Computers and Chemical Engineering 69 (2014) 16–25

Contents lists available at ScienceDirect

Computers and Chemical Engineering

journal homepage: www.elsevier.com/locate/compchemeng

Resource constrained project scheduling problem with setup times after preemptive processes

Behrouz Afshar-Nadjafi*, Mahyar Majlesi

Department of Industrial Engineering, Faculty of Industrial and Mechanical Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran

ARTICLE INFO

Article history:
Received 3 March 2014
Received in revised form 10 June 2014
Accepted 24 June 2014
Available online 2 July 2014

Keywords:
Project scheduling
Preemption
Set up time
Genetic algorithm

ABSTRACT

In this paper, the preemptive resource constrained project scheduling problem with set up times is investigated. In this problem, a fixed setup time is required to restart when an process is preempted. The project contains activities inter-related by finish to start type precedence relations with a time lag of zero, which require a set of renewable resources. The problem formed in this way is an NP-hard. A mixed integer programming model is proposed for the problem and a parameters tuned meta-heuristic namely genetic algorithm is proposed to solve it. To evaluate the validation and performance of the proposed algorithm a set of 100 test problems is used. Comparative statistical results show that the proposed algorithm is efficiently capable to solve the problem.

1. Introduction

The resource constrained project scheduling problem (RCPSP) is a known optimization problem which is NP-hard (Blazewicz et al., 1983). The objective of RCPSP is to minimize the makespan of the project while the renewable resources availabilities are considered given. In the literature there are several exact methods and heuristics that solve the RCPSP (Pritsker et al., 1969; Christofides et al., 1987; Choi et al., 2004; Hartmann and Kolisch, 2006; Zhang et al., 2006; Hartmann and Briskorn, 2010; Jairo et al., 2010; Agarwal et al., 2011; Fang and Wang, 2012; Koné, 2012; Kyriakidis et al., 2012; Paraskevopoulos et al., 2012; Jia and Seo, 2013; Kopanos et al., 2014).

In the basic project scheduling problems it is assumed that each activity once started, will be executed until its completion. Preemptive project scheduling problem refers to the scheduling problem which allows activities to be preempted at any discrete time instance and restarted later. There are some solution methods for the preemptive project scheduling problems in the literature (Kaplan, 1988; Demeulemeester and Herroelen, 1996; Buddhakulsomsiri and Kim, 2006; Damay et al., 2007; Ballestin et al., 2008; Vanhoucke and Debels, 2008; Van Peteghem and Vanhoucke, 2010; Afshar-Nadjafi, 2014; Haouari et al., 2014).

While setup times have been widely studied in machine scheduling (Anglani et al., 2005; Roshanaei et al., 2009; Liao et al., 2012; Nagano et al., 2012) the literature concerning setup times in the context of project scheduling is scant. Generally, in project scheduling a setup has been defined as a preparation of all requirements for execution an activity. The time needed for this preparation is then called a setup time. In the area of project scheduling, the assumption that the duration of an activity reflects both setup and processing times has also been made throughout the years. This common assumption can be justified as long as setup times are relatively small in comparison to processing times. However, in the case which activities require some considerable setup times, modeling and solving such a problem as a classical RCPSP, especially in the preemptive case may lead to poor solutions (Kolisch, 1995).

Motivation of this paper is to show how to model the setup times in the preemptive RCPSP. First, a mixed integer programming model is developed for the preemptive resource constrained project scheduling problem with setup times. This problem is called PRCPSP-ST. This model is not considered in the past literature. Second, a new efficient parameter-tuned algorithm namely genetic algorithm is developed to solve it due to NP-hardness of the problem. Finally, the effectiveness of the proposed method to solve the PRCPSP-ST is evaluated and effect of setup time on makespan of the project is analyzed.

Reminder of the paper is organized as follows: Section 2 describes the problem PRCPSP-ST and mixed integer formulation for it. Section 3 explains the steps of the proposed algorithm to solve the problem. Section 4 contains the computational results and

* Corresponding author. Tel.: +98 9125817105.
E-mail addresses: afsharnb@alum.sharif.edu (B. Afshar-Nadjafi), mahyar_industrial_engineer@yahoo.com (M. Majlesi).

http://dx.doi.org/10.1016/j.compchemeng.2014.06.012
0098-1354/

performance evaluation of the proposed algorithm. Finally, Section 5 concludes the paper.

2. Problem description

The preemptive resource constrained project scheduling problem with setup times (PRCPSP-ST) involves the scheduling of project's activities on a set K of renewable resource types. Each activity i is performed in a single mode with deterministic duration of d_i. Each activity i requires r_{ik} units of renewable resource type k ($k = 1, ..., K$) during each time unit of its execution. For each renewable resource type k, the availability R_k is constant throughout the project horizon. In sequent, assume a project represented in activity on node, *AON*, format by a directed graph $G = \{N, A\}$ where the set of nodes, N, represents activities and the set of arcs, A, represents finish to start precedence relations with a time-lag of zero. The preempt-able activities are numbered from the dummy start activity 0 to the dummy end activity $n+1$ and are topologically ordered, i.e. each successor of an activity has a larger activity number than the activity itself. Once an activity i is preempted, a setup time ST_i is required to restart the activity. The following assumptions are considered in the PRCPSP-ST:

- The activities can be preempted in discrete time points.
- The number of preemptions for an activity is not limited.
- A setup time is required to start an activity after preempted.
- Initial setup time of an activity is included in its duration.
- Setup times are deterministic and schedule-independent.
- Setups are inseparable, i.e. an activity is started immediately after its setup is finished.
- Setups occurred after preemption require the same amount of renewable resources with that of when the activity is processing.
- All parameters are integers.

The objective of the PRCPSP-ST is to schedule a number of activities, in order to minimize makespan of the project. A schedule S is defined by a vector of activities finish (start) times and is said to be feasible if all precedence relations and renewable resources constraints are satisfied. Let $f_{i,j}$ denotes the finish time of jth unit of activity i. The unit of an activity j can be defined as the smallest discrete segment of the activity, i.e. an hour, a day, a week, etc. In order to ease the formulation, $f_{i,0}$ can be used to denote start time of activity i. By defining binary decision variables x_{ij} which specify whether jth unit ($1 \le j \le d_i - 1$) of an activity i is preempted or not, PRCPSP-ST can be conceptually formulated as follows:

$$\min \ C_{\max} = f_{(n+1),0} \tag{1}$$

Subject to:

$$f_{i,d_i} \le f_{j,0}; \quad \forall (i,j) \in A \tag{2}$$

$$f_{i,j-1} + 1 \le f_{i,j} - x_{i,(j-1)}(1 + ST_i); \quad i = 0, 1, ..., n+1; \quad j = 1, ..., d_i \tag{3}$$

$$f_{i,j} - x_{i,(j-1)}(1 + ST_i) \le f_{i,j-1} + 1 + Mx_{i,(j-1)}; \quad i = 0, 1, ..., n+1; \quad j = 1, ..., d_i \tag{4}$$

$$f_{0,0} = 0 \tag{5}$$

$$\sum_{i \in S_t} r_{ik} \le R_k; \quad k = 1, ..., K; \quad t = 1, ..., f_{(n+1),0} \tag{6}$$

$$x_{i,j} \in \{0, 1\}, f_{i,j} \in \text{Int}; \quad i = i = 0, 1, ..., n+1; \quad j = 0, 1, ..., d_i \tag{7}$$

