ACSIJ Advances in Computer Science: an International Journal, Vol. 5, Issue 4, No.22 , July 2016
ISSN : 2322-5157
www.ACSIJ.org

ACSIJ
WWW.ACSIJ.ORG

A bi-population genetic algorithm with two novel greedy mode selection methods for MRCPSP

Siamak Farshidi [1], Koorush Ziarati [2]

[1] University Department, Utrecht University, Utrecht
Utrecht, Netherlands
s.farshidi@uu.nl

[2] University Department, Shiraz University, Shiraz
Shiraz, Iran
ziarati@shirazu.ac.ir

Abstract

The multimode resource-constrained project scheduling problem (MRCPSP) is an extension of the single-mode resource-constrained project scheduling problem (RCPSP). In this problem, each project contains a number of activities which precedence relationship exist between them besides their amount of resource requirements to renewable and non-renewable resources are limited to the resources availabilities. Moreover, each activity has several execution modes, that each of them has its amount of resource requirements and execution duration. The MRCPSP is NP-hard, in addition, proved that if at least 2 non-renewable resources existed, finding a feasible solution for it is NP-complete. This paper introduces two greedy mode selection methods to assign execution modes of the primary schedules' activities in order to balance their resource requirements and thus reduce the number of infeasible solutions in the initialization phase of a bi-population genetic algorithm for the problem. To investigate the usage effect of these greedy methods on the quality of the final results, in addition, to evaluating the performance of the proposed algorithm versus the other meta-heuristics, the instances of the PSPLIB standard library have been solved. The computational results show that by the growth of the problem size, the proposed algorithm reports better results in comparison with the other metaheuristics in the problem literature.

Keywords: Resource-constrained project scheduling, Multi-mode, Genetic algorithm, Makespan, Initialization

1. Introduction

Resource-constrained project scheduling problem (RCPSP) is the generalized version of multi-mode resource constrained project scheduling problem (MRCPSP). In the RCPSP, all activities have only one execution mode, in contrast, each activity in MRCPSP can have several execution modes, which each of them determines the performing duration plus resource requirements of that activity.

An objective function for these class of problems can be the minimization of projects makespan respect to precedence relations between activities, in addition to renewable and non-renewable resource availability constraints. [1] show that, if at least two non-renewable resources in the MRCPSP existed, finding a feasible solution for it is NP-complete. [2] proved that in projects with at least 20 activities and 3 execution modes for each of them, the exact algorithms, like B&B or B&C, cannot find an optimum solution in acceptable time. Therefore, in the recent years, researchers have extended their research in the MRCPSP to heuristics and meta-heuristics area. However, these type of algorithms may not able to obtain global optimum solutions, but their speed of convergence are higher that exact algorithms, hence they could be proper replacements in big or medium size problems. In the last decades, so many heuristic and meta-heuristic algorithms have been proposed, and some of them are introduced in the follows.

[2] proposed a branch and bound algorithm for the MRCPSP, which was limited to time. [3] tested 21 heuristic scheduling rules and suggested a combination of 5 heuristics that have a higher probability of giving the best solution, also in 1996 he introduced a heuristic algorithm based on the critical path method. [1] suggested a local search algorithm that first tried to find a feasible solution and next performed a single neighborhood search on the set of feasible mode assignments. [4] presented a two-phase optimization algorithm that in the first phase an ACO algorithm tried to found a set of feasible mode-assignment candidates, and then in the second phase, an SA algorithm attempted to found a schedule from these mode-assignment candidates. [5] introduced a hybrid method based on PSO algorithm to assign modes to activities and local search optimization to optimize sequences associated to assignments during the evolution of the algorithm. [6] proposed a scatter search algorithm for assignment of different modes to the activities and to optimize the sequence associated with each assignment. [7] developed a hybrid GA to solve the problem. Their algorithm used a mode assignment procedure to maximize the probability of obtaining feasible solutions in the initial population, also it used a fitness function to keep

infeasible solution in its population, and finally by utilizing an improving method attempted to reduce the project makespan. [8] proposed a two-phase genetic local search algorithm that combines a genetic algorithm and a local search method for solving the problem. A set of elite solutions is collected during the first phase, and this set, which acts as the indication of promising areas, is utilized to construct the initial population of the second phase. [9] applied a hybrid rank based evolutionary algorithm to solve the problem. They extend the search space and simplify the evolutionary operators by relaxing non-renewable resource constraints. Furthermore, they introduced a fitness function relying on clustering techniques to promote diversity and avoid premature convergence of the algorithms. [10] proposed a bi-population genetic algorithm, which used of two separate populations and extends the serial schedule generation scheme by introducing a mode improvement procedure that improved the mode selection by choosing that feasible mode of a certain activity that minimizes the finish time of that activity. [11] proposed a scatter search algorithm, which is executed with three different improvement methods, each tailored to the specific characteristics of different renewable and non-renewable resource scarceness values. [12] presented an ALNS-based algorithm for the MRCPSP. He proposed techniques for deriving additional precedence relations and a method for removing modes during execution. These techniques used of bound arguments, also he introduced three bounds for the MRCPSP. [13] developed a shuffled frog-leaping algorithm for solving the problem. They applied priority rules to initialize the population, next they used a two-point crossover and exchanging information during shuffling and partitioning process to evolve the population, and finally they utilized a local search to enhance the exploitation. [14] by combining GA and SA solved the problem. In their algorithm, SA was employed as local search procedure, due to its stochastic neighborhood selection strategy to escape local optima hence a good exploitation strategy, and GA as exploration strategy due to its large number of population. [15] introduced a cooperative discrete particle swarm optimization algorithm for solving the problem. They suggested that, because the positions of particles are discrete values, so they can be updated with crossover and mutation operators. Each particle learns from its past experience and the global experience to balance exploration and exploitation. Moreover, two swarms are separately applied to optimize the two sub-problems: mode assignment problem and activity sequencing problem. Eventually, a merging method used to convert these sub-problems into an integrated problem. [16] applied ACO to solve the problem. In their algorithm, two levels of pheromones were considered with regard to the solution in terms of sequence and mode selection of the activities. Moreover, elitist-rank strategy and non-renewable resource-constraint are incorporated into the updating procedure of the pheromones. [17] presented a hybrid local search technique with EDA to enhance the local search ability. Their local search was based on delete-then-insert operator and a random walk (DIRW) to enhance exploitation abilities of EDA in the neighborhoods of the search space. [18] used a heuristic RCPSP solver and a SAT solver and relies on network transformations that extend the project network and transforms the OR, which specified that only one of the predecessors must be finished before an activity can start, and BI, which specified that two activities cannot be scheduled in parallel, constraints into traditional AND constraints. Thus, the project can be solved by any stochastic project scheduling algorithm without using these logical constraints directly. Their algorithm guaranteed the original precedence logic and is embedded in a meta-heuristic search to resource feasible schedules that respect both the limited renewable resource availability as well as the precedence logic.

