered later, in further iterations.

A serial-SGS produces active schedules (no activity can be started earlier without scheduling another activity later), the class of the active schedules being dominant with regards to the $C_{\max}$ criterion. In contrast, a parallel-SGS produces no-delay schedules, that class being not dominant regarding the $C_{\max}$ criterion. The time complexity of both SGSs is $O(mn^2)$ where m is the number of resources and n the number of activities.

2. Motivations

A brief analysis of the parallel and serial SGSs brings the following remark. The solution space that those SGSs potentially allow to explore is rather restricted because, while inserting an activity, both SGSs do not allow to right-shift any activity that is already scheduled. As a consequence, the considered activity is often inserted at the end of the partial schedule, after all the other activities. Therefore, if the selection order of the activities is not optimal (in the sense it is not compatible with any optimal schedule), the probability for the produced schedule to be optimal is null.

This is illustrated below on the short example of Table 1. Activities $1 \dots 4$ have to be processed on a single cumulative resource of capacity $R = 6$. In Table 1, p_i is the processing time of activity i, r_i is its resource consumption (since there is only one resource the index k is omitted) and lst_i is its latest starting time. The precedence constraints are described on the precedence graph of Figure 1. Activities 0 and 5 are dummy activities representing the origin of time and the end of the project respectively.

Activities	0	1	2	3	4	5
p_i	0	3	2	4	3	0
r_i	0	3	4	6	2	0
lst_i	0	0	0	2	3	6

Table 1. Problem data

Figure 2.a) shows the schedule that is obtained using a serial-SGS and a LST priority rule which consists in selecting first the activity having the smallest latest starting time. The selection order is

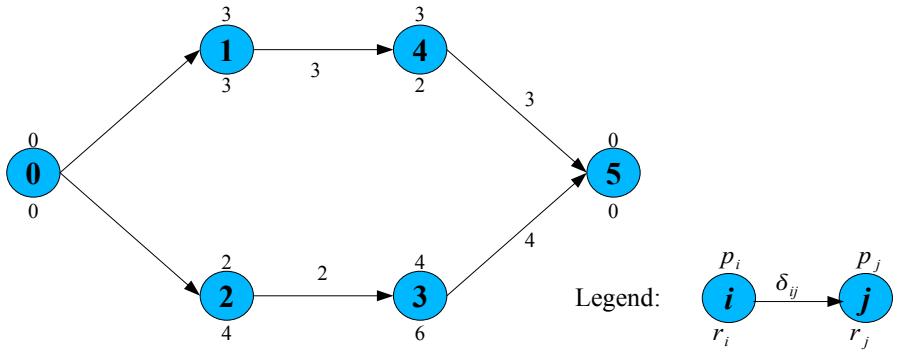
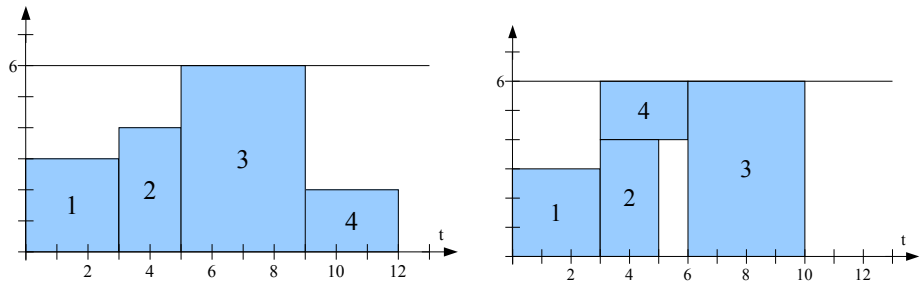


FIGURE 1. Precedence graph

$1 \prec 2 \prec 3 \prec 4$ which is not compatible with the optimal sequence represented on Figure 2.b) (since activity 3 cannot be delayed).



a) Solution using a serial-SGS b) An optimal schedule

FIGURE 2. Two feasible schedules

Another remark concerning the parallel and serial SGSs is relative to the concept of eligibility that they use. Indeed, in order to respect the precedence constraints, only a subset of eligible activities is considered for insertion at each iteration in both SGSs. Now it is known that the locations of some activities (e.g. those having an important resource consumption or a large processing time) can be very decisive regarding the final $C_{\max}$ value . Therefore, if such an activity becomes eligible too late, some poor decisions can be made at the beginning of the schedule process that definitively make the schedule suboptimal.

In accordance with the previous remarks an alternative SGS is proposed below. The activities can be considered in any order (hence the name *Any-Order-SGS*) and consequently the eligibility of an activity is never considered. Moreover the insertion of an activity can cause the right-shifting of some already scheduled activities.