The objective in Eq. (1) is to minimize the makespan of the project. Eq. (2) denotes the finish to start zero time lag precedence relations. Eqs. (3) and (4) guarantee that setup time should be taken into account if an activity is preempted. Parameter M is a considerably positive constant. However, Eqs. (3) and (4) preserve the relation between x_{ij} and f_{ij}. Eq. (5) specifies that start dummy activity 0 should be started at time 0. Constraint set in Eq. (6) take care of the renewable resources limitations. It represents renewable resource constraints for every resource type k by considering for every time instant t for all activities i such that the activity is in progress in period t. S_t denotes the set of activities which are in progress or their setups are in progress at time interval $[t-1, t]$. This constraint set is true logically, however, it cannot be solved directly because there is no easy way to translate the set S_t. Eq. (7) specifies that the decision variables f_{ij} are integers, while x_{ij} are binary.

In continue we developed an improved mathematical formulation for PRCPSP-ST based on following decision variables:

x_{ivt} 1, if vth unit of activity i is finished at time t
0, otherwise (binary decision variable)

y_{ivt} 1, if vth unit ($1 \le v \le d_i - 1$) of activity i is preempted at time t
0, otherwise (binary decision variable)

z_{ivt} 1, if setup of v^h unit ($2 \le v \le d_i$) of activity i is in progress at time interval $[t-1, t]$
0, otherwise (binary decision variable)

It is clear that an activity with duration of 0 is never in progress and thus does not have a corresponding decision variable which is set to 1. This problem, however, can be easily overcome: the dummy start and end activity are assigned a dummy mode with duration of 1. Also, the other parameters for dummy modes are assumed 0. All other activities with zero duration can be eliminated, provided that the corresponding precedence relations are adjusted appropriately. The resulting schedule may be transferred into a schedule for the original problem by removing the dummy start and end activity, and one time unit left shifting. Using the above notation, the PRCPSP-ST can be mathematically formulated as follows, where $EST(i)$ and $LST(i)$ denote the earliest start time and the latest start time of activity i, respectively.

$$\text{Min} = \sum_{t=EST(n+1)+1}^{LST(n+1)+1} t\, x_{(n+1)1t} \tag{8}$$

The objective in Eq. (8) is to minimize the makespan of the project.

$$\sum_{t=EST(i)+1}^{LST(i)+1} t\, x_{id_it} \le \sum_{t=EST(j)+1}^{LST(j)+1} t\, x_{j1t} - 1, \quad \text{for} \ \ (i,j) \in A \tag{9}$$

Eq. (9) denotes the finish to start zero time lag precedence relations constraints.

$$\sum_{t=EST(i)+(v-1)}^{LST(i)+(v-1)} tx_{i(v-1)t} + 1 \le \sum_{t=EST(i)+v}^{LST(i)+v} tx_{ivt} - \left[\sum_{t=EST(i)+(v-1)}^{LST(i)+(v-1)} yi_{(v-t)t} \right] \times (1 + ST_i) \le \sum_{t=EST(i)+(v-1)}^{LST(i)+(v-1)} tx_{i(v-1)t} + 1 + M \quad \text{for} \ \ \{i \in Nd_i \ne 1\}, \quad v = 2, ..., d_i \tag{10}$$

Eq. (10) guarantees that setup time should be taken into account if activity is preempted. Parameter M is a considerably positive constant.

$$x_{ivt} \le x_{i(v+t)(t+1)} + y_{ivt} \le 1$$

$$\text{for} \ \ \{i \in Nd_i \ne 1\}, \quad v = 1, ..., d_i - 1, \quad t = EST(i) + v, ..., LST(i) + v \tag{11}$$

$$x_{ivt} \geq y_{ivt}, \quad \text{for} \quad \{i \in Nd_i \neq 1\}, \quad v = 1, \ldots, d_i,$$
$$t = EST(i) + v, \ldots, LST(i) + v \tag{12}$$

Eqs. (11) and (12) preserve the relation between x_{ivt} and y_{ivt}.

$$Z_{i,v,t} \leq \sum_{t0=EST(i)+(v-1)}^{t-2} x_{i,v-1.t0}, \quad \text{for} \quad \{i \in NST_i \neq 0\}, \quad v = 2, \ldots, d_i,$$
$$t = EST\{i\} + v - ST_{i,\ldots,} LST(i) + v - 1 \tag{13}$$

$$Z_{i,v,t} \leq \sum_{t0=t+1}^{\min(t+ST(i),LST(i)+v)} x_{i,v,t0}, \quad \text{for} \quad \{i \in NST_i \neq 0\},$$
$$v = 2, \ldots, d_i, \quad t = EST(i) + v - ST_{i,\ldots,} LST(i) + v - 1 \tag{14}$$

$$Z_{i,vt} \geq \sum_{t0=EST(i)+(v-1)}^{t-2} x_{i,v-1,t0} + \sum_{t0=t+1}^{\min(t+ST(i),LST(i)+v)} x_{i,v,t0} - 1,$$
$$\text{for} \quad \{i \in NST_i \neq 0\}, \quad v = 2, \ldots, d_i,$$
$$t = EST(i) + v - ST_i, \ldots, LST(i) + v - 1 \tag{15}$$

Eqs. (13)–(15) preserve the relation between x_{ivt} and z_{ivt}.

$$\sum_{t=EST(i)+v}^{LST(i)+v} x_{ivt} = 1, \quad \text{for} \quad i \in N, \quad v = 1, \ldots, d_i \tag{16}$$

Eq. (16) guarantees that only one finish time is assigned to each unit time of each activity i.

$$\sum_{t=EST(i)+v}^{LST(i)+v} y_{ivt} \leq 1, \quad \text{for} \quad \{i \in Nd_i \neq 1\}, \quad v = 1, \ldots, d_i - 1 \tag{17}$$

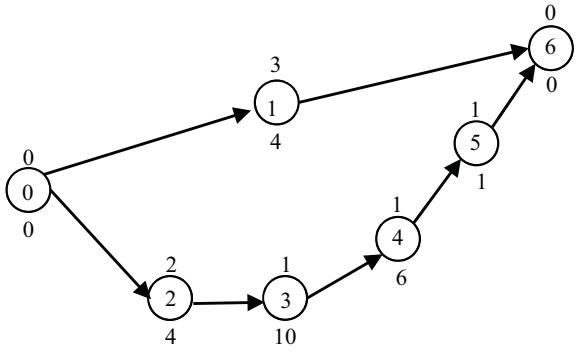

Fig. 1. Problem instance for the PRCPSP-ST.

