Decision Making in Manufacturing and Services
Vol. 10 • 2016 • No. 1–2 • pp. 31–43

Insertion Algorithms with Justification for Solving the Resource-Constrained Project Scheduling

Marcin Klimek*, Piotr Łebkowski**

Abstract. The paper presents the resource-constrained project scheduling problem with the makespan minimization criterion. To solve the problem, the authors propose insertion algorithms that generate schedules with the use of forward serial and parallel decoding procedures. Schedules are improved with the use of the double justification by the extremes technique (first right and then left justification). The efficiency of the procedures proposed is tested on standard test problems from the PSPLIB library.

Keywords: insertion algorithms, resource-constrained project scheduling problem, makespan minimisation, justification, forward scheduling, priority rules

Mathematics Subject Classification: 90B35

Submitted: December 13, 2016

Revised: March 29, 2017

1. INTRODUCTION

Among the important practical problems of computing today is the project scheduling problem; that is, the problem of defining the start or finish times of project activities, allocating resources (plant and equipment, human resources) to individual activities with the adopted optimizing, time, and financial criteria met. On a generated schedule, orders are based on the raw materials necessary to perform project activities, funds for activity performance are secured, etc.

This paper analyzes the classic Resource-Constrained Project Scheduling Problem (RCPSP) with the makespan minimization criterion. This is one of the most-frequently--undertaken operational research problems. The algorithms and models used are discussed in survey papers (Brucker *et al.*, 1999; Hartmann and Briskorn, 2012; Józefowska and Węglarz, 2006; Kolisch and Padman, 2001). Being a generalization of the job-shop problem, the RCPSP is strongly NP-hard (Błażewicz *et al.,* 1983). For such problems, it is of a larger practical value to identify effective heuristic algorithms. The use of exact algorithms (that is, the branch and bound procedure [Demeulemeester and Herroelen, 1992], dynamic programming, discrete optimization, or binary programming)

* State School of Higher Education, Department of Computer Science, Biala Podlaska, Poland, e-mail: marcin_kli@interia.pl
** AGH University of Science and Technology, Faculty of Management, Krakow, Poland

DOI: http://dx.doi.org/10.7494/dmms.2016.10.1-2.31

is limited to scheduling projects with a few activities, as the time required to find an optimal solution is unacceptably long in the case of larger problems. Approximate (heuristic) algorithms with polynomial complexity are commonly used to solve the RCPSP; they find schedules in an acceptable amount of time, even for problems with a larger number of activities. Constructive algorithms generate schedules based on simple priority rules (priority algorithms) or rules governing the insertion of consecutive activities (insertion algorithms), whose advantage is a short execution time. They are used to quickly generate schedules for projects that include a large number of activities as well as generate initial schedules that are subsequently improved (the quality of the solutions generated with constructive algorithms is frequently unacceptable) with the use of metaheuristics; i.e., simulated annealing, genetic algorithms, tabu searches, etc. The effectiveness of metaheuristics is analyzed to a broad extent in Hartmann and Kolisch (2000) and Kolisch and Hartmann (2006).

Priority-rule-based heuristics use priority rules and SGS (Schedule Generation Scheme) procedures (Kolisch, 1996a). They are either single-pass methods generating a single schedule or multi-pass methods (X-pass methods) generating multiple schedules. Single-pass methods are quick and very simple to implement, but their efficiency is low, even for the most-efficient priority rules (Kolisch, 1996b), which are **LFT** (Latest Finish Time), **EFT** (Earliest Finish Time), **MTS** (Most Total Successors), and **MTSPT** (Most Total Successors Processing Time). The proposed multi-pass methods include algorithms that simultaneously use multiple priority rules (multi-priority rule methods) or sampling methods, where priorities for individual activities are allotted at random, with the priority rules used taken into consideration.

