International Journal of Industrial Engineering Computations 2 (2011) 45–60

Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

Artificial Bee colony for resource constrained project scheduling problem

Reza Akbari[a*], Vahid Zeighami[b] and Koorush Ziarati[a]

[a]*Department of Computer Science and Engineering, Shiraz University, Shiraz, Iran*
[b] *Department of Mathematics , School of Science, Shiraz University, Shiraz, Iran*

A R T I C L E I N F O

Article history:
Received 15 April 2010
Received in revised form
19 July 2010
Accepted 20 July 2010
Available online 20 July 2010

Keywords:
Meta-heuristic
Artificial bee colony
Resource constrained project scheduling
Makespan
Single mode

A B S T R A C T

Solving resource constrained project scheduling problem (RCPSP) has important role in the context of project scheduling. Considering a single objective RCPSP, the goal is to find a schedule that minimizes the makespan. This is NP-hard problem (Blazewicz et al., 1983) and one may use meta-heuristics to obtain a global optimum solution or at least a near-optimal one. Recently, various meta-heuristics such as ACO, PSO, GA, SA etc have been applied on RCPSP. Bee algorithms are among most recently introduced meta-heuristics. This study aims at adapting artificial bee colony as an alternative and efficient optimization strategy for solving RCPSP and investigating its performance on the RCPSP. To evaluate the artificial bee colony, its performance is investigated against other meta-heuristics for solving case studies in the PSPLIB library. Simulation results show that the artificial bee colony presents an efficient way for solving resource constrained project scheduling problem.

1. Introduction

Resource constrained project scheduling is known as an important problem in project scheduling. The RCPSP is an optimization problem which tries to find the optimum ordering of the activities to achieve some predefined objectives. It is possible to have many different objectives such as makespan, robustness, etc which are depended on some predefined goals (Abbasi et al., 2006). The makespan minimization, which is referred to as finding the minimum time to complete the entire project, is the most common objective in RCPSP. Also, RCPSP has several varieties so-called single-mode RCPSP (Ranjbar, 2008), multi-mode RCPSP (Damak et al., 2009), RCPSP with non-regular objective functions (Neumann et al., 2003), stochastic RCPSP (Rabbani et al., 2007; Ashtiani et al., 2009), Bin-packing related RCPSP (Fekete & Schepers, 1998), and multi-RCPSP (Krüger & Scholl, 2009). These varieties of RCPSP along with different possible objectives provide a wide area of research. Hence, scheduling resource constrained project has been the subject of extensive researches in the recent years.

In this work, we focus on solving single-mode RCPSP with makespan minimization objective. A large variety of methods ranging from exact to meta-heuristic have been proposed for single-mode RCPSP. We refer to the survey about the solution techniques provided by Hartman and Briskorn

* Corresponding author Tel +98-917-1003767 Fax +98-711-6271747.
E-mail addresses: rakbari@cse.shirazu.ac.ir (R. Akbari),

doi: 10.5267/j.ijiec.2010.03.004

(2010). The proposed methods can be classified as exact, heuristics and meta-heuristics. Exact methods are the first class of RCPSP solvers. Several exact methods have been proposed by authors to solve the RCPSP. The methods proposed by Stork and Uetz (2005), Sprecher (2000), and Mingozzi et al. (1998) are among the most representative exact methods. Although exact methods provide efficiency to solve small-sized instances of the RCPSP, they may not able to solve large-size instances of the RCPSP in a reasonable computational time. Therefore, we need to use heuristics or meta-heuristics when solving large problem instances. These approaches provide optimal or near optimal solutions for the problem at hand. The second class of RCPSP solvers is designed based on heuristics. The methods presented by Tormos and Lova (2001), Kolisch (1996), and Boctor (1990) are examples of the heuristic approaches. The methods of this class start with an empty schedule (i.e. none of the activities has been scheduled). After that, the empty schedule is filled by selecting a subset of activities in each step and assigning the earliest possible starting times to these activities by considering the priority rules and scheduling scheme. This process is continued until all the activities have been considered.