Eq. (17) specifies that at most one preemption is allowed for each unit time of each activity i.

$$\sum_{t=EST(i)+v-ST_i}^{LST(i)+v-1} Z_{ivt} \leq ST_i, \quad \text{for} \quad \{i \in NST_i \neq 0\}, \quad v = 2, \ldots, d_i \tag{18}$$

Constraint set in Eq. (18) preserves the logical relation between decision variables Z_{ivt} and setup time of each activity i.

$$\sum_{i=1}^{n} \sum_{v=1}^{d_i} (r_{ik}(x_{ivt} + Z_{i,v,t}) \leq R_k, \quad \text{for} \quad k = 1, \ldots, K, \quad t = 1, \ldots, T \tag{19}$$

Constraint set in Eq. (19) take care of the non-renewable resources limitations, where T is an upper bound of makespan. The makespan of the project obtained from non-preemptive RCPSP can be considered as T.

$$x_{ivt}, y_{ivt}, Z_{i,v,t} \in \{0, 1\}, \quad \text{for} \quad i \in N, \quad v = 1, \ldots, d_i,$$
$$t = EST(i) + v, \ldots, LST(i) + v \tag{20}$$

Eq. (20) specifies that the decision variables x_{ivt}, y_{ivt} and z_{ivt} are binaries.

Here the problem is illustrated by an instance. The corresponding *AON* project network is shown in Fig. 1. There are 5 real activities

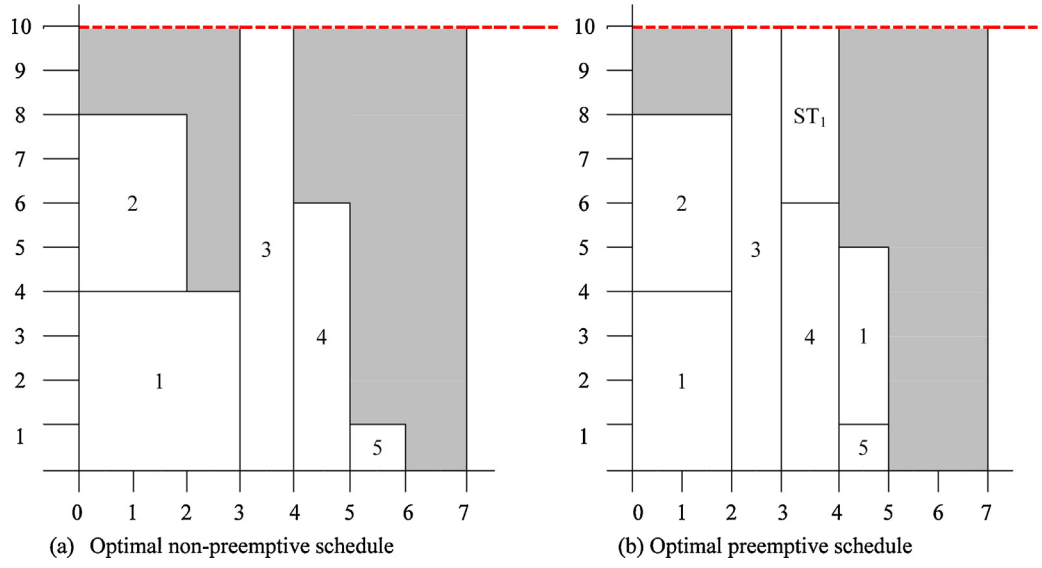

Fig. 2. Optimal schedules for the problem instance. (a) Optimal non-preemptive schedule and (b) optimal preemptive schedule.

(0 and 6 are dummy activities) and a renewable resource type with availability of 10. The number above the node denotes the activity duration, and the number below the node denotes the resource requirement, respectively. The setup time for activities 1 and 2 is assumed 1 while setup time for other activities is assumed 0.

Using the LINGO version 11, based on branch and bound method, we obtained the optimal non-preemptive schedule with a makespan of 6 (Fig. 2a), while the optimal makespan for preemptive case is 5 (Fig. 2b). It is clear from Fig. 2b that activity 1 is preempted at time $t = 2$ and resumed at time $t = 4$. Also, a setup is necessitated due to this interruption.

3. Solution procedure

In this section a solving procedure based of genetic algorithm (GA) is developed to solve PRCPSP-ST. Genetic algorithm tries to implement the idea of survival of the fittest in the field of combinatorial optimization.

3.1. Proposed genetic algorithm

The genetic algorithm (GA) is the well-known meta-heuristic that have been successfully applied to a noticeable number of project scheduling problems (Shadrokh and Kianfar, 2007; Lova et al., 2009; Van Peteghem and Vanhoucke, 2010; Agarwal et al., 2011; Barrios et al., 2011; Afshar-Nadjafi et al., 2013; Khoshjahan et al., 2013). GA is based on the mechanisms of biological evolution and natural genetics. Solution representation, selection and reproduction are the basic elements of GA.

3.1.1. Solution representation

Two important representations for schedules are the random-key (RK) and the activity-list (AL) representation. It is deduced from experimental tests that procedures based on (AL) representation outperform the other procedures (Hartmann and Kolisch, 2000). In this study the AL representation is used to encode a project schedule and revised serial schedule generation scheme (SSGS) followed by a double justification to decode the schedule representation to a schedule. Replacing an activity i with d_i activities with duration of 1, a feasible solution is represented by an $N' = \sum_{i=1}^{n} d_i$ elements vector (I). In the new project structure, each duration unit $j = 1, ..., d_i$ of an activity i is successor of duration unit $(j-1)$. However, the resource requirements of sub-activities are same as the original activities.

$$I = (J^1, J^2, \ldots, J^{N'}) \tag{21}$$

Having obtained a feasible solution represented by the vector described above, the start (finish) times of all activities is defined by a revised version of serial SGS followed by a double justification. The SGS determines how a feasible schedule is constructed by assigning start (finish) times to the activities. The serial SGS sequentially adds activities to the schedule until a feasible complete schedule is obtained. In each run, the first un-scheduled activity in the activity list is selected and the first possible start time is assigned to the selected activity such that precedence and resource constraints are preserved. In the revised version of serial SGS, setup time after preemptions is embedded. The pseudo-code for the revised version of serial SGS is shown in Table 1.

The double justification is a two steps improvement procedure which is implemented on a schedule generated by the revised serial SGS. In the first step, the activities are right justified in the schedule, that is, except for the first and the last dummy activities, all activities are shifted to the right, starting from the final activity immediate predecessors until the initial activity immediate successors. This step generates a right active schedule; a schedule where no activity can be finished later without delaying some other activities or increasing the makespan (Valls et al., 2005). Since the final activity is not shifted, the current makespan is held. If during the first step a slack in the schedule is generated, the second step tries to reduce the makespan by eliminating that slack. In this second step, activities are analogically left justified in the schedule, that is, except for the initial activity; all activities are shifted to the left, starting from the initial activity immediate successors until the final activity. This step generates a left active schedule; a schedule where no activity can be started earlier without violating the precedence or resource constraints. The pseudo-code for the double justification procedure is shown in Table 2.