In this paper, insertion algorithms with justification are proposed; they are built in a similar fashion as effective flow shop procedures, whose effectiveness for the RCPSP has yet to be tested (to the best of the authors' knowledge). Schedules are generated based on the predefined activity list, in which the order of activities follows the predefined priority rule, with ordering constraints taken into consideration. The activity list is decoded to an executable schedule with the use of forward SGS procedures. Solutions are then improved with the use of the justification technique (Valls *et al.,* 2005; Valls *et al.,* 2006; Valls *et al.,* 2008), similar to the technique known as forward-backward improvement (Tormos and Lova, 2001; Tormos and Lova, 2003; Goncalves *et al.,* 2011), used in the most-efficient heuristics for the RCPSP.

The efficiency of the algorithms proposed is tested on test problems from the PSPLIB library (Project Scheduling Problem LIBrary) (Kolisch and Sprecher, 1997).

2. PROBLEM FORMULATION

A project is a unique set of interrelated activities that are to be executed in order to achieve the set objectives with use of available resources (employees, plant, and materials). In this paper, the classic nonpreemptive single-mode RCPSP is discussed. For the purposes of solving the RCPSP, the underlying project is presented as directed graph $G(V, E)$ in the AON (Activity-On-Node) representation, where V is the set of nodes (vertices) representing activities, while E is the set of arcs (edges) representing ordering and finish-start zero-lag precedence relationships between the activities. The activities

are performed with use of renewable resources, whose total number is limited and independent of time. The scheduling objective is to identify a schedule (start times for all activities) that would minimize a project's makespan (see Formula (1)) against resource constraints (see Formula (2)) and ordering constraints (see Formula (3)).

Minimize:

$$F = s_{n+1} - s_0, \tag{1}$$

against the constraints:

$$\forall t = 1, .., s_{n+1}, \forall k = 1, .., K : \sum_{i \in J(t)} r_{ik} \leqslant a_k, \tag{2}$$

$$\forall (i, j) \in E, s_i + d_i \leqslant s_j \tag{3}$$

where:

n – the number of project activities; in graph $G(V, E)$, two additional dummy activities (numbered 0 and $n + 1$) are inserted, representing the graph's initial and final vertices, respectively;
i – the number (index) of an activity;
s_i – the start time of the activity i;
$s_{n+1} - s_0$ – the difference of the start times of dummy activities $n + 1$ and 0; it is equal to the project makespan; s_{n+1} is determined as the latest of the finish times for activities from 1 to n, while s_0 is the earliest of the start times for activities from 1 to n;
$J(t)$ – the set of activities executed in time interval $[t - 1, t]$;
a_k – the number of available resources of type k; at any time t, the number of resources used must not exceed the number of available resources a_k for $k = 1, \ldots, K$;
K – the number of types of renewable resources;
k – the number (index) of a resource type;
r_{ik} – the demand of activity i for the resource of type k;
d_i – the duration of activity i.

A solution to the resource-constrained project scheduling problem with the makespan minimization criterion is the vector of the activities' start or finish times, known as the direct representation. Heuristics use indirect representations. For the RCPSP, the best results are obtained (Hartmann and Kolisch, 2000; Kolisch and Hartmann, 2006) for a representation taking the form of an activity list, where a solution is a permutation of the numbers of consecutive activities, taking into consideration the ordering relations.

An indirect-representation solution is transformed into a direct-representation solution with the use of decoding procedures known as schedule generation schemes (SGS), which transform an activity list into an executable schedule (the vector of the activity start times is determined in the process), taking into consideration the precedence and resource constraints.

The SGSs most often used for the RCPSP with the makespan minimization criterion include (Kolisch, 1996a):

- serial SGS – a procedure in which, for each consecutive instant t, the start time is determined for the first yet-to-be arranged activity in the activity list so that the time is as soon as possible, given the ordering and resource constraints;
- parallel SGS – a procedure in which, at each consecutive instant t, these non-scheduled activities (analyzed in the order defined on the list of activities) are started which may be started at a given instant t, given the ordering and resource constraints.

Various solution-generation techniques can be used to decode an activity list with an SGS:

- forward scheduling – planning of consecutive activities starting from the top of the list;
- backward scheduling – planning of consecutive activities starting from the bottom of the list, determining activity start times against the predefined due date.

The insertion algorithms proposed herein use forward scheduling.