In scheduling process, the activities are selected based on their ranks, and the priority rules are used for ranking the activities. Meta-heuristics are used to design the third class of RCPSP solvers. Several meta-heuristic methods such as Tabu-Search (TS) (Tomas & Salhi, 1998), Simulated Annealing (SA) (Boctor, 1996; Bouleimen & Lecocq, 1993), Genetic Algorithm (GA) (Valls et al., 2008; Mendes et al., 2009), Scatter Search (Mobini et al., 2009), Electromagnetism (EM) (Debels & Vanhoucke, 2004), Immune Algorithm (IA) (Mobini et al. , 2010), Filter and Fan (FF) (Ranjbar, 2008), and Particle Swarm Optimization (PSO) (Chen et al., 2010; Zhang et al., 2005; Zhang et al., 2006) have been proposed by authors. The meta-heuristics approaches can be divided in two main sub-classes. The first sub-class containing approaches such as tabu search and simulated annealing maintain only one solution at each cycle of the algorithm. These methods try to find a new solution with better quality from the current solution iteratively. The second sub-class of meta-heuristics containing population based approaches such as genetic algorithm, ant colony optimization, particle swarm optimization, etc, maintain a set of solutions at each cycle of the algorithm. These approaches solve the RCPSP by employing an initial population of individuals each of which represents a candidate schedule for the project. Then, they evolve the initial population by successively applying a set of operators on the old solutions to transform them in to the new solutions.

Some other approaches for resolving RCPSP problem are designed as a hybrid among different approaches. These methods try to exploit the advantages of two or more methods in order to design an algorithm with better performance. ANGEL (Tseng & Chen, 2006), ACOSS (Chen et al., 2010), Neurogenetic (Agrawal et al., 2010), Scatter Search-FBI (Debles et al., 2006), and Hybrid-GA (Valls et al., 2008) are among most representative hybrid methods presented in literature. Usually in hybrid algorithms, better performance is obtained by improving the balance between exploration and exploitation through combination of the potentials of different approaches. Hence, one can use hybridization as a way to avoid stagnation and premature convergence which are known as the main deficiencies of meta-heuristic approaches. TS, SA, GA, ACO, PSO, SS, EM, IA, FF, and hybrid methods are among the most popular meta-heuristic approaches applied on RCPSP. Previous studies showed that these methods were successful and more efficiency has been obtained by applying them on RCPSP. Methods such as artificial bee colony (Karaboga & Basturk, 2007) and bee swarm optimization (Akbari et al., 2010) which are inspired from intelligent behavior of honey bees are among newly introduced meta-heuristic approaches which have been successfully applied on optimization problems. It seems that they may be useful for solving RCPSP. In this work, we use artificial bee colony (ABC) and present a method to solve a single-mode RCPSP. The proposed method tries to minimize the makespan of a single-mode RCPSP using meta-heuristics inspired from collective behavior of honey bees. It starts by a set of initial schedules and tries to improve them cycle by cycle until the termination condition is met. Each cycle represents an asynchronous process that involves four steps. These steps are used to transform the old schedules into the new schedules

with shorter makespans. This paper is organized as follows. Section 2 presents the foundations of the bee algorithms by describing self-organizing and collective behaviors of honey bees in nature. Section 3 explains the formulation of the RCPSP. In Section 4, we explain the steps of artificial bee colony and its application for solving single-mode RCPSP. Section 5 illustrates the numerical examples and reports the comparison results. Finally, Section 6 concludes this work and represents some future research directions.

2. Bees in the Nature

The collective behaviors of honey bees provide them with the ability to perform complex tasks as reported by Teodorovic and Dell Orco (2007), Teodorovic et al. (2006), Pham et al. (2008), and Akbari et al. (2010), using relatively simple rules of individual bees' behaviors. Collecting, processing, and advertising of nectars are examples of intelligent behaviors of honey bees (Teodorovic et al., 2006). These behaviors help a colony in finding the flower patches in the environment. Information about flower patches provided by the employed bees is shared among other bees when they return to the hive. The employed bees share their information using waggle dance on the dance floor. The provided information by employed foragers is shared with a probability proportional to the profitability of the food source. So, the onlooker bees which are waiting in the hive can employ a probabilistic approach to choose an employed bee among numerous dancers and adjust their search trajectories toward the most profitable sources. The onlooker bees choose more profitable food sources with greater probabilities. Hence, more profitable food sources attract more onlooker bees. After choosing the food source by the onlooker bee, she flies to find the food source. When she finds the food source, the onlooker bee switches its type to the employed bee. The employed bee memorizes the location of food source and then starts exploiting it. Then she takes a part of nectar from the food source, it returns to the hive and saves the nectar in a food area in the hive. After saving the food, the bee enters to the decision making process and selects one of the following three options. A bee may select one of these options to switch its type based on different information such as quality of food source, its distance from the hive, its direction, and ease of extracting the food source. 1) If the nectar amount is decreased to a low level or it is exhausted, she abandons the food source and switches its type as scout. Scout bees fly spontaneously around the hive and search for new food sources without any knowledge about the environment. 2) If there are still sufficient amount of nectar in the food source, it can continue to forage without recruiting the nestmates. 3) She can perform waggle dance to inform the nestmates about the same food source. After that she recruits the nestmates before returning to the food source. The waggle dance mechanism and the search behavior of honey bees were used by Karaboga and Bastruk (2007) to design an optimization algorithm called artificial Bee colony (ABC). It seems that ABC has competitive performance compared to other meta-heuristics such as ACO, PSO, GA, etc as reported by Karaboga and Bastruk (2007), Karaboga and Akay (2009), Alatas B. (2010) and Pan et al. (2010). Hence, this work aims at adapting ABC algorithm as a newly developed meta-heuristic to resolve the resource constrained project scheduling problem.