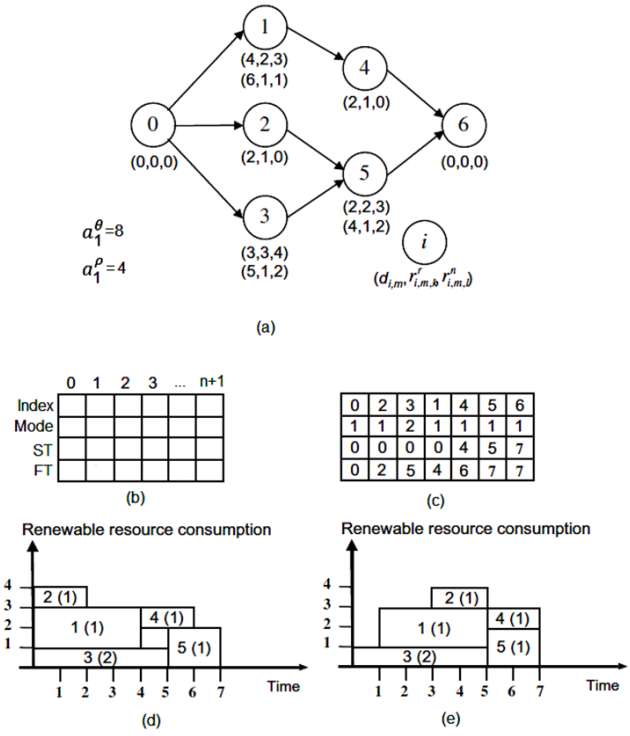

Fig. 1. an instance of an AON graph with a representation of forward schedule besides its backward schedule and their Gantt chart [1].

This paper introduces a Genetic Algorithm with two Greedy mode selection methods (GAG) to solve the MRCPSP. In contrast to regular GA, the purposed algorithm is based on the bi-population approach, as presented by [19] for the RCPSP. Furthermore, in order to balance the amount of resource requirements of each schedule' activities and thus reduce the number of

infeasible solutions in the initial population, two greedy mode selection methods have been employed to assign execution modes to the initial schedules' activities. Moreover, due to evaluate the generated schedules and give improvement chances to the infeasible solutions, the penalty function of [5] has been used. The reminder of the paper is organized as follows: Section 2 describes the general formulation of the problem after that section 3 describes different steps of the algorithm in details. Section 4 shows the effects of different parts of the algorithm on its performance, and finally, the obtained results compared with other meta-heuristics.

2. Problem formulation

The MRCPSP formulated as follows: A set of activities $N = \{0, \dots, n+1\}$ that has to schedule preemptively based on availability of renewable resources $K^{\rho} = \{1, \dots, |K^{\rho}|\}$ and non-renewable resources $K^{\theta} = \{1, \dots, |K^{\theta}|\}$ besides their precedence relations. Each renewable resource $h \in K^{\rho}$ has a constant value of availability a_h^{ρ} in different time periods, while each non-renewable resource $k \in K^{\theta}$ is limited to a fixed value of a_k^{θ} in total project running period. Each activity $j \in N$ performs in execution mode $m \in M_j$ with $M_j = \{1,2, \dots, |M_j|\}$. Moreover, the execution mode m for activity j represents by a triple $(d_{j,m}, r_{j,m,h}^{\rho}, r_{j,m,k}^{\theta})$, which includes predefined values of $d_{j,m}$ as performing duration, $r_{j,m,h}^{\rho}$ units of resource $h \in K^{\rho}$, and $r_{j,m,k}^{\theta}$ units of resource $k \in K^{\theta}$ as the amount of needs of the activity to renewable and non-renewable resources respectively. In set N, 0 and n+1 indices indicate start and end dummy activities of the project, in addition have only one execution mode and their execution duration besides amount of needs to renewable and non-renewable resources are equal to zero units. Suppose that, G (N,P) is an acyclic graph, then the network of a project can be shown as an AON topological order, so that the time log of all activities are equal to zero, and P is the set of pairs of activities which shows a finish-start precedence relationship between them. If schedule S defines by a vector of activities, then it will be feasible, only if respects to all precedence relations of activities and resource limitation of the project. In this paper, the objective function is finding a feasible solution that minimizes the project makespan. [20] introduced the following linear programming to solve the MRCPSP:

$$Min. \quad \sum_{\tau=EST_{n+1}}^{LST_{n+1}} \tau . x_{n+1,1,\tau} \tag{1}$$

$$S.t \quad \sum_{m=1}^{|M_j|} \sum_{\tau=EST_j}^{LST_j} x_{j,m,\tau} = 1 \quad ; j = 1,2, \dots, n \tag{2}$$

$$\sum_{m=1}^{|M_i|} \sum_{\tau=EST_i}^{LST_i} (\tau + d_{i,m}) . x_{i,m,t} \leq \sum_{m=1}^{|M_j|} \sum_{\tau=EST_j}^{LST_j} \tau . x_{j,m,t} \quad ; j = 1,2, \dots, n; i \in P_j \tag{3}$$

$$\sum_{j=1}^{n} \sum_{m=1}^{|M_j|} r_{j,m,h}^{\rho} \sum_{s=max(\tau - d_{j,m}, EST_j)}^{min(\tau-1, LST_j)} x_{j,m,s} \leq a_h^{\rho} \quad ; \forall h \in K^{\rho}; \ \tau = 1,2, \dots, T \tag{4}$$

$$\sum_{j=1}^{n} \sum_{m=1}^{|M_j|} r_{j,m,k}^{\theta} \sum_{\tau = EST_j}^{LST_j} x_{j,m,\tau} \leq a_k^{\theta} \quad ; \forall k \in K^{\theta} \tag{5}$$

$$x_{j,m,\tau} \in \{0,1\} \quad j = 1,2, \dots, n \quad m = 1,2, \dots, |M_j| \quad \tau = EST_j, \dots, LST_j \tag{6}$$

The binary variable $x_{jm\tau}$ is equal to one, when activity j in mode m starts at time τ, otherwise it is equal to zero.

3. Genetic algorithm for the MRCPSP

[21] introduced genetic algorithm (GA), which inspired by evolutionary biology, to solve complicated optimization problems. GA uses natural selection, crossover, and mutation operators to generate individuals of the next generations of the population.

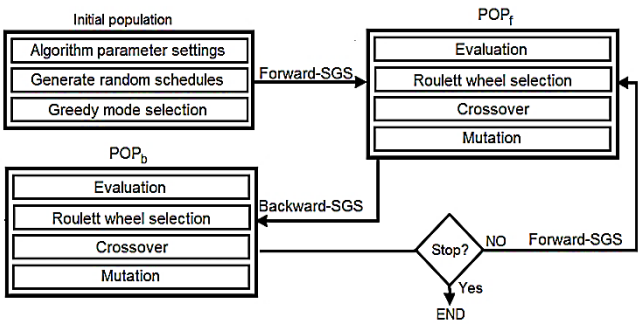

Fig. 2. a conceptual view of the performing steps of the proposed algorithm.