Table 1
The pseudo-code for the revised serial SGS.

```
1  model=input_problem_data;
2  procedure SGS (solution, model)
3    for j∈{solution} do
4      j is the rth part of the activity (i) |d_j = 1;
5      if r=1 then
6        t1←calculate the earliest start time of the activity (i) according to
   precedence relations;
7        t2←find the earliest start time of the activity (i) in [t1, T] with respect
   to resource constraints in t2;
8        schedule activity (i) to start in t=t2;
9        update remaining resources in t2;
10   else
11     t1←calculate the earliest start time of the activity (i) according to
   precedence relations;
12     if remaining resources are enough in t1 then
13     schedule activity (i) to start in t=t1;
14     update remaining resources in t1;
15     else
16       t2←find the earliest start time of the activity (i) in [t1+1, T] with
   respect to resource constraints in [t2, t2+STi];
17       schedule activity (i) to start in t=t2+STi;
18       update remaining resources in [t2, t2+STi];
19     endif
20   endif
21 return t2;
21 endfor
23 end procedure
Where:
STi: setup time of activity(i)
di: duration of activity(i)
T=Upper bound of project makespan
```

3.1.2. Initial generation

The GA starts with producing the initial generation with size *npop* solutions. As used in the literature (Afshar-Nadjafi et al., 2013), the initial generation can be created randomly. In proposed GA, each solution in initial generation is randomly created as follows:

Table 2
The pseudo-code for the double justification.

```
1   model=input_problem_data;
2   procedure SGSDJ (solution, model)
3     S1=SGS (solution, model);
4     solution←updatesolution through forming activities in non-increasing
   order of their finishing times in S1;
5     S2 = SGS (solution, reverse_project_model);
6     S3←justifying schedule S2 to right through: cost(S1)–F(S2);
7     S3←update S3 through removing time slack between t = 0 and
   t = min(earliest start times in S3);
8     solution←update solution through forming activities in non-decreasing
   order of their starting times in S3;
9     SDJ=SGS (solution, model);
10    returnSDJ;
11  end procedure
Where:
Cost(S): makespan of schedule(S)
F(S): finishing times of activities in schedule(S)
```

an empty $N' = \sum_{i=1}^{n} d_i$ elements vector is generated. To fill this vector an activity (with duration of 1) is randomly selected from set of activities that are not selected before and all its predecessors are already met. This process is repeated until the activity list is full.

3.1.3. Selection and reproduction

Once the initial generation is created, the process continues in an iterative way in order to obtain the next high quality generation. The better solutions of current generation will have higher possibilities to survive for the reproduction. In proposed GA, the reproduction process consists of four phases: in the first one, the codified segments of two selected solutions are exchanged following the crossover operators. In the second phase, the selected solutions are randomly changed using the mutation operators. In the third phase, the solutions created by crossover and mutation operators are inserted to the current generation. In the last phase, the current generation is sorted by best to worst fitness function. Then the top *npop* solutions are moved to the next generation. In this way, these operators lead to the exploitation of good solutions and to the exploration of new areas of search space.

The selected solutions for crossover and mutation operators are called parents. The parents are selected based on their fitness values with roulette wheel (RW) scheme or 2-tournament procedure. In the roulette wheel scheme each solution has a probability of selection that is directly proportional to its fitness value. In the 2-tournament selection procedure, two population elements are chosen randomly and the element with the best fitness function value is selected. The selected solutions are grouped in couples randomly. Then the crossover operators act with a probability P_c for each couple, and the mutation operators act on each child created by the crossover operators with a probability P_m.

Selection plays a major role in evolutionary algorithms since it determines the direction of search whereas other genetic operators propose new search points in an undirected way. The term selective pressure β is usually used to denote the strength of selection to influence the directedness of the search.

3.1.3.1. Crossover operators. Crossover operators lead to the exploitation of good solutions which depend on the type of solution representation. Given two parent solutions selected for crossover, two child solutions are produced. In the proposed GA, two following operators are considered which act on the selected couple:

i. One point crossover: an integer x is randomly selected from set $\{1, \ldots, N'\}$. The first x positions in the child solution 1 are directly taken from the parent 1, in the same order. The next $N' - x$ positions are taken with their relative order in the parent 2 in their relative order. The child solution 2 is created inversely.
ii. Two points crossover: two random crossover-points, x_1 and x_2 are drawn from set $\{1, \ldots, N'\}$ with $x_1 < x_2$. The first x_1 positions in the child solution 1 are directly taken from the parent 1, in the same order. The positions between $x_1 + 1$ and x_2, are taken with their relative order in the parent 2's sequence, and the positions between $x_2 + 1$ and N' are again taken from the parent 1, in their relative order. The child solution 2 is created inversely.

3.1.3.2. Mutation operators. Mutation operators lead to the exploration of new areas of search space. In the proposed GA, two following operators are considered which act on the selected solution:

i. *Insertion*: First, an integer a is randomly selected from set $\{1, \ldots, N'\}$. Let J_e^a denotes the last predecessor and J_w^a denotes the first successor of activity J^a in activity list of the selected solution I. Then, an integer h different from a is randomly selected from set $\{e+1, \ldots, w-1\}$. Finally, the activity in position a is moved to position h. This operator preserves the feasibility of the new solution (Shadrokh and Kianfar, 2007).
ii. *Swap*: Two random integers, c and d are drawn from set $\{1, \ldots, N'\}$ with $c < d$. Then the positions of activities J^c and J^d in the activity list are exchanged. Also, some activities between these positions are shifted to left or right such that feasibility of resulting solution is preserved.

Table 3
The pseudo-code for the proposed GA.

```
1  model=input_problem_data;
2  define parameters number_of_population[npop], p_crossover[Pc],
   p_mutation[Pm],tournament_size, selective pressure [β],max_schedules;
3  crossover_population[nc]=Pc*npop;
4  mutation_population[nm]=Pm* Pc *npop;
5  initialize schedule_counter[k]=0;
6  initialize population with npop feasible solutions created randomly;
7  evaluate population;
8  sort population in non-increasing order of fitness value;
9  best_solution←The first chromosome in population;
10 i=1;
11   while i=nc do
12        select 2-parent chromosome from population according to
   selection procedure;
13        apply crossover;
14        i=i+1;
15   endwhile
16 j=1;
17   while j=nm do
18        select a chromosome from crossover_populationrandomly;
19        apply mutation;
20        j=j+1;
21   endwhile
22 new_population←mergenc, nm, population;
23 evaluate new_population;
24 sort new_population in non_increasing order of fitness value;
25 population←Truncatenew_population;
26 best_solution←The first chromosome in the population;
27 if k is smaller than max_schedules then
28        k=k+1;
29     go to line 10
30   else
31 termination the proposal genetic algorithm;
32 returnbest_solution;
33 endif
```

3.1.4. Stopping criterion

The procedure is continued until a predetermined number of schedules are produced. We obtained good results by indexing the number of produced schedules to the size of the problem, i.e. use of the small number of produced schedules for small problems and large number of produced schedules for larger problems. Therefore after some trials to obtain reasonable results, we fixed the number of produced schedules limited to 100 N'. The pseudo-code for the proposed GA is shown in Table 3. Also, a graphical representation of the proposed method is shown in Fig. 3.