3. Formulation of the RCPSP

Considering the limited capacity of the resources, the main concern in the resource constrained project scheduling problem is to assign jobs or activities to a set of resources in order to meet some predefined objectives. Although, many different objectives are possible, but minimizing the makespan is commonly considered in most of the RCPSPs. This work aims at designing an approach for minimizing the makespan in a single mode scheduling problem. In a single mode RCPSP, each project has a single execution mode in which both the activity duration and its requirements for a set of resources are assumed to be fixed. The single-mode resource constrained project scheduling is defined as follows: assume that we have a project that involves $n+1$ activities where each activity has to be processed in order to complete the project. The project can be modeled as a directed graph $G(A,C)$ (see Fig. 1 (a)) where the nodes in the graph correspond to activities and the arcs specify precedence relationships. The graph represents a set $A=\{a_0,a_1,....,a_{n+1}\}$ of dummy and non-dummy

activities with the associated durations $D = \{d_0, d_1, ..., d_{n+1}\}$. The dummy activities 0 and $n+1$ (with durations $d_0 = d_{n+1} = 0$) represent the 'project start' and the 'project end', respectively. As mentioned earlier the arcs in the graph specify the precedence constraint. If an arc (i, j) appears in the graph, it means that the activity j cannot be started before its immediate predecessor activity i has been finished. Besides, an activity requires resources with limited capacities to be performed. An activity may be scheduled if both of the constraints (i.e. precedence constraint and resource limitation constraint) are satisfied.

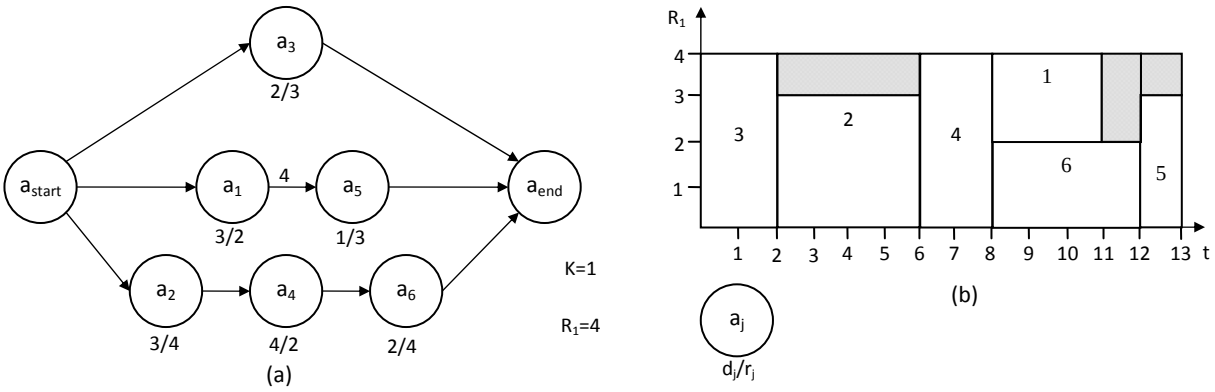


Fig. 1. An example for resource constrained project scheduling problem. The project comprising 6 activities that needs to be scheduled subject to $K = 1$ renewable resource type with a capacity of 4 units. (a) presents the precedence graph of the activities, and (b) presents the corresponding optimal schedule of the activities

We have a set of resource types $R = \{R_1, R_2, ..., R_K\}$ where each resource type k has a limited capacity of R_k at any point of the time. During running of the project, each activity a_j requires $r_{j,k}$ units of resource type k during every time instant of its non-preemptable duration d_j. For the 'project start' and 'project end' activities we have $r_{j,k} = 0 \quad \forall j \in \{0,1,..., n+1\}, \forall k \in \{1,2,..., K\}$. The objective of the RCPSP is to find an ordering of the activities (see Fig. 1 (b)) that minimizes the makespan of the schedule F_{n+1} subject to the following constraints:

$$F_h \le F_j - d_j, \quad j = 1,..., n+1; h \in P_j, \tag{1}$$

$$\sum_{j \in A(t)} r_{jk} \le R_k, \quad k \in \{1,2,..., K\}; t \ge 0, \tag{2}$$

where F_j is the finishing time of activity a_j, P_j is a set of preceding activities (or predecessors of activity a_j), $A(t) = j \in V \mid F_j - d_j \le t < F_j$ and $F_j \ge 0, \quad J = 1,..., n+1$ describe the constraints of decision variables. The equation (1) enforces the precedence constraints between activities, and equation (2) enforces the resource limitation constraint.