In contrast to the regular genetic algorithm, GAG is based on the bi-population approach, as proposed by [19] for

RCPSP, which one of them contains forward schedules (POP_f) and the other one includes backward schedules (POP_b). After generating initial population randomly, two greedy mode selection methods perform to balance the amount of resource requirements of activities of each schedule and thus reduce the number of infeasible schedules, in this way the primary population of POP_f will be generated. Next, the population individuals will be evaluated, and the genetic operators will be performed on them. After that, the POP_f population by using forward-backward procedure will be converted to POP_b and the genetic operators will be performed on its individuals too. This procedure will be continued until the termination condition meets. Fig. 2, represents a conceptual view of the performing steps of the algorithm.

3.1. Solution representation

Four vectors with equal size used to represent solutions, so that the first vector contains a sequence of activities (Index), and the seconds to third vectors are execution mode (Mode), start time (ST) and finish time (FT) of each activity respectively. Fig. 1b illustrates this four vector representation and Fig. 1c shows a feasible solution instance for the network of Fig. 1a.

3.2. Forward-backward procedure

Forward serial schedule generation scheme (S-SGS) proposed by [22], and it works as follows: it starts at the beginning of the priority list of activities, and schedule them at the earliest possible time with respect to the limitation of the resource existences and their precedence relations. Furthermore, in order to generate a backward schedule, sort activities descending based on their finish time, then schedule them with respect to their reverse precedence relations. After that, to obtain a forward schedule from its backward schedule, first sort activities ascending, then if a gap between start times of the project from zero existed, the start time and finish time of activities are shifted to the left. The alternative conversion of forward schedules to their backward schedules and vice versa called the forward-backward procedure and was proposed by [23] as a local search for RCPSP. For example, Fig. 1d and Fig. 1e draw Gantt charts of a forward schedule and its backward schedule for the network of Fig. 1a.

3.3. Preprocessing

[24] introduced a reduction procedure for reducing search space of the problem, so that in each activity excludes those modes which are inefficient, amount of their resource usage plus execution duration are higher than the other modes, or non-executable, violate the resource constraints. Moreover, this procedure removes redundant non-renewable resources, which the sum of the maximal request for them does not exceed their availabilities.

3.4. Initial population

GAG uses two different populations: population POP_f which contains only forward schedules and population POP_b that includes only backward schedules. Both populations have the same number of POP solutions. First, the algorithm generates the schedules of POP_f randomly, next one of the greedy mode selection methods selects by chance and performs on them in order to balance the resource requirements of activities in each schedule and thus reduce the number of primary infeasible solutions.

$$
\begin{aligned}
&for\ j = 1\ \ to\ \ n\\
&\quad N_{need}^{k} = a_k^{\theta} - \sum_{i=1}^{n}\sum_{m=1}^{|M_i|} r_{i.m.k}^{\theta} \sum_{t=EST_i}^{LST_i} x_{i.m.t}\ \ ;k = 1, \dots, \left|K^{\theta}\right|\\
&\quad if\ (\ (N_{need}^{1} < 0)\ \wedge \dots \wedge (N_{need}^{k} < 0))\\
&\qquad for\ l1 = 1\ \ to\ \ \left|K^{\theta}\right|\\
&\qquad\quad for\ l2 = 1\ \ to\ \ \left|K^{\theta}\right|\\
&\qquad\qquad if\ \left((N_{need}^{l1} < N_{need}^{l2}) \wedge (l1 \neq l2)\right)\\
&\qquad\qquad\quad Mode_j = argmin\left\{r_{j.m.l_1}^{\theta} + r_{j.m.l_2}^{\theta} + \left(\frac{r_{j.m.l_2}^{\theta}}{2}\right)\right\}; m = 1, \dots, \left|M_j\right|\\
&\qquad\qquad else\\
&\qquad\qquad\quad Mode_j = argmin\{r_{j.m.l_1}^{\theta} + r_{j.m.l_2}^{\theta}\}; m = 1, \dots, \left|M_j\right|
\end{aligned}
$$

Algorithm 1: Greedy mode selection procedure number 1 which by selection proper modes for activities balances the amount of usage of non-renewable resources.

$$
\begin{aligned}
&for\ j = 1\ \ to\ \ n\\
&\quad N_{need}^{k} = a_k^{\theta} - \sum_{i=1}^{n}\sum_{m=1}^{|M_i|} r_{i.m.k}^{\theta} \sum_{t=EST_i}^{LST_i} x_{i.m.t}\ \ ;k = 1, \dots, \left|K^{\theta}\right|\\
&\quad if\ (\ (N_{need}^{1} < 0)\ \wedge \dots \wedge (N_{need}^{k} < 0))\\
&\qquad for\ l = 1\ \ to\ \ \left|K^{\theta}\right| - 1\\
&\qquad\quad if\ \left((N_{need}^{l} < N_{need}^{l+1})\right)\\
&\qquad\qquad Mode_j = argmin\left\{\frac{\sum_{m=1}^{|M_j|} r_{j.m.l}^{\theta}}{K} + \frac{\sum_{m=1}^{|M_j|} r_{j.m.k}^{\theta}}{K}\right\}; k = 1, \dots, \left|K^{\theta}\right|\\
&\qquad\quad else\ if\ \left((N_{need}^{l} > N_{need}^{l+1})\right)\\
&\qquad\qquad Mode_j = argmin\left\{\frac{\sum_{m=1}^{|M_j|} r_{j.m.l+1}^{\theta}}{K} + \frac{\sum_{m=1}^{|M_j|} r_{j.m.k}^{\theta}}{K}\right\}; k = 1, \dots, \left|K^{\theta}\right|\\
&\qquad\quad else\\
&\qquad\qquad Mode_j = argmin\left\{\frac{\sum_{m=1}^{|M_j|} r_{j.m.k}^{\theta}}{K}\right\}; k = 1, \dots, \left|K^{\theta}\right|
\end{aligned}
$$

Algorithm 2: Greedy mode selection procedure number 2 which by selection proper modes for activities balances the amount of usage of non-renewable resources

3.5. Greedy mode selection methods

After generating the initial population randomly, the greedy mode selection methods by selecting a proper execution mode for each activity try to balance the amount of usage of the resources with respect to their availabilities, so that the number of infeasible schedules in the primary population will be decreased. Algorithm 1 and Algorithm 2 are the pseudo code of this greedy execution mode selection methods. In fact, both methods attempt to select an execution mode for each activity that does not lead to use non-renewable resources exceedingly and amount of usage of all non-renewable resources become closed to each other. (See section 2)