3. Description of the Any-Order-SGS

The core component of the Any-Order-SGS is the insertion procedure proposed by Artigues et al. [1,2]. This procedure aims at solving the problem of inserting an activity inside a partial schedule, minimizing the total project duration increase. It is used in [1] for adapting reactively a schedule when unexpected activities occur and, in [2], as a core component of a Tabu Search procedure which unschedules some activities for inserting them in other positions. In the best of our knowledge, the work presented in this paper is the first attempt to use this insertion procedure as a component of a SGS.

In order to describe the insertion procedure, we need to define the activity-on-node network $N =< V, U_{\prec}, U_R >$ associated with a current partial schedule. Each vertex $i \in V = \{0, \ldots, n+1, n+2\}$ of N corresponds to a project activity. Classically, the dummy activities 0 and $n+1$ correspond to the beginning and the end of the project respectively. An arc $u \in U_R$ between two vertices i and j of N corresponds to a resource flow associated with a capacity vector $(c_{1,i\rightarrow j}, \ldots, c_{m,i\rightarrow j})$ (where $c_{k,i\rightarrow j}$ is the number of resource units k which are transferred from i to j). An arc $u \in U_{\prec}$ from i to j corresponds to a precedence constraint (i.e. $s_j - s_i \geq p_i$). The additional dummy activity $n+2$ allows to measure the increase of the $C_{\max}$ value after the insertion. There is an arc between $n+1$ and $n+2$ having the length $\delta_{n+1,n+2} = -est_{n+1}$ and s_{n+2} is set to 0.

For instance, Figure 3 shows the network N associated with the schedules of Figure 2.a-b), just before the insertion of activity 4. The arcs of U_R are in bold.

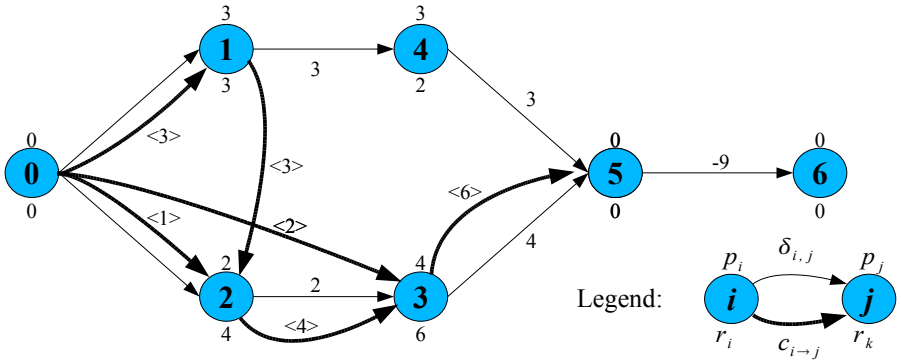


FIGURE 3. Graph G associated with a partial schedule

On the basis of such a network, the earliest starting time, est_i, and the latest finishing time, lft_i, of each activity i can be easily computed using the Bellman-Ford's algorithm. Indeed, an arc $u \in U_R$ from activity i to activity j, corresponding to a resource flow, obviously induces a precedence constraint between these activities, i.e. $s_j - s_i \geq p_i$. Therefore, replacing each arc $u \in U_R$ by the corresponding precedence constraint, a precedence graph is obtained which can be used for applying the Bellman-Ford's algorithm.

The Artigues's insertion procedure is described by Algorithm 1. It enumerates a finite set of maximal insertion cuts. A maximal cut corresponds to a set of arcs $u \in U_R$ having the sum of their capacity vectors equals to $(R_1, \ldots, R_m)$. It defines a bipartition of the activities. Each maximal cut characterizes in its turn a set of subcuts, each one being composed by a specific set of arcs chosen among those of the maximal cut. A subcut is said valid, with regard to the insertion of a given activity j, when the sum of the capacity vectors of the arcs is greater than $(r_{j,1}, \ldots, r_{j,m})$. The major result that is stated in [1,2] is that the number of maximal cuts and subcuts that need to be explored for finding an optimal insertion position is polynomial (since a lot of maximal cuts and subcuts are dominated by others).