3. PROPOSED INSERTION ALGORITHMS

For various optimization problems (including project scheduling), constructive algorithms are developed based on the insertion approach. In the case of such algorithms, a set of test solutions is created for scheduling problems by inserting activities at various places in the current solution (received during earlier iterations of the algorithm). From such a test solution set, the solution is selected with the best value of the objective function to serve as the current schedule in the next algorithm iteration. As a rule, an insertion algorithm runs in two phases:

1) initial phase, during which the initial activity list is developed with a selected algorithm using priority rules;
2) proper insertion phase, during which a sequence of n partial permutations is created, starting from a one-element permutation and finishing at an n-element permutation; each consecutive partial permutation is built by inserting the next activity on the activity list into the current partial permutation.

The initial activity list, order of selecting activities from the list, and places assigned to activities in partial permutations are algorithm specific. In this paper, new constructive algorithms **Alg1**, **Alg2**, and **Alg3** for solving the RCPSP are proposed; they have been developed by the authors (Klimek, 2010; Klimek and Łebkowski, 2010) based on the concept of procedures used for flow shop problem (Nawaz *et al.*, 1983; Woo and Yim, 1998).

The operation of the first of the algorithms proposed, **Alg1**, may be presented in the following steps, in which P is the activity list that the SGS decoding procedure transforms into the current schedule, while L is the list of currently eligible activities; i.e., those that may be started, as all of their predecessors are already on list P:

Step 1. Creation of initial list L comprised of all activities that may be started at time $t = 0$, ordered in line with the adopted priority rule.

Step 2. Insertion of the first activity from list L into all possible positions on current activity list P (with ordering constraints taken into consideration) and, in each case (each insertion position), generation of a partial schedule. From among all of the partial schedules obtained, the best schedule is selected against the optimization criterion adopted for partial schedules.

Step 3. Updating L: the activity inserted in Step 2 is deleted, and all of its successors (all of whose predecessors are already on list P) are placed on the list with the order according the adopted priority rule maintained.

Steps 2 and 3 are repeated until all activities are on list L, which is subsequently transferred by an SGS scheme into a schedule - solution of the RCPSP. The operation of procedures **Alg2** and **Alg3** is similar to that of **Alg1**, with the proviso that:

- the activities on list L may be ordered arbitrarily, as the order does not affect the operation of the algorithm;
- in Step 2 of the **Alg2** procedure, the insertion of each activity from list L into the last place of list P is sampled, and that activity is selected to occupy that place for which the best partial schedule is generated;
- in Step 2 of the **Alg3** procedure, the insertion of each activity from list L into every available place on list P is sampled; then, that activity and that insertion place are selected for which the best partial schedule is generated.

In Step 2 of the **Alg1**, **Alg2** and **Alg3** procedures, partial schedules are assessed in the following way: the project makespan is determined for the activity list represented by current list P followed by the other project activities in the order determined based on the priority rule used.

Table 1 presents the priority rules that have proven effective for the RCPSP (Kolisch, 1996b) and that have been used in the insertion algorithms proposed to arrange activities not yet on list P while determining the value of objective function F, as well as to arrange activities on list L in the case of the **Alg1** procedure.

Table 1. *Activity priority rules*

Rule name	Rule description
LST	minimum Latest Start Time for activities, with due date taken into consideration
LFT	minimum Latest Finish Time for activities, with due date taken into consideration
MTS	Most Total Successors, maximum number of all activity successors
MTSPT	Most Total Successors Processing Time, maximum aggregate duration of a given activity and all of its successors

Given a priority rule, if multiple activities have the same priority, then such activities are placed on the activity list in the order of activity numbers (the lower the activity number, the higher the activity is on the list). While generating solutions, the use of double justification by extremes is tested. Such a justification transforms (improves) a schedule generated by SGS procedures. First, all activities are right

justified, then they are left justified. The right (left) justification by extremes means that activities with maximum finish time (minimum start time) in schedule S subject to justification are justified consecutively. The right (left) justification of a given activity consists in determining (for that activity) the start time as late (early) as possible, given the ordering and resource constraints. The use of the justification technique enables a new schedule to be found for which activity makespans are no longer than in schedule S, and often shorter (Valls *et al.,* 2005; Valls *et al.,* 2006; Valls *et al.,* 2008).