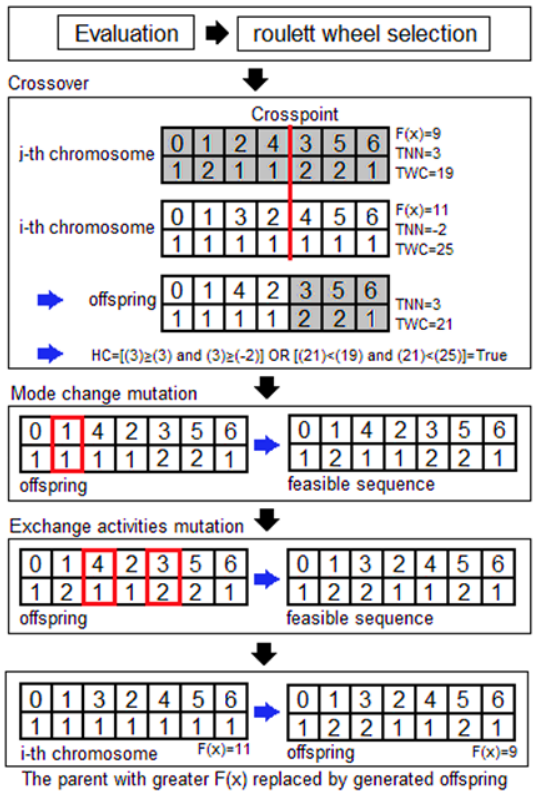

Fig. 3. performing genetic operators on two selected parents, i and j, and generating a new schedule

3.6. Details of the genetic algorithm

In this section, the details of the bi- population genetic algorithm will be discussed.

3.6.1. Evaluation

An infeasible solution is a schedule which violates precedence relations between activities or resource availability constraints. Because of using serial SGS in GAG, generating an infeasible solution which violates precedence relations between activities or uses renewable resources more than their availability in different time periods is impossible. Therefore, the definition of an infeasible solution changes here to a solution which uses non-renewable resources exceedingly. In an initial population of GAG maybe too many infeasible solutions generated, while performing the algorithm they convert to high-quality feasible ones. In consequence, GAG needs a mechanism to keep infeasible solutions in its population, in addition, evaluate their quality, moreover use them to generate and improve the next generations. The penalty function of [5] has been used in the algorithm due to implement mentioned mechanism so that a constant value δ adds to makespan of the schedule per each unit of illegal usage of non-renewable resources. Suppose that, C_{max} is the finish time of the last activity in the i-th schedule in the population, and CP is the value of the critical path method of the project, then the value of the objective function [5] will be calculated by equation (7).

$$Fitness_i = \begin{cases} C_{max} & if\ feasible \\ C_{max} + \sum_{k=1}^{K^\theta} \delta \,.\, max\left(0\,, N_{need}^k\right) & otherwise \end{cases} \tag{7}$$

$$N_{need}^k = \sum_{j=1}^{n} \sum_{m=1}^{|M_j|} r_{j.m.k}^\theta \sum_{t=EST_j}^{LST_j} x_{j.m.t} - a_k^\theta; k = 1, \dots, \left|K^\theta\right|$$

Obviously, the lower fitness value for a schedule means that schedule is more valuable, furthermore the value of function (7) never become lower than the value of the critical path method, which is a lower bound for each project. (See section 2)

3.6.2. Parent Selection

Parent selection is one of the main operators of the genetic algorithm. GAG uses roulette wheel selection to select two parents i and j from the current population, POP_f or POP_b. In this type of selection, schedules with a lower value of the objective function, more valuable schedules, have a higher probability to select.

3.6.3. Crossover operator

Genetic crossover operator performs on two selected parents, i and j, in one-point fashion and generates a child which inherit the attributes of its parents. In the one-point crossover, an integer number "Crosspoint" selected randomly in an interval between 1 and length of activities' vector (index), next all left side data of Crosspoint copy from parent j and the remaining data from parent i copy to the child.

3.6.4. Mutation

Genetic mutation operator applies to create tiny tweaks in the attributes of different individuals of the population. GAG uses two types of mutation operators: exchange activity mutation which tries to exchange the place of two random activities without violating their precedence constraints, in addition, change execution mode mutation which attempts to change execution mode of a random activity to another legal one. Both of these operators perform by Pmut probability. Fig. 3 illustrates an example of performing these genetic operators.

3.6.5. Update

After performing the genetic crossover operator, the promising test on the generated child applies, so that if the test result was true, then activities of the generated child sort based on their precedence constraints and the mutation operators by Pmut probability perform on them, finally calculate the start time and finish time of the activities. Otherwise, if the test result was false, the generated child omits without submitting to the SGS. Equation (8) and (9) calculate the total non-renewable resource usage (TNN) and total work contents of a schedule (TWC) respectively. Furthermore, HC in equation (10) is the promising condition, and its value will be true, if the value of TNN, which is the total request for non-renewable resource, or the value of TWC, which is an estimation of the project makespan, of the generated child, are not more that their values for its parents. In fact, if the amount of needs to non-renewable resources become higher, the probability of being infeasible increases while the amount of needs to renewable resources and the execution time duration of activities become higher, the gap between project makespan and its value of the critical path method increases. (See section 2)

$$'NN = \sum_{k=1}^{|K^{\theta}|} \left(a_k^{\theta} - \sum_{j=1}^{n} \sum_{m=1}^{|M_j|} r_{j.m.k}^{\theta} \sum_{t=EST_j}^{LST_j} x_{j.m.t} \right) \tag{8}$$

$$TWC = \sum_{j=1}^{n} \sum_{h=1}^{|K^{\rho}|} r_{j,m,h}^{\rho} \, d_{jm} \tag{9}$$

$$HC = [(Child_{TNN} \geq Parent1_{TNN}) and (Child_{TNN} \geq Parent2_{TNN})] \quad OR \quad [(Child_{TWC} < Parent1_{TWC}) and (Child_{TWC} < Parent2_{TWC})] \tag{10}$$

Fig. 3 shows an example of performing genetic operators for the network of Fig. 1a. Obviously, two unequal chromosomes (schedule), i and j, are selected by roulette wheel selection, next by applying crossover operator on them, a new child generated, after that because the value of HC was valid, the mutation operators by Pmut probability perform on it, finally the parent with equal or higher value of objective function, means less valuable schedule, than the generated child replaced by it. If the value of the objective function of the parents were equal to each other, one of them replaces by the child randomly. Therefore, the size of the population is always fixed.