4. Artificial Bee Colony and Its Application for RCPSP

This section presents the detailed description of the artificial bee colony algorithm to solve RCPSP. Fig. 2 presents the ABC algorithm in pseudocode. ABC algorithm uses the priority-based representation (Zhang et al., 2006) for its individuals. Each bee represents a position in the search space. If the project has *n* activities, the bees will fly in the search space with *n* dimensions. A position is a candidate for a priority list where each of its elements fixedly represents an activity and

its corresponding value shows the priority of that activity. Hence, the position vector $\vec{x}_i$ of each bee *i* is used to represent the priority values of a schedule *i* with *n* activities. Each element *d* of the position vector $\vec{x}_i$ is located between 0 and 1 (i.e. $0 \le x_{id} \le 1$). Hence, each element with a value larger than 1 or smaller than 0 is set to 1 or 0, respectively.

Algorithm ABC (*Population size*, *Scouts*, *Max_Trial*, *Prj*)

Initialization
  Define *FoodNumber = Population size*/2
  For *i* = 1 **to** *FoodNumber*
    Initialize food source *i* randomly
    $\text{Trial}_i = 0$
  End For
Repeat
  For *i* = 1 **to** *FoodNumber*
    Evaluate food source *i* using serial-SGS
  End For
  (Send Employed Bees)
    For *i* = 1 **to** *FoodNumber*
      Select a parameter *d* randomly
      Select Neighbor *k* randomly
      Calculate $v_{id} = x_{id} + \omega_1 r_{id}(x_{id} - x_{kd})$
      Evaluate new food source using serial-SGS
      If the new food source presents a schedule with smaller makespan
        Update the position
      If the food source has not been improved
        Increment its Trial by 1
    End For
  (**Send Onlooker Bees)**
    Calculate probabilities for each food source using equation (5)
    For *i* = 1 **to** *FoodNumber*
      Select a parameter *d* randomly
      Select Neighbor *k* from food sources based on equation (5)
      Calculate $v_{id} = x_{id} + \omega_2 r_{id}(x_{id} - x_{kd})$
      Evaluate new food source using serial-SGS
      If the new food source presents a schedule with smaller makespan
        Update the position
      If the food source has not been improved
        Increment its Trial by 1
    End For
  (Send Scout Bees)
    Define *i* as the food source with the maximum Trial
    Initialize food source *i* randomly
    $\text{Trial}_i = 0$
Until termination condition is met

Return *best schedule*

Fig. 2. Pseducode of the ABC algorithm for RCPSP

The ABC employs a population of different types of bees to find the schedule with minimum makespan. The type of a bee is defined based on the behavior she uses to find the food sources. A bee waiting on the dance area for making decision to choose a food source is called onlooker bee; the bee which goes to the food source already visited by herself just before is named as employed bee, and the bee which flies spontaneously in the search space is called scout bee. The ABC uses the following steps to find a schedule with minimum makespan:

Step 1 (Initialization): ABC receives a set of parameters as inputs: population size (*Population size*), number of scouts (*Scouts*), *Max_Trial*, and project (*Prj*). *Max_Trial* is the parameter used to identify the food sources that should be abandoned. At initialization step, the number of food sources (*FoodNumber*) will be set to half of Population size, and the population is equally subdivided as employed bees and onlookers. Next food sources will be initialized randomly. *Trial* is the parameter used to be incremented when a food source is not optimized in two consecutive cycles, and *Prj* is the project to be scheduled.

Step 2 (Bee evaluation): At the start of each cycle, all the food sources need to be evaluated. To evaluate the fitness of a food source, we need to generate the schedule from the priority list. Hence, we need to use a schedule generation scheme (SGS). We use serial-SGS that constructs active schedules (Kolisch & Hartmann, 1999). The serial-SGS is an activity oriented scheme that generates a schedule in *n* stages from the priority list. Serial SGS uses two disjoint activity sets at each stage $s \in \{1,2,...,n\}$: the set of scheduled activities and the set E_s of eligible activities (i.e. all activities for which all predecessors are scheduled). In each stage, serial-SGS select one eligible activity $j \in E_s$ and schedules it at the earliest precedence and resources feasible time. Next, the set of eligible activities and the resource profiles of partial schedule are updated.