Given an initial dominant maximal cut C_0, an activity x to insert with its execution window $[est_x, lft_x]$, the insertion procedure visits all the dominant valid subcuts still memorizing the best one. C_α^- and C_α^+ refer to as the activities being located at the left and at the right of the maximal cut C_α respectively. Given a maximal cut C_α, the activities i of C_α^- having the maximum eft_i are progressively remove from C_α^-, for generating the subcuts, until the remaining resource capacity becomes lower than the resource consumption of x. In this case, the activity j of C_α^+ having the minimum lst_j is determined. If $lst_j < lft_x$ then the procedure ends. In the other case, the next maximal cut $C_{\alpha+1}$ is determined. It is obtained from the previous maximal cut C_α by transferring j from C_α^+ to C_α^-. The time complexity of this procedure is $O(n^2 m)$ where n and m are the number of activities and resources respectively.

Algorithm 1 Search_Optimal_Insertion_Cut(x, N) - Artigues's insertion procedure

$\delta^* \leftarrow \infty$;
$C_0 \leftarrow$ Search_Initial_Dominant_Cut(x, N);
$\alpha \leftarrow 0$;
repeat
  $remcap_k \leftarrow R_k$, $\forall k$;
  $j \leftarrow \text{argmin}(lst_l)$, $\forall l \in C_\alpha^+$;
  repeat
    $i \leftarrow \text{argmax}\ (eft_l)$, $\forall l \in C_\alpha^-$;
    $\delta C_{\max} = \max(0, \max(est_x, eft_i) + p_x - \min(lft_x, lst_j)$;
    if $\delta C_{\max} < \delta^*$ **then**
      $C_{opt}^- \leftarrow C_\alpha^-$;
      $\delta^* \leftarrow \delta C_{\max}$;
    end if
    $remcap_k \leftarrow remcap_k - r_{ik}$, $\forall k$;
    $C_\alpha^- \leftarrow C_\alpha^- \setminus \{i\}$;
  until $\exists k$ such that $remcap_k < r_{xk}$;
  if $lst_j \geq lft_x$ **then**
    $\alpha \leftarrow \alpha + 1$;
    $C_\alpha \leftarrow$ Transfer$(C_{\alpha-1}, j)$;
  end if
until $lst_j < lft_x$

The Any-Order-SGS is described by Algorithm 2. At the beginning of the algorithm, U_R does not contain any arc, excepted the one between 0 and $n+1$ having the maximum capacity $(R_1, \ldots, R_m)$. The arcs of $U_\prec$ all correspond to precedence constraints (excepted the arc between $n+1$ and $n+2$).

Algorithm 2 Any-Order_SGS(N, r)

$\varepsilon \leftarrow \{1...n\}$;
for i $\leftarrow$ 1 to n **do**
  $j \leftarrow$ Select_Activity(ε, r);
  $C^* \leftarrow$ Search_Optimal_Insertion_Cut(j, N);
  Insert(j, C^*, N);
  Forward_Backward_Propagation(N);
  $\varepsilon \leftarrow \varepsilon - \{j\}$;
end for

At each iteration, an activity $j \in \varepsilon$ is selected according to the priority rule r. Then an optimal insertion subcut C^* is determined for the insertion of j in U_R, using the previous insertion procedure. In the next step, the insertion is performed leading to an update of U_R and possibly, an increase of s_{n+1} (in that case an update of the length of the arc between $n+1$ and $n+2$ has to be done). After insertion, the earliest and latest starting time have to be updated using a forward-backward propagation procedure. This update can be done in $O(n^2)$ [3]. The algorithm stops after n iterations when all the activities have been scheduled. Its time complexity is $O(n^3 m)$ since the insertion procedure works in $O(n^2 m)$.

An important issue is to determine whether it always exists a selection order of the activities which leads the Any-Order-SGS to find an optimal schedule (i.e. the Any-Order-SGS is dominant with regard to the makespan minimization). For proving that this property holds, we use a recurrent reasoning.

Let us consider an optimal active schedule S^* and let $N^* =< V^*, U_\prec, U_R^* >$ be its associated optimal activity-on-node network. We further refer to as C_i^* the optimal insertion subcut chosen for activity i in N^*. Without loss of generality, it is assumed that the activities are numbered with respect

to the increasing order of their finishing time f_i in S^*. Now, we assume that the activity are selected with respect to the increasing order of their number (i.e. $1 \prec 2 \cdots \prec n$) and progressively inserted by the Any-order-SGS in the activity-on-node network N.