4. Computational experiments

We code and run our algorithm in C#.net 2010 on Lenovo G510 with an Intel core i5 2.5 GHz processor and use the standard MRCPSP test dataset J10, J12, J14, J16, J18, J20 and J30 of the well-known PSPLIB. The PSPLIB are generated by the problem generator ProGen designed by [25]. The optimal solutions for J10–20 sets, that have at most 20 activities, are available, however no optimal solutions for J30 set have been found. Each dataset contains 640 instances, some of which are infeasible. Table 1 shows the number of instances in each set that at least have one feasible solution, Fs column, besides the number of activities, Acts row, in each set.
Instances of this dataset have 10 to 30 activities, moreover, each activity has 3 execution modes, which each of them determines execution duration and amount of resource requests of that activity. Furthermore, 2 renewable and 2 non-renewable resources exist for each project, in addition, execution duration of each activity varies between 1 and 10. Suppose that, S is the number of instances in each set that at least have one feasible solution (i.e. S = 547, in J10), $Fitness_i$ and $best_i$ are the value of the objective function (7) and the best result value for i-th instance, respectively, so that, the formula (11) calculates the value of the average percentage of deviation from best known results.

$$Dev = \frac{1}{S} \sum_{i=1}^{S} \frac{Fitness_i - best_i}{best_i} \times 100\% \tag{11}$$

4.1. The training set

A training set contains 700 random instances, which 100 random instances selected from each J10-J30 sets, has been used to investigate the effect of different parts of the algorithm on its final performance, besides due to reduce the impact of chance element on the obtained results, each instances of the training set 10 times sent to the algorithm and the average of these 7000 values from formula (11), whose $best_i$ is equal to the value of the critical path method of the project (lower bound) for the i-th instance, has been reported as the value of F(x). In all of the experiments that have been done on this training set, the termination condition was equal to 500 generated schedules and $T_{CPU(ms)}$ denotes as the average CPU time in

millisecond that the algorithm reached to the termination condition.

Table 1: the number of feasible instances in each set (Fs) besides the number of activities in each set

	J10	**J12**	**J14**	**J16**	**J18**	**J20**	**J30**
Fs	536	547	551	550	552	554	552
Acts	10	12	14	16	18	20	30

Table 2: the impact of the greedy mode selection methods-on the trainig set-(500 generated schedules)

Sch		**RND**	**G1**	**G2**	**G1+G2**
0	Inf	2576	1271	150	143
	F(x)	376.30	220.90	160.85	160.35
500	Inf	446	18	8	0
	F(x)	143.56	27.01	26.72	26.31
	$T_{CPU(ms)}$	868.66	83.02	85.25	89.90

Table 3: the impact of the effect of the mutation and crossover operators-on the trainig set-(500 generated schedules)

	C	M_A	M_M	$M_A + M_M$	**C+** $M_A + M_M$
F(x)	48.93	83.45	27.54	27.32	26.31
$T_{CPU(ms)}$	48.45	43.07	76.39	79.75	89.90

Table 4: the best value of Pmut- on the training set- (500 generated schedules)

P_{mut}	**0.1**	**0.3**	**0.5**	**0.7**	**0.9**
F(x)	36.36	31.81	29.54	28.04	26.31
$T_{CPU(ms)}$	45.53	53.37	56.64	63.70	89.90

4.2. The impact of the mode selection methods

Suppose that, Sch is the number of generated schedules by the algorithm, Inf is the number of infeasible instances which the algorithm couldn't even find a feasible solution for them after reaching to the termination condition. Table 2 shows the effect of using greedy mode selection methods on the performance of the algorithm. (See section 4.1)
Obviously, in the initial population, when Sch is equal to zero, without using the greedy methods and generating the population randomly (RND), the algorithm for more than third of the instances started its search process from an infeasible point, and even after reaching to the termination condition, when Sch is equal to 500, reported 446 still infeasible schedules. The great gap between initial schedules and feasible solutions for the problem has been inherited by the next generations, and thus they omitted just as they created, because of applying the promising condition (10), so didn't submitted to the SGS and caused the increasing the algorithm execution time ($T_{CPU(ms)}$). In contrast, the number of infeasible solutions by using the first greedy procedure (G1) reduced to 18, and by applying the second greedy procedure (G2) reduced to 8, finally by performing both of them (G1+G2) randomly with 0.5 probability reduced to zero, because the algorithm started its search process from feasible or near feasible points and thus the next generations, which produced by the combination of good features of their parents, have higher quality and their deletion ration by the promising condition (10) has been decreased, thus the algorithm terminated faster than RND. As it is obvious in the F(x) rows, by reducing the number of infeasible schedules, the quality of the final results improved and the best results obtained by using the combination of both of the greedy mode selection methods.

4.3. The impact of the genetic operators

Table 3 shows the effect of using crossover and mutation operators on the performance of the algorithm. Suppose that, C shows the only usage of the crossover operator in the genetic algorithm, moreover M_A and M_M mean the only usage of the exchange activities mutation operator and the only applying the change execution mode mutation operator on the chosen parent with lower value of the objective function (more valuable), respectively. (See section 4.1)
Table 3 shows that the worst case obtained by using M_A, because the precedence relation between activities constraints limit their exchange extremely, so the place of an activity can't change with the other one easily. In the other side tiny tweaks in execution mode of activities can lead to big changes in amount of usage of resources or project makespan, therefore the usage of M_M had the most effect among applying single operators. Furthermore, using C reported better results than M_A, because it tried to produce a higher quality child than its two parents by combining their attributes. In fact, the GA by applying only mutations, in single operators or the combinatorial one ($M_A + M_M$), degrades to a local search, so stick into a local minimum, also by using only crossover operator acts like a random search. Ultimately by observing ($M_M + M_A + C$) column, which demonstrates the using of all three proposed operators, concludes that the best results obtained by balancing the exploration and exploitation. Notice that except the last column ($M_M + M_A + C$), which the promising condition (10) performed after execution of the crossover operator and before execution of the mutation operators, in the other columns the promising condition performed just after execution of the single or combinatorial operators.
The computational results in table 4 show that the best value for Pmut, the probability of performing mutation operators, is equal to 0.9. The high value of Pmut means the frequent usage of the mutation operators which leads to increase computational time besides exploitation in the algorithm. In contrast to the other genetic algorithm in the literature, like MHGA [7], TGLS [8], PVGA [10] and CGA [14], in GAG did not used any local search procedure to tweak execution modes of activities during running the GA in order to improve the quality of the individuals, therefore to have more exploitation we

increased the probability of performing of the genetic mutation operators. Finally, the comparison of $T_{CPU(ms)}$ and F(x) rows in both tables 3 and table 4 shows that the results quality level have a direct influence on the deletion rate of the promising condition (10) thus the algorithm execution time. In the other word, when the quality of feasible solutions raise, the competition between parents and their children for survival increase too, and thus the average computational time of the algorithm will be higher.