4. ILLUSTRATIVE EXAMPLE

In order to explain the operation of the algorithms and double justification technique proposed, we will give an example. The project illustrating the problem considered is presented in Figure 1. The project consists of eight activities executed with the use of a single resource type, whose availability is 10. Vertices 0 and 9 represent the start and finish of project $G(V, E)$; they are dummy activities with a zero makespan and zero demand for resources. The edges of the graph reveal "technological" ordering relationships between activities; for instance, Activity 6 may start only after Activity 4 is finished.

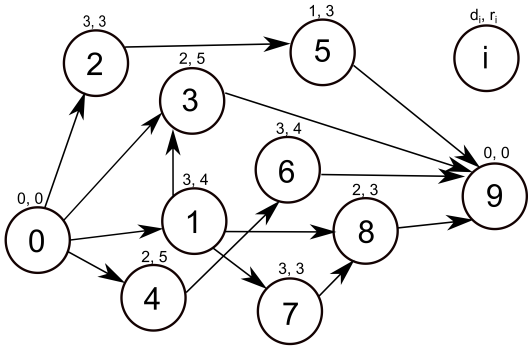

Fig. 1. *Example of project in Activity-On-Node representation*

Based on the priority rules, the proposed insertion algorithms determine the order of activities on the list L (**Alg1** algorithm) and arrange activities not on list P while generating schedules for the purposes of assessing the partial schedules (**Alg1**, **Alg2**, and **Alg3** algorithms). Assume that the **MTS** priority rule is used, according to which activities are placed on the activity list in decreasing order of the numbers of activity successors, with the ordering relations taken into consideration. The priorities of individual activities are as follows: $MTS_1 = 3, MTS_2 = 1, MTS_3 = 0$, $MTS_4 = 1$, $MTS_5 = 0, MTS_6 = 0$, $MTS_7 = 1$, and $MTS_8 = 0$. The resulting activity list is {1, 2, 4, 7, 3, 5, 6, 8}. For a given activity list, the decoding SGS procedure determines a schedule; that is, the start (finish) times for individual activities. For instance, Figure 2 presents the schedule generated by the serial SGS procedure for activity list {1, 2, 4, 7, 3, 5, 6, 8} obtained based on the **MTS** priority rule.

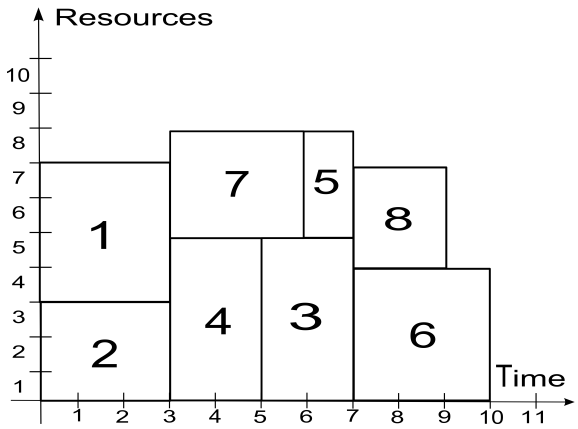

Fig. 2. *Schedule based on **MTS** priority rule, generated by serial procedure*

The schedule illustrated in Figure 2 (whose makespan is 10) admits improvement with use of the double justification technique. Right justification by extremes is applied first. Activities are subject to this justification in decreasing order of their respective finish times; justified are, consecutively, Activities 6, 8, 5, 3, 7, 4, 2, and 1 (where activities have the same finish time, they are justified in decreasing order of their numbers). Right justification of a given activity comes down to determining the latest possible start time for that activity, given the ordering and resource constraints and the current project makespan of 10. The schedule obtained after right justification (presented in Figure 3) is clearly better than the original one, as its makespan is 9 (by 1 less than that of the schedule in Figure 2). It has proved possible to delay the start of all activities except Activity 6.