3.1.5. Calibration

Performance of the meta-heuristics depends excessively on the value of their parameters and operators. In this paper the Taguchi experimental design is used to tune the parameters of GA. The Taguchi method determines the optimal level of controllable factors and minimizes the effect of noise (Taguchi, 1986). In the proposed GA, the factors that should be tuned are *npop*, P_c, P_m and β, where *npop* is the population size, P_c and P_m are the crossover and mutation probabilities. Also, β denotes the selective pressure in the 2-tournament selection procedure. Different levels of these factors are shown in Table 4.

A randomly generated problem with 30 non-dummy activities (with 116 sub-activities with duration of 1) is used for parameter

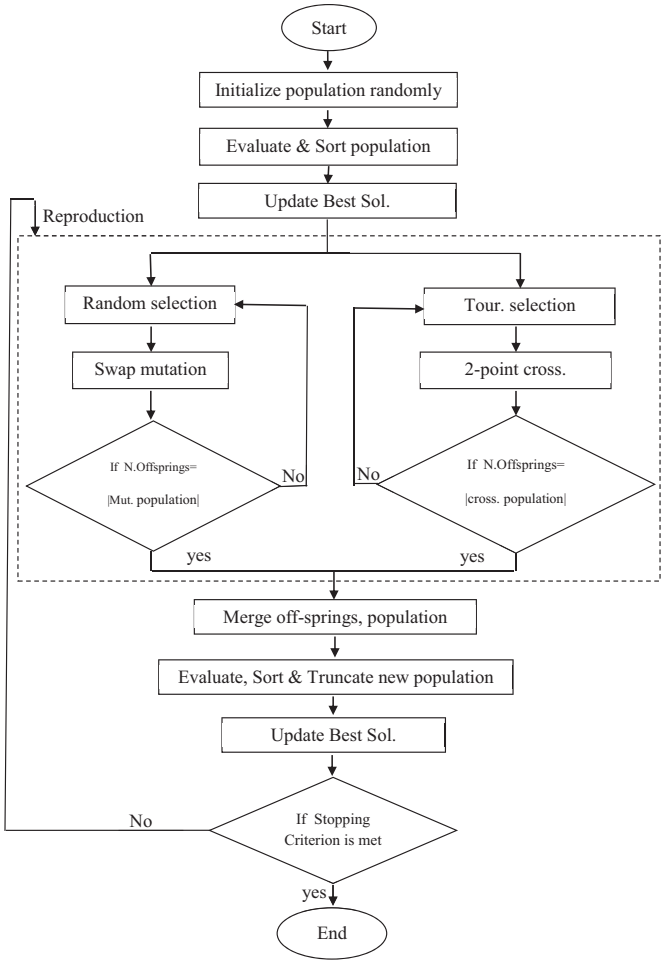

Fig. 3. The graphical representation of the proposed GA.

Table 4
Factors and factors levels.

Factor	Level
β	6, 7, 8
P_c	0.65, 0.8, 0.95
P_m	0.1, 0.3, 0.4
Npop	50, 100, 150

tuning. Using MINITAB software version 16, a L27 orthogonal array design is applied. To obtain more reliable data each experiment executed 4 times and the best result is used. Also, same selection and reproduction scheme is used for all 108 runs. This design is shown in Table 5.

The average S/N ratio obtained at each level is shown in Fig. 4. According to this figure, the optimal levels of population size *npop*, probability of crossover P_c, probability of mutation P_m, and selective pressure β are 100, 0.8, 0.1, and 7, respectively.

Analysis of variance (ANOVA) for S/N ratios is reported in Table 6. This table indicates the relative significance of individual factors in terms of their main effect on the objective function. Also, the response table for signal to noise ratios is shown in Table 7. The last row in this table ranks the factors according to their impact degree (lower rank means higher impact).

However, the Taguchi design is used to calibration of the selection and reproduction scheme. In the proposed GA, two levels are considered for selection, crossover and mutation operators as shown in Table 8.

Using MINITAB software version 16, a L8 orthogonal array design is applied. The randomly generated problem for parameters tuning used again with tuned values for *npop*, P_c, P_m and β. Each experiment executed 4 times to obtain more reliable data. The number of produced schedules limited to 10,000 as stopping criterion. This design is shown in Table 9.

The average S/N ratio obtained at each level is shown in Fig. 5. According to this figure, the 2-tournement scheme for selection, two point scheme for crossover and insertion scheme for mutation is more calibrated.

Table 5
The orthogonal L27 array design for parameters tuning.

Experiment	Factors				Iterations				Best iteration
	S-pressure	P-crossover	P-mutation	N-population	1	2	3	4	
1	6	0.65	0.1	50	50	50	50	50	50
2	6	0.65	0.1	100	48	50	48	48	48
3	6	0.65	0.1	150	49	50	49	49	49
4	6	0.8	0.3	50	50	50	50	50	50
5	6	0.8	0.3	100	50	49	50	50	49
6	6	0.8	0.3	150	50	49	50	49	49
7	6	0.95	0.4	50	50	50	50	50	50
8	6	0.95	0.4	100	48	50	48	50	48
9	6	0.95	0.4	150	49	50	50	49	49
10	7	0.65	0.3	50	49	49	49	50	49
11	7	0.65	0.3	100	48	50	50	50	48
12	7	0.65	0.3	150	49	50	50	48	48
13	7	0.8	0.4	50	50	50	49	48	48
14	7	0.8	0.4	100	48	50	48	48	48
15	7	0.8	0.4	150	49	49	49	49	49
16	7	0.95	0.1	50	50	49	49	50	49
17	7	0.95	0.1	100	49	49	50	48	48
18	7	0.95	0.1	150	49	49	49	49	49
19	8	0.65	0.4	50	49	50	50	49	49
20	8	0.65	0.4	100	50	50	50	50	50
21	8	0.65	0.4	150	50	50	50	49	49
22	8	0.8	0.1	50	48	50	48	49	48
23	8	0.8	0.1	100	48	48	50	50	48
24	8	0.8	0.1	150	50	50	48	48	48
25	8	0.95	0.3	50	50	50	50	50	50
26	8	0.95	0.3	100	48	48	48	48	48
27	8	0.95	0.3	150	50	50	48	48	48

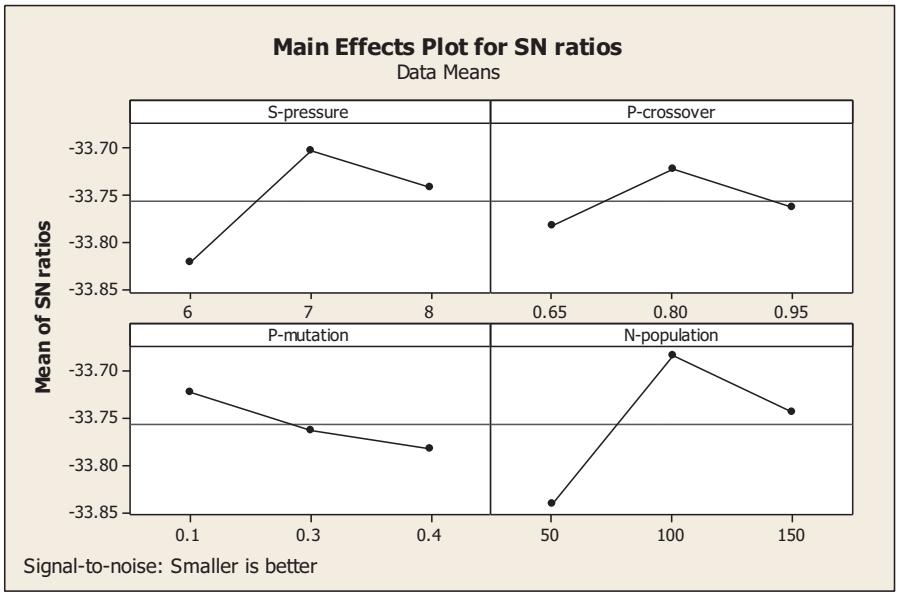

Fig. 4. The mean S/N ratio plot for each level of the factors.