4.4. Determination of the population size

The large population size voids its homogeneity, also increases the computational time, and in the case of time limitation generates only a few schedules, consequently restricts the performance of the GA too. In the other hand, small population size prevents the evolutionary performance of the algorithm. Therefore, we need to determine the population size relative to the size of the problem. When configuring the algorithm, we have found out that the number of activities of each project besides the population size have a direct influence on the performance of the algorithm. Moreover, we have noticed the number of activities in each project is negatively related to the population size, if the number of activities in project increases, the performance of the algorithm will increase by decreasing the population size. Similar results were found in [19], [26], [27] and [10]. A nonlinear least squares regression based on the best population size values for the different number of activities by the formula (14) can be calculated, in which POP is the population size and N is the number of activities. The both population, POP_f and POP_b, have the same number of individuals POP, so the total population size in the algorithm is equal to $2 \times POP$.

$$POP = \left\lfloor \frac{e^{1.999+\frac{19.3}{N}}}{2} \right\rfloor \quad (12)$$

4.5. Comparison with other meta-heuristics

In the literature of the MRCPSP, the maximum number of generated schedules and the maximum CPU time consumed are mostly used as the stopping conditions for comparison. According to the literature, we used the following three kinds of stopping conditions: (1) 5000 generated schedules; (2) 1 s CPU time; (3) 0.15 s CPU time per activity. We run the GAG 10 times independently for each instance. The statistical results are reported in Tables 5, 6 and 7, respectively. Moreover in the following tables Opt_{var} , Opt_{max}, Opt_{min} and Opt_{avg} columns denote variance, maximum, minimum and average percentage of instances which optimum solutions have been found in each set respectively, besides $Dev_{var}, Dev_{max}, Dev_{min}, Dev_{avg}$ columns mean variance, maximum, minimum and average percentage of deviation of the algorithm' results from the best results, which is optimum values for J10 to J20 sets and the critical path method values as lower bound of projects for J30 set, that are calculated by the formula (11) respectively, also Fs ,$T_{CPU(S)}$ and Set columns denote the number of instances in each set that at least have one feasible solution, the average CPU time in seconds that the algorithm reached to the termination condition, and the sets of the PSPLIB standard library, respectively. In the mentioned tables, the FS column illustrates that all obtained results were feasible, moreover the values of Dev_{var} column, that are the variances for all the problem sets, are very small, which shows that the GAG is very stable. In addition, the values of the $T_{CPU(S)}$ column demonstrates that the average computational time of the algorithm increases as the problem size increases too.

Next, we compare the GAG with some existing algorithms based on the PSPLIB to further show the effectiveness of the GAG. The compared algorithms include the hybrid algorithm of ACO and SA [4] denoted as TAS, the combinatorial PSO algorithm [5] denoted as CPSO, the scatter search algorithm [6] denoted as PSS, the hybrid GA [7] denoted as MHGA, the a two-phase genetic local search algorithm [8] denoted as TGLS, the hybrid rank based evolutionary algorithm [9] denoted as RBEA, the genetic algorithm [10] denoted as PVGA, the scatter search algorithm [11] denoted as PVSS, the ALNS-based algorithm [12] denoted as ALNS, the shuffled frog-leaping algorithm [13] denoted as SFLA, the hybrid algorithm of GA and SA [14] denoted as CGA, the cooperative discrete particle swarm optimization algorithm [15] denoted as CDPS, the ACO [16] denoted as ACO, and the hybrid local search technique with EDA [17] denoted as SEDA.

Table 8, 9 and 10 compare the obtained results of GAG with other algorithms based on 5000 generated schedules, 1 seconds, and 0.15 second per each activity as termination conditions. Furthermore, the one-tailed t-test is adopted to explain whether the GAG is significantly better than the compared metaheuristics. We denote Dev_{avg} of our algorithm as μ and Dev_{avg} of the compared algorithm, which has the closest value to the GAG, as μ_0. Suppose the null hypothesis (H_0) is equal to $\mu \geq \mu_0$, and the alternative hypothesis (H_1) is $\mu < \mu_0$. The t statistic can be calculated as follows:

$$t = \frac{\mu - \mu_0}{\left(\frac{s}{\sqrt{n}}\right)} \quad (13)$$

That n is the number of independent running tries of the GAG, and $s = \sqrt{Dev_{var}}$ is the value of standard deviation of samples which calculated based on the

Dev_{var} in tables 5 through 7. According to the obtained value of formula (13), we can find the equivalent $p-value$ from the corresponding lookup table of the $t-test$ that is the probability to make a mistake if H_1 accepted as true. All the $p-value$ of the applying $t-test$ on results of GAG and Dev_{avg} of the compared algorithm, which has the closest value to it, are listed in the last rows of tables 8 through 10.

Each schedule assigns a start time to each activity of the project by SGS. The number of generated schedules calculates based on the number of times which each activity of the project gets a feasible start time divides by the number of the project activities [28]. In contrast to the other genetic algorithm in the literature, like MHGA [7], TGLS [8], PVGA [10] and CGA [14], in GAG did not used any local search procedure to tweak execution modes of activities during running the GA in order to improve the quality of the individuals, so that after performing crossover operator and assurance of promising the new child by condition (10), mutation operators used. Moreover, in order to have more exploitation in the algorithm, we increased the probability of performing of the genetic mutation operators. In addition, if the quality of the generated child become lower than its parents ($Hc = false$), it will be removed immediately without sending to the SGS. Obviously in the table 8, which compared the algorithms by the 5000 generated schedules as termination condition, the results of GAG are extremely similar to the scatter search algorithm of [11] and their results gap increased by the growth of the problem size, finally GAG represented better performance, while the GAG started its search process from feasible or near feasible points by performing greedy mode selection methods, and thus it ignore a huge search space in initialization time, moreover by pruning its non-promising children with condition (10) during search process tries to avoid deviation of improvement path besides prevent going to the ineffective search areas. Similar results obtained in table 9 and 10, based on the reported values of the $p-value$ in each column, so that by increasing the number of activities in the problem not only the average percentage of deviation from the best known results in our algorithm are lower significantly than the other metaheuristics but also the average percentage number of problems that solved optimally are greater significantly than the others. Obviously in the table 10, because of the termination condition, 150 milliseconds per each activity, which provided more opportunity to execution for the algorithm, in all data sets except J12 the GAG had better performance. According to the above comparisons between the GAG and the other metaheuristics, it can be concluded that our proposed GAG is effective in solving the MRCPSP.