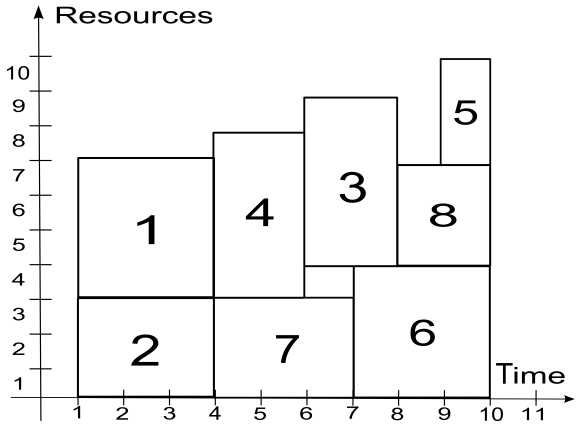

Fig. 3. *Schedule from Figure 2 transformed by right justification*

Right justification by extremes is followed by left justification by extremes. Activities are subject to this justification in increasing order of their respective start times

in the right justified schedule in Figure 3; justified are, consecutively, Activities 1, 2, 4, 7, 3, 6, 8, and 5 (where activities have the same start time, they are justified in increasing order of their numbers). Left justification of a given activity comes down to determining the earliest possible start time for that activity, given the ordering and resource constraints. The schedule obtained after left justification is presented in Figure 4; its makespan is 9. It has proved possible to start all activities earlier than in the schedule presented in Figure 3.

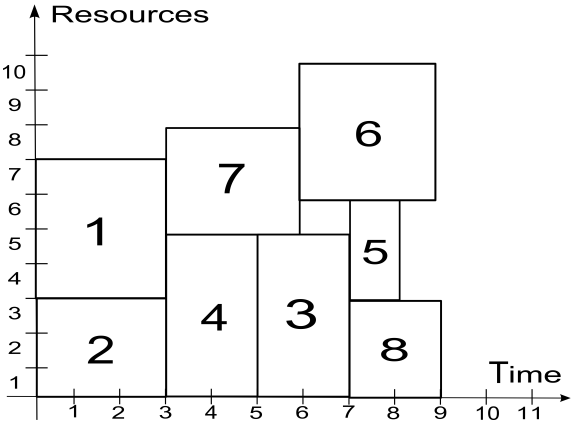

Fig. 4. *Schedule from Figure 3 transformed by left justification*

Even without the justification techniques, the use of the insertion algorithms proposed leads to identifying optimal schedules with a makespan of 8 for this simple project analyzed herein as an example. Figure 5 presents the schedule found by **Alg1** with use of the Serial SGS procedure against the **MTS** priority rule.

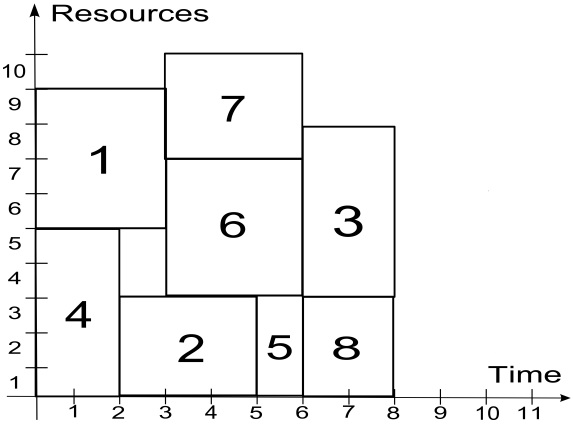

Fig. 5. *Minimum-makespan schedule found by **Alg1** with use of Serial SGS procedure against **MTS** priority rule*

5. RESULTS OF EXPERIMENTS

Experiments were run using an application implemented in C#, in the Visual Studio.NET environment, on an Intel Core i7-4770 CPU 3.4-GHz, 8-GB-RAM computer for 480 test instances from the J30 set (30-activity projects) and 480 instances from the J90 set (90-activity projects) sourced from the PSPLIB library (Kolisch and Sprecher, 1997).