At the first iteration, activity 1 is obviously inserted at time 0 in N inside the subcut $C_1 = C_1^*$, causing a makespan increase of p_1. At iteration 2, activity 2 is selected in its turn. Since schedule S^* is active, activity 1 and 2 cannot be scheduled earlier. Therefore, when inserting activity 2, the optimal makespan increase is exactly $(f_2 - f_1)$ and, among the optimal insertion subcuts which are characterized, there is necessarily the optimal subcut C_2^*. Without loss of generality, we assume that activity 2 is inserted inside this subcut (i.e. $C_2 = C_2^*$). Now let us consider the i^{th} iteration of the Any-Order-SGS where activity i has to be inserted inside the partial schedule composed by activities $1 \dots i-1$. It is assumed that the already scheduled activities have been inserted at their optimal position (i.e. $C_i = C_i^*$). The current $C_{\max}$ value is f_{i-1}. Because schedule S^* is active, none of the activities $1 \dots i$ can be scheduled earlier. Therefore, when inserting activity i in the partial schedule, the optimal makespan increase is exactly $(f_i - f_{i-1})$ and the optimal insertion subcut C_i^* necessarily belongs to the set of optimal subcuts that the insertion procedure characterizes. Under the hypothesis that, at each iteration i , the optimal insertion subcut C_i^* is always chosen among the set of optimal insertion subcuts which is characterized by the insertion procedure , the final schedule is optimal.

4. Experiments

Both the serial-SGS and the Any-Order-SGS have been tested on the problem instances of the PSPLIB library (`http://129.187.106.231/psplib/datasm.html`). This library offers 480 problem instances with 30 activities - 480 problem instances with 60 activities and 600 problem instances with 120 activities.

For each problem class, 4 priority rules A, B, C and D have been studied:

- A**:** selecting the activity having the minimum latest starting time;
- B**:** selecting the activity having the minimum latest finishing time;
- C**:** selecting the activity having the greatest ratio $p_i/(lft_i - est_i)$;
- D**:** selecting the activity having the greatest ratio $(p_i * \sum_k r_{ik})/(lft_i - est_i)$.

Those rules were chosen since, after having tested numerous other priority rules, they seem rather efficient, with regard to the average number of optimal solutions that were found, for at least one of the SGSs.

For each problem class, each instance, each priority rule and each SGS, a schedule is computed. Then, for each instance, the best schedule found using the Any-Order-SGS is compared in terms of makespan with the best schedule found using the serial-SGS (among the 4 ones computed using rules A-B-C-D). Table 2 reports the number of times the Any-Order-SGS gets better, equivalent or worse results than the Serial-SGS. For instance, for problem with 30 activities, the Any-Order-SGS is better in 169 cases (35.21%), equivalent in 293 cases (61.04%) and worse in 18 cases (3.75%). That table shows that the percentage of times where the Any-Order-SGS is better or equivalent to the serial-SGS is always important (96.25% and 95.42% for problem with 30 and 60 activities!) even if it tends to decrease for problem with 120 activities (78.66%). Moreover, when problem instances become larger, we observe that the Any-Order-SGS tends to be efficient when the serial-SGS fails and vice-versa. This observation shows that the Any-Order-SGS produces solutions intrinsically different from the ones produced by the serial one.

	30 activities			60 activities			120 activities		
Any-Order SGS is	Better	Equivalent	Worse	Better	Equivalent	Worse	Better	Equivalent	Worse
Percentage	35.21	61.04	3.75	33.13	62.29	4.58	56.5	22.16	21.34

Table 2. Comparison between the serial-SGS and the Any-Order-SGS

Table 3 shows the number of optimal solutions that both SGSs were able to find (the total number of known optimal solution is indicated in parentheses) and, on the other hand, the average relative distance between the best found makespan and the best known lower bound (i.e. $(s_{n+1} - LB)/LB$). For example, for problem having 60 activities, the Any-Order-SGS finds 304 optimal solutions out of the 357 which are known, while the serial-SGS only finds 283 optimal solutions. For the same problem, the Any-Order-SGS produces solutions that are 4.31% distant in average from the best known lower bound, when the serial-SGS produces solutions that are 5.66% distant. For any problem class, the table shows that the Any-Order-SGS is always better or equivalent than the serial one, regarding the number of total optimal solution found, even if we again observe that for large problem instances, the gap between both SGSs becomes smaller. However, when regarding the average distance from the best known lower bound, the Any-Order-SGS always produces better results, independently of the instance size.

Problem class	30 activities		60 activities		120 activities	
SGS	Any-Order	Serial	Any-Order	Serial	Any-Order	Serial
#opt	312 (/480)	274 (/480)	304 (/357)	283 (/357)	101 (/208)	99 (/208)
Avg Distance to best known LB	1.72%	3.43%	4.31%	5.66%	11.58%	13.33%

TABLE 3. Number of optimal solutions and average relative distance from the best known lower bounds

In order to refine the analysis, the same criteria as those considered in the previous table are reported for each priority rule separately (see Tables 4, 5 and 6). First, we see that the Any-Order-SGS remains efficient for any considered priority rule. We especially observe that rules C and D, which are quite poor when used with the serial-SGS, become really competitive with the Any-Order-SGS. The rule D seems particularly efficient since, with the Any-order-SGS, it produces in average solution which are always closer to the best known lower bound than the ones produces with the serial-SGS. This observation seems to indicate that, when the priority rule changes, the Any-Order-SGS is able to diversify the solutions it found, while the serial-SGS only produces efficient solutions for time-oriented rules (A and B).