Table 5: GAG results-on the PSPLIB datasets-(5000 generated schedules)

Set	Dev_{avg}	Dev_{min}	Dev_{max}	Dev_{var}	$T_{CPU(S)}$	Fs	Opt_{avg}	Opt_{min}	Opt_{max}	Opt_{var}
J10	0.002	0.000	0.009	1.47E-5	0.025	100	99.94	99.81	100	7.31E-7
J12	0.037	0.028	0.053	6.98E-5	0.064	100	99.20	98.90	99.45	2.14E-6
J14	0.094	0.065	0.155	6.19E-4	0.140	100	97.46	95.82	98.36	4.48E-5
J16	0.149	0.111	0.192	5.49E-4	0.208	100	96.18	95.27	97.09	2.64E-5
J18	0.218	0.169	0.260	7.61E-4	0.303	100	94.11	93.11	95.28	3.95E-5
J20	0.277	0.238	0.328	1.05E-3	0.577	100	92.09	90.61	93.32	9.11E-5
J30	13.421	13.374	13.507	1.31E-3	1.512	100	n/a	n/a	n/a	n/a

Table 6: GAG results-on the PSPLIB datasets-(1 second)

Set	Dev_{avg}	Dev_{min}	Dev_{max}	Dev_{var}	$T_{CPU(S)}$	Fs	Opt_{avg}	Opt_{min}	Opt_{max}	Opt_{var}
J10	0.005	0.000	0.019	4.96E-5	1	100	99.88	99.62	100	2.23E-6
J12	0.048	0.025	0.080	2.76E-4	1	100	98.92	98.35	99.26	8.32E-6
J14	0.152	0.107	0.175	4.47E-4	1	100	95.95	95.09	97.27	3.36E-5
J16	0.225	0.197	0.276	6.60E-4	1	100	94.07	92.90	94.72	4.18E-5
J18	0.414	0.362	0.473	1.45E-3	1	100	89.47	88.04	90.94	1.24E-4
J20	0.584	0.504	0.625	1.27E-3	1	100	85.09	83.93	87.00	9.07E-5
J30	14.826	14.739	14.969	4.05E-3	1	100	n/a	n/a	n/a	n/a

Table 7: GAG results-on the PSPLIB datasets-(0.15 s CPU time per activity)

Set	Dev_{avg}	Dev_{min}	Dev_{max}	Dev_{var}	$T_{CPU(S)}$	Fs	Opt_{avg}	Opt_{min}	Opt_{max}	Opt_{var}
J10	0.000	0.000	0.000	0.00E-0	1.8	100	100	100	100	0.00E-0
J12	0.034	0.014	0.048	1.10E-4	2.1	100	99.25	98.90	99.63	4.31E-6
J14	0.082	0.072	0.096	5.13E-5	2.4	100	97.77	97.27	98.00	3.99E-6
J16	0.100	0.079	0.156	4.39E-4	2.7	100	97.25	95.81	97.81	2.74E-5
J18	0.172	0.129	0.220	7.67E-4	3.0	100	95.18	93.65	96.37	6.51E-5
J20	0.266	0.219	0.328	8.54E-4	3.3	100	92.53	91.15	93.68	4.84E-5
J30	13.262	13.171	13.345	2.16E-3	4.8	100	n/a	n/a	n/a	n/a

Note: The optimal values of J30 are not available.

Table 8: Comparison with the other metaheuristics- average percentage of gap with the best results (11)/ average percentage of obtained optimum results – (5000 generated schedules as termination condition)

	Dev_{avg} (%)							Opt_{avg} (%)					
	J10	J12	J14	J16	J18	J20	J30	J10	J12	J14	J16	J18	J20
TAS	0.16	0.21	0.97	1.40	1.40	1.98	-	96.01	93.89	82.19	84.18	81.42	76.20
CPSO	0.03	0.09	0.36	0.44	0.89	1.10	-	99.25	98.47	91.11	85.91	79.89	74.19
PSS	0.05	0.11	0.38	0.52	0.84	1.03	-	99.07	98.35	93.66	87.66	83.33	79.24
MHGA	0.08	0.27	0.31	0.50	0.69	0.98	-	99.01	96.53	92.92	90.00	84.96	80.32
TGLS	0.33	0.52	0.93	1.08	1.32	1.69	18.33	95.16	90.57	82.03	77.39	73.38	66.66
RBEA	0.14	0.24	0.77	0.91	1.30	1.62	-	97.31	94.94	83.18	77.27	70.78	63.16
PVGA	0.01	0.09	0.22	0.32	0.42	0.57	13.75	99.63	98.17	94.56	92.00	88.95	85.74
PVSS	***0.00***	***0.02***	***0.08***	*0.15*	*0.23*	*0.32*	13.66	***100***	***99.45***	*97.28*	***96.73***	***94.75***	*87.73*
ALNS	0.05	0.20	0.55	0.78	1.14	1.52	15.96	99.01	95.59	88.75	84.76	78.75	72.87
SFLA	0.10	0.21	0.46	0.58	0.94	1.40	*13.46*	97.92	95.97	90.86	86.49	79.43	72.84
CGA	0.16	0.31	0.63	0.49	0.52	1.04	-	97.39	93.95	83.48	82.36	74.23	68.91
CDPS	0.05	0.21	0.46	0.82	1.21	1.62	-	98.99	95.94	90.41	82.61	75.49	70.92
ACO	0.09	0.13	0.40	0.57	1.02	1.10	-	98.30	96.50	90.30	86.90	76.10	72.40
SEDA	0.09	0.12	0.36	0.42	0.85	1.09	-	97.62	97.11	92.31	91.05	82.46	77.57
GAG	0.00	0.03	0.09	**0.14**	**0.21**	**0.27**	**13.42**	99.94	99.20	**97.46**	96.18	94.11	**92.09**
P-value	0.964	0.999	0.946	0.463	0.116	0.001	0.000	0.965	0.999	0.209	0.995	0.994	0.000

Note: the closet value to the results of GAG denoted as italic and the best results in each column denoted as bold.

Table 9: Comparison with the other metaheuristics- average percentage of gap with the best results (11)/ average percentage of obtained optimum results – (1 second CPU time as termination condition)

	Dev_{avg} (%)						Opt_{avg} (%)					
	J10	J12	J14	J16	J18	J20	J10	J12	J14	J16	J18	J20
RBEA[a]	0.09	0.13	0.43	0.46	0.67	0.91	98.6	97.3	90.00	88.90	84.10	78.52
SFLA[b]	*0.03*	***0.03***	***0.13***	***0.21***	*0.46*	*0.91*	*99.22*	***99.12***	***96.64***	***94.44***	*88.46*	*80.42*
GAG[c]	**0.005**	0.04	0.15	0.22	**0.41**	**0.58**	**99.88**	98.92	95.95	94.07	**89.47**	**85.09**
P-value	0.000	0.996	0.995	0.953	0.002	0.000	0.000	0.991	0.997	0.947	0.009	0.000

Note: the closet value to the results of GAG denoted as italic and the best results in each column denoted as bold.

[a] Pentium 3.00 GHz
[b] T7500 2.2 GHz
[c] Intel 2.5 GHz

Table 10: Comparison with the other metaheuristics- average percentage of gap with the best results (11)/ average percentage of obtained optimum results – (0.15 second per activity CPU time as termination condition)

	Dev_{avg} (%)						Opt_{avg} (%)					
	J10	J12	J14	J16	J18	J20	J10	J12	J14	J16	J18	J20
RBEA	0.03	0.05	0.26	0.25	0.45	0.58	99.25	98.72	93.93	93.09	88.22	84.81
SFLA	*0.03*	***0.02***	*0.08*	*0.13*	*0.28*	*0.57*	*99.37*	***99.48***	***97.99***	*96.73*	*92.50*	*86.55*
GAG	**0.000**	0.03	**0.08**	**0.10**	**0.17**	**0.26**	**100**	99.25	97.77	**97.25**	**95.18**	**92.53**
P-value	0.000	0.999	0.434	0.000	0.000	0.000	0.000	0.996	0.996	0.005	0.000	0.000

Note: the closet value to the results of GAG denoted as italic and the best results in each column denoted as bold.