The experiments were designed to assess the efficiency of the insertion algorithms developed for the project makespan minimization as well as to verify the effectiveness of schedule improvement with the use of double justification (DJ). The tables presenting the numerical results set forth the following: Av_dev – the average deviation from optimal (for 30-activity projects) or the best-known solutions (for 90-activity projects); Best_s – the number of solutions (from among the 480 test instances) with the value of objective function F equal to that recorded for the best-known schedule (the optimal schedule in the case of the 30-activity projects).

The priority rules used in the computations were the effective **LST**, **LFT**, **MTS**, and **MTSPT** rules described above, and additionally the **RND** rule, according to which the priorities are assigned to individual activities at random. For comparison purposes, numerical experiments were first performed for the single-pass-priority-rule heuristic and the selected priority rules, with solution quality checked following the the use of double justification (DJ). The results of the numerical experiments for the single-pass-priority-rule heuristic are presented in Tables 2 and 3, and the proposed **Alg1**, **Alg2**, and **Alg3** insertion algorithms are shown in Tables 4 and 5. The running times and numbers of the solutions checked for **Alg1**, **Alg2**, and **Alg3** are presented in Table 6.

Table 2. *Results of experiments for 30-activity projects – single-pass priority rule heuristic*

	Serial SGS		**Serial SGS+DJ**		**Parallel SGS**		**Parallel SGS+DJ**	
	Av_dev	Best_s	Av_dev	Best_s	Av_dev	Best_s	Av_dev	Best_s
RND	16.40%	146	6.42%	245	10.99%	159	4.74%	251
LST	12.51%	160	5.71%	239	8.52%	166	4.23%	260
LFT	11.56%	166	5.44%	242	8.25%	166	4.14%	258
MTS	11.63%	158	5.52%	235	8.32%	163	4.24%	256
MTSPT	12.46%	158	5.65%	238	8.71%	164	4.29%	256

Table 3. *Results of experiments for 90-activity projects – single-pass priority rule heuristic*

	Serial SGS		**Serial SGS+DJ**		**Parallel SGS**		**Parallel SGS+DJ**	
	Av_dev	Best_s	Av_dev	Best_s	Av_dev	Best_s	Av_dev	Best_s
RND	18.99%	135	6.60%	303	12.55%	135	5.30%	305
LST	15.49%	146	5.71%	302	11.17%	144	4.71%	313
LFT	15.34%	153	5.70%	304	11.33%	143	4.91%	311
MTS	14.74%	152	5.65%	302	10.60%	147	4.67%	313
MTSPT	15.60%	148	5.65%	303	11.05%	143	4.85%	310

Table 4. *Results of experiments for 30-activity projects – proposed insertion algorithms*

		Serial SGS		Ser. SGS+DJ		Parallel SGS		Par. SGS+DJ	
		Av_dev	Best_s	Av_dev	Best_s	Av_dev	Best_s	Av_dev	Best_s
Alg1	**RND**	8.25%	180	3.38%	270	6.02%	185	2.59%	286
	LST	3.08%	273	2.26%	307	3.15%	239	1.60%	316
	LFT	3.05%	271	1.75%	332	3.08%	235	1.40%	322
	MTS	3.44%	255	1.88%	322	3.20%	228	1.53%	317
	MTSPT	3.06%	271	1.88%	326	3.09%	238	1.39%	322
Alg2	**RND**	5.03%	224	1.89%	311	4.03%	211	1.79%	308
	LST	4.35%	225	1.97%	313	3.93%	211	1.61%	314
	LFT	4.34%	231	1.83%	316	3.73%	212	1.63%	310
	MTS	4.21%	236	1.81%	315	3.71%	211	1.64%	306
	MTSPT	4.35%	230	1.85%	314	3.84%	210	1.55%	309
Alg3	**RND**	4.68%	226	1.85%	311	4.02%	205	1.68%	311
	LST	3.92%	232	1.80%	319	3.78%	213	1.56%	318
	LFT	3.93%	237	1.63%	324	3.60%	214	1.58%	313
	MTS	3.86%	242	1.69%	320	3.56%	213	1.55%	312
	MTSPT	4.00%	237	1.74%	318	3.63%	213	1.49%	313