Step 3 (Position updating): After all the bees are evaluated, each employed bee *i* selects another employed bee as its own neighbor. After that, a parameter $d \in \{1,2,...,n\}$ will be selected randomly. Each food source will be optimized through following equation,

$$v_{id} = x_{id} + \omega_1 r_{id}\left(x_{id} - x_{kd}\right), \tag{3}$$

where *i* represents the food sources which is going to be optimized, $k \in \{1,2,...,FoodNumber\}$ and $d \in \{1,2,...,n\}$ is a randomly chosen index. Although *k* is determined randomly, it has to be different from *i*. The random number r_{id} is selected in range of [-1, 1]. Parameter ω_1 controls the production of neighbor food sources around x_{id} and represents the comparison of two food positions visually by a bee. As can be seen from equation (3), as the difference between the parameters of the x_{id} and x_{kd} decreases, the perturbation on the position x_{id} is decreased, too. Thus, as the search process approaches the optimum solution in the search space, the step length is adaptively reduced. If a parameter value produced by this operation exceeds its predetermined limit, the parameter can be set to an acceptable value. After the employed bees explore the new areas of the food sources, they come into the hive and share the nectar information of the sources with the onlooker bees waiting on the dance area. Sharing the information in the hive, an onlooker bee only needed to employ a decision making process to select one of the food sources advertised by the employed bees. For this purpose, the probability for each food source *k* advertised by the corresponding employed bee will be calculated as follows,

$$p_k = \frac{fit\left(\vec{x}_k\right)}{\sum_{m=1}^{FoodNumber} fit\left(\vec{x}_m\right)}, \tag{4}$$

where $fit(\vec{x}_m)$ is the probability of the food source proposed by the employed bee *k* which is proportional to the quality of the food source. The quality depends on the makespan of the schedule proposed by the food source *k* and calculated using following equation,

$$fit(\vec{x}_m) = \frac{1}{makespan_m}, \tag{5}$$

where $makespan_m$ is the value of the makespan proposed by the food source *m*. After calculating the probabilities, each onlooker bee employs the roulette wheel to choose a food source advertised by the employed bee *k* based on its probability. By selecting a food source, the onlooker bee updates its position using the following equation if the newly discovered food source proposes a schedule with smaller makespan than the old one,

$$v_{id} = x_{id} + \omega_2 r_{id}(x_{id} - x_{kd}), \tag{6}$$

where parameter ω_2 controls the importance of the social knowledge provided by the employed bees. Under this probabilistic approach, the food sources with better schedules attract more onlooker bees. At each cycle of the algorithm, the positions are evaluated and if a food source cannot be improved after a predetermined number of iterations (called *Max_Trial*), then the corresponding food source is abandoned. The *Max_Trial* parameter is determined manually. In this work, the value of *Max_Trial* is set to 5. The abandoned food source is replaced with the new one founded by the scouts. A scout produces a new position randomly and replaces the abandoned food source if the new food source has better nectar. Assume that the abandoned source is x_i and $j \in \{1,2,...,n\}$, then the scout discovers a new food source to be replaced with x_i. This operation can be defined as follows,

$$v_{id} = r_{id}, \tag{7}$$

where r_{id} is selected in the range of [0,1] randomly. After each candidate source position v_{ij} is produced and evaluated by the artificial bee, its performance is compared with that of its old one. If the new food source proposes a schedule with smaller makespan than the old one, it is replaced with the new one in the memory. Otherwise, the old one is retained in the memory.

Step 4 (Termination): By termination of the ABC algorithm, the schedule with minimum makespan obtained by the population is returned as the output.