Table 6
Analysis of variance for S/N ratios.

ANOVA	DF	Seq SS	Adj SS	Adj MS	*F*	*P*
S-pressure	2	0.06529	0.06529	0.032643	2.21	0.139
P-crossover	2	0.01639	0.01639	0.008196	0.55	0.584
P-mutation	2	0.01639	0.01639	0.008196	0.55	0.584
N-population	2	0.11447	0.11447	0.057234	3.87	0.040
Residual error	18	0.26643	0.26643	0.014802		
Total	26	0.47897				

Table 7
Response table for S/N ratios.

	S-pressure	P-crossover	P-mutation	N-population
1	−33.82	−33.78	−33.72	−33.84
2	−33.70	−33.72	−33.76	−33.68
3	−33.74	−33.76	−33.78	−33.74
Delta	0.12	0.06	0.06	0.16
Rank	2	3.5	3.5	1

Table 8
Levels of the selection and reproduction operators.

Factor	Level
Selection procedure	RW, 2-tournament
Crossover procedure	One-point, two-point
Mutation procedure	Insertion, swap

4. Performance evaluation

4.1. Validation of proposed model and GA

In order to validate the proposed GA algorithm for the PRCPSP-ST, a set of 10 problems with 10 non-dummy activities is generated by the generator ProGen developed by Drexl et al. (2000) using the parameters given in Table 10.

The proposed GA were coded in Borland C++ 5.02 and executed on a personal computer with an Intel Core i5, 2.4 GHz processor and 4000 MB memory. Table 11 presents the computational results of the proposed algorithm. The computational time of the proposed GA to solve these problems is less than 20 s. For problems with 10 activities, the results are compared with the optimal solutions obtained by LINGO 11. In this table, set up time ST of an activity is defined as a percent of its duration. ST = 0% means that there is no setup time to restart after preemption while ST = 25% and ST = 50% mean that setup time of an activity is 25% and 50% of its duration, respectively.

Table 11 shows that when the number of activities is equal to 10, the results obtained by proposed GA and LINGO are identical. Also, Table 11 reveals that for problems 2, 3, 6, 8 and 10, makespan of the project in the preemptive RCPSP is same as the non-preemptive case. In problem 1, makespan of PRCPSP without setup time (ST = 0%) is less than non-preemptive RCPSP, while when setup time is ST = 25% and 50%, preemption has no improving effect. In problems 4 and 5, a setup time up to ST = 25% and in problems 7

Table 9
The orthogonal L8 array design for calibration of the selection and reproduction scheme.

Experiment	Factors			Iterations				Best iteration
	Selection	Crossover	Mutation	1	2	3	4	
1	RW	One-point	Insertion	50	50	50	51	50
2	RW	One-point	Swap	49	49	50	50	49
3	RW	Two-point	Insertion	48	48	48	48	48
4	RW	Two-point	Swap	51	50	50	51	50
5	2-Tour.	One-point	Insertion	48	50	48	48	48
6	2-Tour.	One-point	Swap	48	48	48	48	48
7	2-Tour.	Two-point	Insertion	48	48	48	48	48
8	2-Tour.	Two-point	Swap	48	48	50	48	48

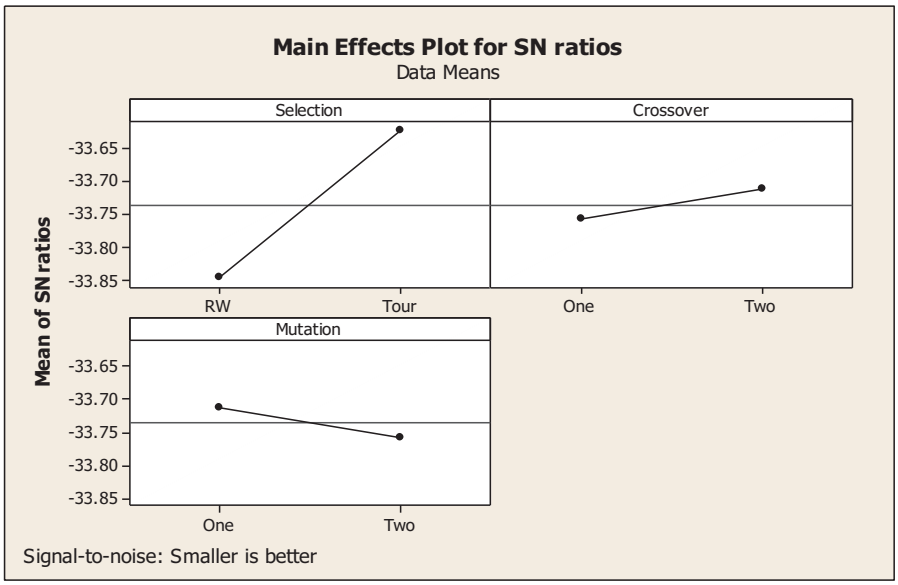

Fig. 5. The mean S/N ratio plot for each level of the operators.

and 9, a setup time up to ST = 50% has improving effect on makespan of the project.

4.2. Experimental results

In order to evaluate the proposed GA for problems with more activities which LINGO is unable to solve the problem optimally in a reasonable time, a set of 90 project networks is considered. These project networks contain 30, 60 and 90 activities which are randomly chosen from the PSPLIB made available by Kolisch and Sprecher (1996). The proposed GA executed 10 times for each problem to obtain more reliable data. The results are reported in Table 12.

Table 10
The parameter settings for the problem set.

Control parameter	Value
Activity durations	Integer [1, 5]
Number of initial activities	Integer [1, 3]
Number of terminal activities	Integer [1, 2]
Maximal number of successors and predecessors	3
Number of renewable resources	2
Activity renewable resource demand (per period)	Integer [1, 10]
Resource factor (RF)	0.5
Resource strength (RS)	0.2
Network complexity (NC)	1.5

Table 11
Comparison of the results for problems with 10 activities obtained by the GA and LINGO.