30 activities				
	Any-Order-SGS		Serial-SGS	
	#opt (/480)	Avg. Dist. to best known LB	#opt (/480)	Avg. Dist. to best known LB
A	265	3.29%	243	4.41%
B	237	4.11%	234	5.02%
C	250	2.98%	179	8.84%
D	275	2.59%	178	9.32%

TABLE 4. Number of optimal solutions and average relative distance from the best known lower bounds for problems with 30 activities and for each priority rule

60 activities				
	Any-Order-SGS		Serial-SGS	
	#opt (/357)	Avg. Dist. to best known LB	#opt (/357)	Avg. Dist. to best known LB
A	271	5.72%	271	6.08%
B	248	6.55%	262	6.51%
C	269	5.24%	164	11.8%
D	284	4.87%	172	11.68%

TABLE 5. Number of optimal solutions and average relative distance from the best known lower bounds for problems with 60 activities and for each priority rule

120 activities				
	Any-Order-SGS		Serial-SGS	
	#opt (/208)	Avg. Dist. to best known LB	#opt (/208)	Avg. Dist. to best known LB
A	76	14.04%	95	13.72%
B	58	15.84%	75	14.69%
C	75	12.88%	10	24.92%
D	90	12.45%	14	24.56%

TABLE 6. Number of optimal solutions and average relative distance from the best known lower bounds for problems with 120 activities and for each priority rule

5. Conclusion

When solving resource-constrained project scheduling problems, most heuristics use as a core component either the serial or the parallel SGSs. This paper proposes an alternative Any-Order generation scheme that allows at each iteration to insert any unscheduled activity in the partial schedule (whether their predecessors are scheduled or not), allowing to delay some already scheduled activity. As the serial-SGS, this new SGS is dominant with regard to the makespan minimization since there always exist a selection order of the activities which allows to find an optimal schedule. Even if it is a bit slower (it runs in $O(n^3m)$ when the serial and the parallel ones work in $O(n^2m)$), the Any-Order-SGS produces efficient solutions, often better than the ones found using other SGSs. So it can be advantageously embedded in more sophisticated heuristics or metaheuristics for determining initial feasible schedules which can be improved in further stages, or for computing efficient upper bounds that allow to cut parts of search trees during branch-and-bound procedures. Moreover, the performance of the Any-Order-SGS is rather insensitive to the nature of the priority rule which is used, this feature being interesting for producing efficient diversified solutions. This kind of property is desirable in neighborhood search for diversifying the search space by exploring the neighborhood of various solutions which are not too similar.

References

[1] Artigues, C. & Roubellat, F., A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple mode, 2000, European Journal of Operational Research, 127, 297-316.

[2] Artigues, C., Michelon, P. & Reusser, S., Insertion Techniques for static and dynamic resource-constrained project scheduling, 2003, European Journal of Operational Research, 149, 249-267.

[3] Bartusch, M., Möhring, R.H. & Radermacher, F.J., Scheduling project networks with resource constraints and time windows„1988, Annals of Operations Research, 16, 201-240.

[4] Brucker, P., Drexl, A., Möhring, R.H., Neumann, K. & Pesh, E. Resource-constrained project scheduling: Notation, classification, models and methods, 1999, European Journal of Operational Research, 112, 3-41.

[5] Blazewicz, J., Lenstra, J. & Rinnooy Kan A., Scheduling subject to resource constraints: Classification and complexity, 1983, Discrete Applied Mathematics, 5, 11-24.

[6] Brucker P, Knust S., Schoo A. & Thiele O., A branch-and-bound algorithm for the resource-constrained project scheduling problem, 1998, European Journal of Operational Research, 107, 272-288.

[7] Demeulemeester, E. L. & W. S. Herroelen, New Benchmark Results for the Resource-Constrained Project Scheduling Problem, 1997, Management Science , 1485-1492.

[8] Hartmann, S. & Kolisch, R., Experimental evaluation of the state-of-the-art heuristics for the resource-constrained project scheduling problem, 2000, European Journal of Operational Research, 127, 394-407.

[9] Kolisch, R. & Hartmann, S., Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update, 2006, European Journal of Operational Research, 174, 1, 23-37.