5. Computational Experiments

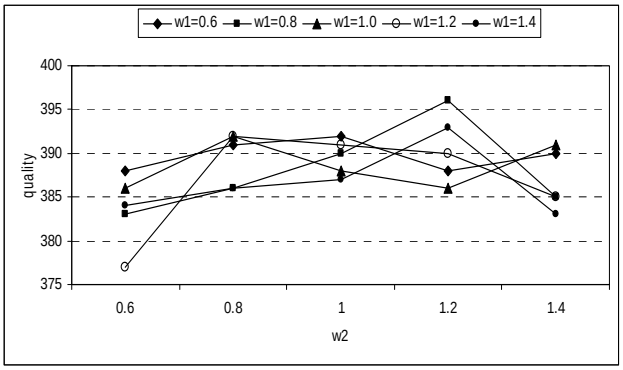
This section presents the experiments conducted to investigate the performance of ABC and the other algorithms on RCPSP datasets in PSPLIB. We have used several problem instances which are successfully solved by an algorithm as a measure for performance comparison. This measure differs from the other measure so-called *average deviation from the optimal solution* used in literature for performance comparison. Hence, we need to implement the investigated meta-heuristic approaches. To investigate the performance of our ABC-based algorithm[1], we implement some of the most representative meta-heuristics for solving RCPSP problems in java: ant colony optimization (ACO) (Chen et al., (2006)), genetic algorithm (GA) (Hartmann, 1998), standard particle swarm optimization (PSO) (Zhang et al., 2005), PSO+ (Chen et al., 2010), OOP-GA (Montoya-Torres et al., 2010), GAPS (Mendes et al., 2009), ANGEL (Tseng & Chen, 2006), and ACOSS (Chen et al., 2010), Neurogenetic (Agrawal et al., 2010). The experiments were executed on a Core 2 Duo 1.66 GH Pentium. We have used well-known scheduling case studies from PSPLIB to evaluate performance of the algorithms. PSPLIB involves three case studies j30, j60, and j90 that consist of 480 problem instances with four resource types and 30, 60, and 90 activities, respectively. Also PSPLIB involve j120 case study that

[1] The source code of ABC-based algorithm may be obtained by e-mailing to *rakbari@cse.shirazu.ac.ir*

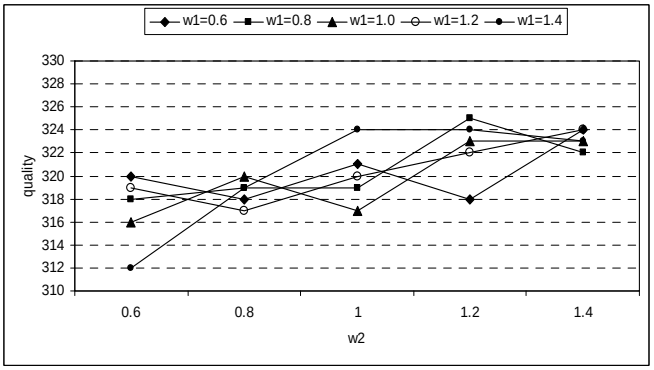
consists of 600 problem instances with four resource types and 120 activities. We have tested ten approaches under the following configurations.

5.1. Experimental Settings

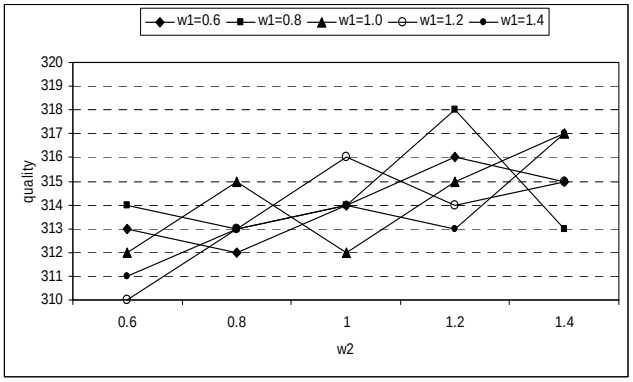
In our experiments, each algorithm is configured under parameters values which result the best performance. In this section we specify these suitable parameter values. For the proposed ABC-based algorithm, parameters such as coefficients ω_1 and ω_2, population size, number of iterations, and *Max_Trial* influence the performance of this algorithm. To determine the suitable parameter values, we conduct two experiments to study the effects of the ABC parameters while solving problem instances of the j30, j60, j90, and j120 case studies. Our empirical studies have shown that *Max_Trial* has not significant effect on the performance of our algorithm. Hence, we exclude it from our analysis. In the first experiment, the performance of the proposed algorithm is studied under different values of the coefficients ω_1 and ω_2. These parameters vary from 0.6 to 1.4 with the step size of 0.2. The population size and the iteration number are set to 100 and 50, respectively. Fig. 3 shows the effect of coefficients ω_1 and ω_2 on the performance of our algorithm. The vertical axis shows the number of problem instances which are successfully solved by our algorithm, and the horizontal axis shows the coefficient ω_2. The results show that these two parameters have the positive effect on the performance of the algorithm. The best results are obtained for $\omega_1 = 0.8$ and $\omega_2 = 1.2$. Our empirical study have shown that the quality of the algorithm decreases when both the parameters are set to values larger than 1.4. Also, the quality of the algorithm decreases when ω_1 has a small value and ω_2 has a large vale. Hence, we recommend using the proposed algorithm under following configuration: $0.7 \le \omega_1 \le 0.9$ and $1.1 \le \omega_2 \le 1.3$.