Problem number	RCPSP	GA for PRCPSP-ST			LINGO for PRCPSP-ST		
		ST = 0%	ST = 25%	ST = 50%	ST = 0%	ST = 25%	ST = 50%
Problem 1	14	13	14	14	13	14	14
Problem 2	19	19	19	19	19	19	19
Problem 3	15	15	15	15	15	15	15
Problem 4	18	17	17	18	17	17	18
Problem 5	20	19	19	20	19	19	20
Problem 6	19	19	19	19	19	19	19
Problem 7	27	26	26	26	26	26	26
Problem 8	22	22	22	22	22	22	22
Problem 9	24	23	23	23	23	23	23
Problem 10	19	19	19	19	19	19	19

Table 12
Comparison of the results for problems with 30, 60 and 90 activities obtained by the GA.

# Problems	# Activities	ST (%)	Max. imp. (%)	Avr. imp. (%)	Imp. Inst. (%)	Avr. Opt. Dev (%)	ARD (%)	Avr. CPU
30	30	0	6.4	1.18	30	1.13-	0.17	120.038
		25	4.7	0.34	10	0.003-		80.341
		50	2	0.18	6.66	0.09		57.949
30	60	0	8.1	0.82	16.66	0.45-	0.16	848.425
		25	3.9	0.27	13.33	1.23		460.483
		50	1.9	0.06	3.33	1.54		300.273
30	90	0	4.2	0.35	16.66	0.92	0.25	381.729
		25	1.3	0.04	3.33	3.36		222.163
		50	0.00	0.00	0.00	**3.56**		147.786

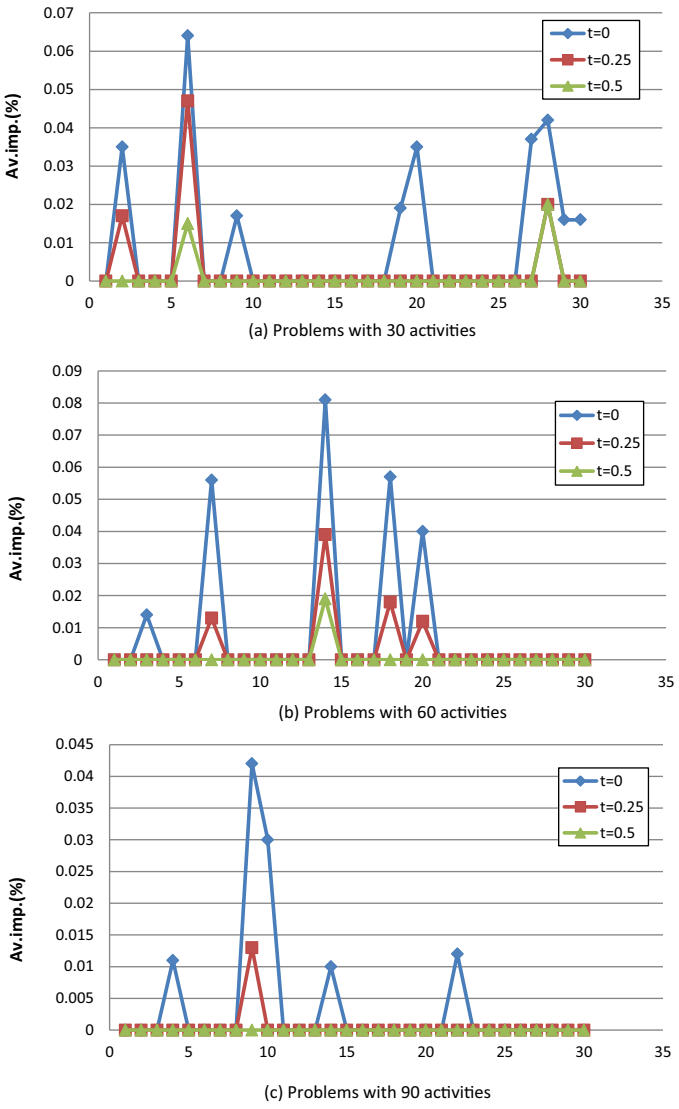

Fig. 6. Average improvement PRCPSP-ST compared to RCPSP.

In Table 12, Max. imp. (%) and Avr. imp. (%) denote the maximum and average percentage of improvement in project makespan compared to non-preemptive RCPSP, respectively. Also, Imp. Inst. (%) denotes the percentage of improved problems compared to non-preemptive RCPSP. These measures in Table 12 reveal that when the number of activities or setup time is increased, the justifiability of preemption is reduced. This is demonstrated in Fig. 6 based on Avr. imp. (%). It is clear from Fig. 6(c) that in the problems with 90 activities in the case of ST = 50% no preemption is justified. Avr. CPU denotes the average CPU-time for the GA (in seconds). Average CPU-time for GA indicates when the number of activities is increased the complexity of problem is increased, too. Also, average CPU-time is a decreasing function of setup time ST%. ARD (%) denotes the average relative deviation percentages for the GA, while relative deviation (*RD*) percentage for each problem is obtained as follows:

$$RD = \frac{C_{max} - C^*_{max}}{C^*_{max}} \quad (22)$$

C_{max} is makespan of the project and C^*_{max} is the best makespan of the project obtained by GA. ARDs for the GA algorithm are not high. This means that proposed GA gives robust solutions.

However, Avr. Opt. Dev. (%) denotes the average percentage of makespan's deviation from the lower bound of non-preemptive RCPSPs available in PSPLIB. The negative values mean that averagely preemption is justified in comparison with available lower bounds for non-preemptive RCPSPs. A maximum value of this measure (3.56%) is related to problems with 90 activities in the case of ST = 50% in which preemption is not justified at all. This low value shows that the results obtained by proposed GA are close to available lower bounds.

Overall, these results reveal that when the number of activities is large, while the LINGO is unable to solve the problem, there is a satisfying solution by the proposed GA in a reasonable CPU time.

5. Summary and conclusions

In this paper, we attempted to solve the preemptive resource constrained project scheduling problem with setup time to restart after preemptions. The objective is to schedule the activities in order to minimize makespan of the project subject to the precedence relations and renewable resource constraints. The problem conceptually formulated, and then genetic algorithm (GA) designed to solve it. To improve the efficiency of the proposed GA, the parameters were fine-tuned based on Taguchi experimental design. In order to validate the proposed model and developed GA, a set of 10 test problems with 10 non-dummy activities were generated using the ProGen software package. For the generated problems, the results obtained by proposed GA were compared with the optimal solutions obtained by LINGO 11 with three levels for setup times. From the computation results, we could clearly see that the GA algorithm could efficiently find the optimal schedule for the problems with 10 activities within 20 s. In order to generalize the statistical results for problems with more activities which LINGO is unable to solve the problem optimally in a reasonable time, a set of 90 project networks with 30, 60 and 90 activities were randomly chosen from the PSPLIB. For these problems, we could find out that the proposed GA is capable to find a satisfying solution in a reasonable CPU time measured by the average percentage of makespan's deviation from the lower bound of non-preemptive RCPSPs available in PSPLIB. Also, from the computation results, we could clearly see that the GA algorithm could efficiently give robust solutions for the project scheduling problem measured by the average relative deviation (ARD) percentages. However, results displayed that the average CPU time is an increasing function of the number of activities and setup times. Results showed that the justifiability of preemption is a decreasing function of the number of activities. Also, the results revealed that increasing setup time of preemptions leads to preemptions be less attractive insofar as in the problems with 90 activities in the case of ST = 50% no preemption is justified. This research helps mangers to schedule their projects in order to minimize the makespan of their projects when preemption permitted with set up times. Extensions of this research might be of interest, such as studying the preemptive project scheduling combined with generalized precedence relations or multi-mode case. Also, the considered problem with limited number of preemptions might be of interest.