Table 5. *Results of experiments for 90-activity projects – proposed insertion algorithms*

		Serial SGS		Ser. SGS+DJ		Parallel SGS		Par. SGS+DJ	
		Av_dev	Best_s	Av_dev	Best_s	Av_dev	Best_s	Av_dev	Best_s
Alg1	**RND**	13.32%	157	4.92%	314	8.78%	161	4.01%	321
	LST	4.33%	316	3.35%	332	4.07%	245	2.71%	333
	LFT	4.59%	307	3.26%	329	4.12%	236	2.53%	337
	MTS	5.18%	279	3.38%	330	4.65%	217	2.62%	333
	MTSPT	4.74%	304	3.27%	332	4.32%	234	2.67%	335
Alg2	**RND**	8.35%	207	3.54%	329	6.42%	191	3.02%	329
	LST	7.46%	219	3.17%	334	5.77%	199	2.75%	331
	LFT	7.40%	217	3.13%	333	5.69%	196	2.76%	334
	MTS	7.27%	218	3.20%	334	5.64%	207	2.81%	331
	MTSPT	7.48%	216	3.26%	329	5.93%	200	2.80%	334
Alg3	**RND**	7.91%	217	3.33%	330	6.07%	205	2.98%	335
	LST	7.17%	221	3.12%	336	5.59%	202	2.72%	331
	LFT	7.16%	219	3.11%	333	5.56%	199	2.69%	334
	MTS	6.98%	219	3.06%	333	5.53%	209	2.72%	332
	MTSPT	7.12%	217	3.19%	330	5.75%	201	2.70%	34

Table 6. *Comparison of running times and numbers of solutions checked for insertion algorithms under analysis*

		Serial SGS		**Serial SGS+DJ**		**Parallel SGS**		**Parallel SGS+DJ**	
		CPU	Nr_s	CPU	Nr_s	CPU	Nr_s	CPU	Nr_s
J30	**Alg1**	0.006	157.9	0.068	158.0	0.035	158.1	0.092	158.1
	Alg2	0.004	124.2	0.055	123.1	0.028	124.1	0.075	123.2
	Alg3	0.009	217.7	0.096	215.3	0.048	217.1	0.127	215.5
J90	**Alg1**	0.043	1078.2	0.732	1078.1	0.397	1079.9	1.085	1079.1
	Alg2	0.030	711.7	0.499	699.4	0.256	702.1	0.715	695.0
	Alg3	0.056	1332.6	0.935	1308.3	0.476	1324.0	1.336	1313.4

Algorithm **Alg1** (using the parallel SGS procedure and double justification) proved most-effective. The use of effective priority rules improves the quality of the obtained solutions. Schedules generated by algorithms using random priority rule (**RND**) are of the worst quality and have the longest average running times. The quickest is the **Alg2** algorithm, which defines and verifies the least number of schedules in the course of its operation. The slowest is the **Alg3** algorithm, which defines and verifies the largest number of schedules. The best schedules were generated by **Alg1** using the parallel SGS and the double justification technique.

The parallel SGS yields better solutions than the serial SGS. A significant improvement in solution quality is observed following the use of the double justification of solutions. While use of this technique increases computation time, it decreases the project makespan.

When an insertion algorithm is used, the number of analyzed schedules grows quickly with an increase in the number of project activities, which may prevent their use for scheduling very large projects. The number of solutions checked for 90-activity projects is several times larger than that for 30-activity projects.

It is difficult to compare the algorithms proposed herein with the others analyzed in the research literature. They are more-efficient than single-pass priority rule methods. For multi-pass priority rule methods, the research reports give schedule quality after 1000 or 5000 iterations (1000 or 5000 generated schedules). The schedules generated for 30-activity projects with the use of multi-pass priority rule methods (such as sampling procedures) are slightly better than those found by the algorithms proposed (which, however, analyze significantly fewer solutions). The use of schedules found by insertion algorithms as initial solutions for metaheuristics might be profitable. The authors plan to research this problem in the future. The results of our experiments confirm the good efficiency of the constructive algorithms proposed, using efficient activity prioritizing rules.