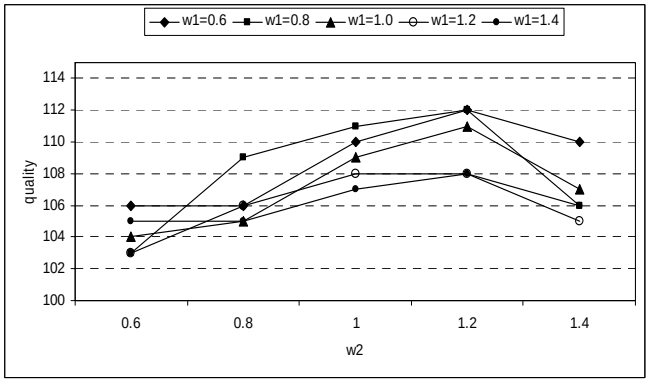
(a) j30 case study



(b) j60 case study



(c) j90 case study



(d) j120 case study

Fig. 3. The effect of coefficients ω_1 and ω_2 on the performance of the ABC algorithm

We have conducted the second experiment to observe if the performance of our algorithm under fixed number of schedules is affected by the number of iterations or population size. Here, both the values of the population size and the number of iterations are varied from 10 to 250 subject to the following constraint: the number of produced schedules is fixed at 2500. The population size varies in steps of 10, and for each of them the corresponding number of iterations computed as $\left\lceil 2500/pop\_size \right\rceil$. The first experiment shows that the best results are obtained for $\omega_1 = 0.8$ and $\omega_2 = 1.2$, hence we use these values. Table 1 shows the effect of population size and the number of iterations on the performance of our algorithm. The results show that the quality of our algorithm is relatively affected by these two parameters. The percentage of problem instances successfully solved by our algorithm varies in range of [78.13%, 78.75%], [66.20%, 67.29%], [64.80%, 65.42%], and [17.67%, 18.50%] for j30, j60, j90, and j120 case studies, respectively. The success rate implies that although one can obtain better result by fine tuning the number of iteration and size of the population, the rate of improvement is not significant under fixed number of schedules. Hence we can say that the proposed algorithm provides stability in solving RCPSP under fixed number of schedules. As a result, the parameters of the ABC algorithm are set as follows:

For the ABC algorithm, the population is equally subdivided into the employed and onlooker bees and one individual is selected as scout bee. The parameter ω_1 and ω_2 are respectively set to 0.8 and 1.2. The value of *Max_Trial* is set to 5 manually. The population size is set at 100, and each case study was tested 15 trials. Other algorithms' parameters are set as follows:

The parameters of ACO algorithm are set as: $\tau_0 = 0.5$, $q_0 = 0.9$, $q_1 = 0.9$, $\alpha = 1$, $\beta = 1$, $c = 10$, $\delta = 0.1$, and $\rho = 0.1$. For genetic algorithm, the mutation probability is set to 0.4, and two-point crossover is used.

For particle swarm optimization, the maximum and minimum values of inertia weight (i.e. $w_{\max}$ and $w_{\min}$) are set to 0.9 and 0.4, respectively. The linear inertia weight is used here where the inertia weight linearly decreases from $w_{\max}$ to $w_{\min}$ throughout iterations. The particles are positioned randomly in range of [0,1] at initial time, and the initial velocity of each particle is set to 0. The acceleration coefficients c_1 and c_2 are set to 0.85.

Table 1
The effect of population size and number of iterations on the performance of the proposed ABC algorithm

Population size		10	20	30	40	50	60	70	80	90	100
#iterations		250	125	83	63	50	42	36	31	28	25
Success rate	j30	78.54%	78.54%	78.33%	78.75%	78.75%	78.54%	78.54%	78.55%	78.13%	78.13%
	j60	67.29%	67.09%	66.45%	66.25%	66.62%	66.25%	66.62%	66.58%	66.45%	66.25%
	j90	65.42%	65.63%	65.42%	65.63%	65.21%	65.00%	65.21%	64.80%	65.21%	64.80%
	j120	18.50%	17.67%	18.50%	18.33%	18.00%	17.83%	17.83%	18.17%	18.33%	17.67%