References

Afshar-Nadjafi B. A solution procedure for preemptive multi-modeproject scheduling problem with mode changeability to resumption. Appl Comput Inf 2014 [in press] doi: 10.1016/j.aci.2014.02.003.
Afshar-Nadjafi B, Rahimi A, Karimi H. A genetic algorithm for mode identity and the resource constrained project scheduling problem. Sci Iran 2013;20(3):824–31.
Agarwal R, Colak S, Erenguc S. A neurogenetic approach for the resource constrained project scheduling problem. Comput Oper Res 2011;38(1):44–50.
Anglani A, Grieco A, Guerriero E, Musmanno R. Robust scheduling of parallel machines with sequence-dependent set-up costs. Eur J Oper Res 2005;161(3):704–20.
Ballestin F, Valls V, Quintanilla S. Pre-emption in resource constrained project scheduling. Eur J Oper Res 2008;189:1136–52.

Barrios A, Ballestin F, Valls V. A double genetic algorithm for the MRCPSP/max. Comput Oper Res 2011;38(1):33–43.
Blazewicz J, Lenstra J, Rinnooy Kan A. Scheduling subject to resource constraints: classification and complexity. Discrete Appl Math 1983;5:11–24.
Buddhakulsomsiri J, Kim D. Properties of multi-mode resource constrained project scheduling problems with resource vacations and activity splitting. Eur J Oper Res 2006;175:279–95.
Choi J, Realff MJ, Lee JH. Dynamic programming in a heuristically confined state space: a stochastic resource-constrained project scheduling application. Comput Chem Eng 2004;15:1039–58.
Christofides N, Alvarez-Valdes R, Tamarit JM. Project scheduling with resource constraints: a branch-and-bound approach. Eur J Oper Res 1987;29(3):262–73.
Damay J, Quilliot A, Sanlaville E. Linear programming based algorithms for preemptive and non-preemptive RCPSP. Eur J Oper Res 2007;182:1012–22.
Demeulemeester E, Herroelen W. An efficient optimal procedure for the preemptive resource constrained project scheduling problem. Eur J Oper Res 1996;90:334–48.
Drexl A, Nissen R, Patterson JH, Salewski F. ProGen/(x – an instance generator for resource constrained project scheduling problems with partially renewable resources and further extensions. Eur J Oper Res 2000;125:59–72.
Fang C, Wang L. An effective shuffled frog-leaping algorithm for resource constrained project scheduling problem. Comput Oper Res 2012;39(5):890–901.
Hartmann S, Kolisch R. Experimental investigation of heuristics for resource constrained project scheduling: an update. Eur J Oper Res 2006;17:23–37.
Hartmann S, Briskorn DA. Survey of variants and extensions of the resource constrained project scheduling problem. Eur J Oper Res 2010;207(1):1–14.
Hartmann S, Kolisch R. Experimental evaluation of state-of-the-art heuristics for the resource constrained project scheduling problem. Eur J Oper Res 2000;127:394–407.
Haouari M, Kooli A, Néron E, Carlier J. A preemptive bound for the resource constrained project scheduling problem. J Scheduling 2014;17(3):237–48.
Jairo R, Torres M, Edgar GF, Pirachicán-Mayorga C. Project scheduling with limited resources using a genetic algorithm. Int J Project Manage 2010;28(6):619–28.
Jia Q, Seo Y. Solving resource constrained project scheduling problems: conceptual validation of FLP formulation and efficient permutation-based ABC computation. Comput Oper Res 2013;40(8):2037–50.
Kaplan LA. (Ph.D. thesis) Resource-constrained project scheduling with preemption of jobs (Ph.D. thesis). University of Michigan; 1988.
Khoshjahan Y, Najafi AA, Afshar-Nadjafi B. Resource constrained project scheduling problem with discounted earliness-tardiness penalties: mathematical modeling and solving procedure. Comput Ind Eng 2013;66:293–300.
Kolisch R. Project Scheduling under resource constraints – efficient heuristics for several problem classes. Heidelberg: Physica; 1995.
Kolisch R, Sprecher A. PSPLIB – a project scheduling library. Eur J Oper Res 1996;96:205–16.
Koné O. New approaches for solving the resource constrained project scheduling problem. 4OR 2012;10(1):105–6.
Kopanos GM, Kyriakidis TS, Georgiadis MC. New continuous-time and discrete-time mathematical formulations for resource-constrained project scheduling problems. Comput Chem Eng 2014;68:96–106.
Kyriakidis TS, Kopanos GM, Georgiadis MC. MILP formulations for single- and multi-mode resource-constrained project scheduling problems. Comput Chem Eng 2012;36:369–85.
Liao CJ, Chao CW, Chen LC. An improved heuristic for parallel machine weighted flow time scheduling with family set-up times. Comput Math Appl 2012;63(1):110–7.
Lova A, Tormos P, Cervantes M, Barber F. An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes. Int J Prod Econ 2009;117(2):302–16.
Nagano MS, Silva AA, Lorena LAN. A new evolutionary clustering search for a no-wait flow shop problem with set-up times. Eng Appl Artif Intell 2012;25(6):1114–20.
Paraskevopoulos DC, Tarantilis CD, Ioannou G. Solving project scheduling problems with resource constraints via an event list-based evolutionary algorithm. Expert Syst Appl 2012;39(4):3983–94.
Pritsker AAB, Watters LJ, Wolfe PM. Multi-project scheduling with limited resources: a zero-one programming approach. Manage Sci 1969;16(1):93–108.
Roshanaei V, Naderi B, Jolai F, Khalili M. A variable neighborhood search for job shop scheduling with set-up times to minimize makespan. Future Gener Comput Syst 2009;25:654–61.
Shadrokh S, Kianfar F. A genetic algorithm for resource investment project scheduling problem, tardiness permitted with penalty. Eur J Oper Res 2007;181:86–101.
Taguchi G. Introduction to quality engineering. Tokyo: Asian Productivity Organization; 1986.
Valls V, Ballest F, Quintanilla S. Justification and RCPSP. A technique that pays. Eur J Oper Res 2005;165(2):375–86.
Vanhoucke M, Debels D. The impact of various activity assumptions on the lead time and resource utilization of resource constrained projects. Comput Ind Eng 2008;54:140–54.
Van Peteghem V, Vanhoucke M. A genetic algorithm for the preemptive and non-preemptive multi-mode resource constrained project scheduling problem. Eur J Oper Res 2010;201(2):409–18.
Zhang H, Li H, Tam CM. Particle swarm optimization for resource constrained project scheduling. Int J Project Manage 2006;24(1):83–92.