5. Conclusion

The multimode resource-constrained project scheduling problem (MRCPSP) is an extension of the single-mode resource-constrained project scheduling problem (RCPSP). In this problem, each project contains a number of activities which precedence relationship exist between them besides their amount of resource requirements to renewable and non-renewable resources are limited to the resources availabilities. Moreover, it is proved that in projects that contain at least 20 activities and 3 execution modes per each activity, exact algorithms, like B&B and B&C, cannot find an optimum solution in acceptable execution time. The MRCPSP is NP-hard, in addition, proved that if at least 2 non-renewable resources existed, finding a feasible solution for it is NP-complete. In this paper, we have introduced two greedy mode selection methods to assign execution mode to the initial schedules' activities. By the aim of these greedy methods can generate feasible or near feasible schedules, thus, can use

them in the initialization phase of population-based algorithms. In order to show the efficiency of these greedy methods, we have implemented them in the initialization phase of a bi-population genetic algorithm to solve the MRCPSP. In addition, our GA used an effective penalty function to preserve infeasible schedules in the population and give improvement chance to them, also by pruning non-promising children, which generated during performing the algorithm, tried to reduce search space. The computational results show the well performance of the proposed algorithm by increasing the number of activities in the projects in comparison with the other meta-heuristics in the literature.

References

[1] Kolisch R, Drexl A (1997) Local search for non-preemptive multi-mode resource constrained project scheduling. IIE Transactions 29:987–999

[2] Spreche A, Drexl A (1998) Multi-mode resource constrained project scheduling by a simple, general and powerful sequencing algorithm. European Journal of Operational Research 107:431–450, DOI 10.1016/S0377 2217(97)00348-2

[3] Boctor F (1993) Heuristics for scheduling projects with resource restrictions and several resource-duration modes. International Journal of Production Research 31:2547–2558, DOI 10.1080/00207549308956882

[4] Ling C (2004) A two-phase hybrid optimization for solving multi-mode resource-constrained project scheduling problems. Specializes in teaching 96:257–270

[5] Jarboui B, Damak N, Siarry P, Rebai A (2008) A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. Applied Mathematics and Computation 195:299–308

[6] Pourghaderi A, Torabi S, Talebi J (2008) Scatter search for multi-mode resource constrained project scheduling problems. In: Industrial Engineering and Engineering Management, IEEM 2008. IEEE International Conference, pp 163–167, DOI 10.1109/IEEM.2008.4737852

[7] Lova A, Tormos P, Cervantes M, Barber F (2008) A hybrid genetic algorithm for the multi-mode resource constrained project scheduling problem. In: Eleventh International Workshop, PMS 2008, pp 189–192

[8] Tseng L, Chen S (2009) Two-phase genetic local search algorithm for the multi-mode resource-constrained project scheduling problem. IEEE Transactions on Evolutionary Computation 13:848–857, DOI 10.1109/TEVC.2008.2011991

[9] Elloumi S, Fortemps P (2010) A hybrid rank-based evolutionary algorithm applied to multi-mode resource constrained project scheduling problem. European Journal of Operational Research 205:31–41

[10] Peteghem V, Vanhoucke M (2010) A genetic algorithm for the preemptive and nonpreemptive multi-mode resource constrained project scheduling problems. European Journal of Operational Research 201:409–418

[11] Peteghem V, Vanhoucke M (2011) Using resource scarceness characteristics to solve the multi-mode resource constrained project scheduling problem. Journal of Heuristics 17:705–728, DOI 10.1007/s10732-010-9152-0

[12] Muller L (2011) An adaptive large neighborhood search algorithm for the multimode rcpsp. DTU Management Engineering 3:25

[13] Wang L, Fang C (2011) An effective shuffled frog-leaping algorithm for multi-mode resource-constrained project scheduling problem. Computers and Operations Research 181:4804–4822, DOI 10.1016/j.ins.2011.06.014

[14] Bilolikar V, Jain K, Sharma M (2012) An annealed genetic algorithm for multi-mode resource constrained project scheduling problem. International Journal of Computer Applications 60(1):36–42

[15] Shen H, Li X (2013) Cooperative discrete particle swarms for multimode resource-constrained projects. In: IEEE 17th International Conference on Computer Supported Cooperative Work in Design, pp 31–36, DOI 10.1109/CSCWD.2013.6580935

[16] Li H, Zhang H (2013) Ant colony optimization-based multi-mode scheduling under renewable and non-renewable resource constraints. Computers and Operations Research 35:431–438

[17] Soliman O, Elgendi E (2014) A hybrid estimation of distribution algorithm with random walk local search for multi-mode resource-constr ined project scheduling problems. International Journal of Computer Trends and Technology 8:57–64

[18] Coelho J, Vanhoucke M (2015) An approach using sat solvers for the rcpsp with logical constraints. European Journal of Operational Research DOI 10.1016/j.ejor.2015.08.044

[19] Debels D, Vanhoucke M (2005) A bi-population based genetic algorithm for the resource-constrained project scheduling problem. Lecture Notes on Computer Science 3483:378–387

[20] Talbot F (1982) Resource-constrained project scheduling with timeresource tradeoffs: The nonpreemptive case. Management Science 28:1197 1210

[21] Holland J (1975) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. University of Michigan Press

[22] Kelley J (1963) The critical-path method: Resources planning and scheduling. In: Industrial Scheduling. Prentice-Hall, New Jersey, pp 347–365

[23] Li K, Willis R (1992) An iterative scheduling technique for resource-constrained project scheduling. European Journal of Operational Research 56:370–379

[24] Sprecher A, Hartmann S, Drexl A (1977) An exact algorithm for project scheduling with multiple modes. OR Spectrum 19:195–203

[25] Kolisch R, Sprecher A (1997) Psplib - a project scheduling problem library: Or software-orsep operations research software exchange program. European Journal of Operational Research 96:205–216, DOI 10.1016/S0377 2217(96)00170-1

[26] Hartmann S (2001) Project scheduling with multiple modes: a genetic algorithm. Annals of Operations Research 102:111–135

[27] Alcaraz J, Maroto C, Ruiz R (2003) Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. Journal of the Operation Research Society 54:614–626

[28] Kolisch R, Hartmann S (2006) Experimental investigation of heuristics for resource constrained project scheduling. European Journal of Operational Research 174:23–37