6. CONCLUSIONS

In this paper, the authors have proposed insertion algorithms for the RCPSP with the makespan minimization criterion, with schedule improvement through the use of the double justification technique. The procedures analyzed use the ideas applied

to solve the flow shop scheduling problem. The efficiency of these procedures has been verified for test instances from the PSPLIB library. Numerical experiments have confirmed the good efficiency of the **Alg1** algorithm, which generates schedules slightly worse than the best-known (or optimal) solutions.

The schedules generated by the insertion algorithms proposed may be used as initial solutions for local search algorithms.

REFERENCES

Błażewicz, J., Lenstra, J., Kan, A.R., 1983. Scheduling subject to resource constraints classification and complexity. *Discrete Applied Mathematics*, **5**(1), pp. 11–24.

Brucker, P., Drexl, A., Mohring, R., Neumann, K., Pesch, E., 1999. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, **112**(1), pp. 3–41.

Demeulemeester, E., Herroelen, W., 1992. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, **38**(12), pp. 1803–1818.

Goncalves, J., Resende, M., Mendes, J., 2011. A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. *Journal of Heuristics*, **17**(5), pp. 467–486.

Hartmann, S., Briskorn, D., 2012. A Survey of Variants and Extensions of the Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research*, **207**(1), pp. 1–14.

Hartmann, S., Kolisch, R., 2000. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, **127**(2), pp. 394–407.

Józefowska, J., Węglarz, J. (eds.), 2006. Perspectives in modern project scheduling, Springer, Berlin.

Klimek, M., Łebkowski, P., 2010. Algorytmy wstawień dla zagadnienia harmonogramowania projektu ze zdefiniowanymi kamieniami milowym. In: Knosala, R. (red.), *Komputerowo zintegrowane zarządzanie*, T. 1, Opole, Oficyna Wydawnicza PTZP, pp. 676–685.

Kolisch, R., 1996a. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, **90**(2), pp. 320–333.

Kolisch, R., 1996b. Efficient priority rules for the resource-constrained project scheduling problem. *Journal of Operations Management*, **14**(3), pp. 179–192.

Kolisch, R., Hartmann, S., 2006. Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update. *European Journal of Operational Research*, **74**(1), pp. 23–37.

Kolisch, R., Padman, R., 2001. An integrated survey of deterministic project scheduling. *OMEGA*, **29**(3), pp. 249–272.

Kolisch, R., Sprecher, A., 1997. PSPLIB – a project scheduling library. *European Journal of Operational Research*, **96**(1), pp. 205–216.

Nawaz, M., Enscore, E., Ham, I., 1983. A heuristic algorithm for the m machine, n-job flow-shop sequencing problem. *OMEGA*, **11**(1), pp. 91–95.

Tormos, P., Lova, A., 2001. A competitive heuristic solution technique for resource-constrained project scheduling. *Annals of Operations Research*, **102**(1), pp. 65–81.

Tormos, P., Lova, A., 2003. An efficient multi-pass heuristic for project scheduling with constrained resources. *International Journal Of Production Research*, **41**(5), pp. 1071–1086.

Valls, V., Ballestin, F., Quintanilla, S., 2005. Justification and RCPSP: a technique that pays. *European Journal of Operational Research*, **165**(2), pp. 375–386.

Valls, V., Ballestin, F., Quintanilla, S., 2006. Justification technique generalisations. In: Józefowska, J., Węglarz, J. (eds.), *Perspectives in Modern Project Scheduling*, Springer, Berlin, pp. 205–223.

Valls, V., Ballestin, F., Quintanilla, S., 2008. A hybrid genetic algorithm for the resource--constrained project scheduling problem. *European Journal of Operational Research*, **185**(2), pp. 495–508.

Woo, D.S., Yim, H.S., 1998. A heuristic algorithm for mean flowtime objective in flowshop scheduling. *Computers and Operations Research*, **25**(3), pp. 175–182.