The PSO+ is tested under the following configuration: the inertia weight w and the learning factors c_1 and c_2 are set to 0.7, and the parameters q_0 and q' are set to 0.05 and 0.95, respectively. Two-point crossover is used for OOP-GA, and the crossover and the mutation probabilities are set to 0.7, and 0.1, respectively. For the GAPS, the following parameters are considered: the crossover probability is set to 0.7, the top 15% from the previous population chromosomes are copied to the

next generation, and the bottom 20% of the population chromosomes are replaced with randomly generated chromosomes. The ACOSS method is tested based on the following control parameters: decay factor ρ is set to 0.02, the control parameters α, β are set to 1 and 2.5, respectively, and the pheromone trail limits are selected as the way reported by Chen et al. (2010). For the Neurogenetic method, the learning rate is set to 0.05, the weights are initialized at 1, two-point crossover is used for its GA part and mutation probability is set to 0.5. The number of interleaving is set to 5, the proportion of GA is taken as 90%, and the number of GA solutions to feed NN is used as four.For the ANGEL method, the parameters Loop_limit and Generation_limit are set to 3 and 5, respectively. Other parameters are set as: $\alpha = 0.9$, $\rho = 0.1$, $\Delta_{ini} = 0.00001$, $P_{cro} = 0.75$, and $P_{mutl} = 0.05$. We have used two stopping criteria in our experiments. An algorithm stops if the founded solution is equal to the lower bound which the critical path calculated without resource constraints or if a predetermined number of maximum of iterations are reached. In our experiments, the results are obtained for 10, 50, and 500 iterations.

5.2. Comparative Study

The following experiments were conducted to see how many cases of PSPLIB library can be solved by the proposed algorithm. We say that a case study is solved if the algorithm finds optimal solution or lower bound solution for that case study. Tables 2-5 present the experimental results for the j30, j60, j90, and j120 case studies. Each cell of a table indicates the percentage of the problem instances which are successfully solved by an algorithm.

Table 2
The results of using GA, PSO, ACO, ANGEL, GAPS, OOP-GA, PSO+, ACOSS, Neurogenetic and ABC for j30 cases study

	Number of iterations		
Method	10	50	500
GA (Hartmann, 1998)	51.87%	53.75%	58.96%
PSO (Zhang et al., 2005)	53.54%	58.13%	61.26%
ACO (Chen et al., 2006)	57.50%	60.63%	63.55%
ANGEL(Tseng & Chen, 2006)	71.46%	78.70%	89.11%
GAPS (Mendes et al., 2009)	57.30%	63.29%	68.33%
OOP-GA (Montoya-Torres et al., 2010)	53.19%	57.48%	65.82%
PSO+ (Chen et al., 2010)	67.80%	75.05%	87.15%
ACOSS(Chen et al., 2010)	**77.41%**	**85.04%**	**93.27%**
Neurogenetic(Agrawal et al., 2010)	74.13%	81.33%	91.05%
ABC (This paper)	72.71%	80.84%	90.42%

Table 2 presents the results of our approach and the other meta-heuristics approaches for j30 case study after predetermined number of iterations. For this case study, the results show that ACOSS has better performance than other approaches. The ANGEL, Neurogentic, and ABC provide competitive

results to ACOSS approach. The ABC approach obtains the third rank on j30 case study after 10, 50, and 500 iterations. Table 3 summarizes the results of our approach and the other meta-heuristics approaches for j60 cases study after predetermined number of iterations. The best results for the problem instances of this case study are found by ABC algorithm. From the results we can see that the performance of the algorithms decreases as the number of activities increases. Table 4 demonstrates the experimental results of all 480 instances for j90 cases study with 90 activities after 10, 50, and 500 iterations. The PSO+ approach surpass other algorithms for 10 and 50 iterations. The second rank was obtained by ABC approach for 10 and 50 iterations. However, similar to j60 case study, ABC approach outperforms other algorithms after 500 iterations. The last case study with 120 activities is the most difficult case study to solve. Table 5 summarizes the results of our approach and the other meta-heuristics approaches for this case study after predetermined number of iterations. The results show that the approaches have the least performance on this case study compared to other ones. The ABC approach provides schedules with better qualities for j120 case study.

Table 3
The results of using GA, PSO, ACO, ANGEL, GAPS, OOP-GA, PSO+, ACOSS, Neurogenetic, and ABC for j60 cases study

	Number of iterations		
Method	10	50	500
GA (Hartmann, 1998)	43.33%	52.30%	55.84%
PSO (Zhang et al., 2005)	45.21%	53.96%	58.55%
ACO (Chen et al., 2006)	56.88%	58.75%	61.46%
ANGEL(Tseng & Chen, 2006)	53.95%	61.33%	64.91%
GAPS(Mendes et al., 2009)	46.39%	52.40%	56.07%
OOP-GA (Montoya-Torres et al., 2010)	48.52%	54.04%	59.26%
PSO+ (Chen et al., 2010)	58.66%	62.05%	66.76%
ACOSS(Chen et al., 2010)	55.12%	60.94%	65.21%
Neurogenetic(Agrawal et al., 2010)	56.37%	63.86%	68.35%
ABC (This paper)	**61.88%**	**67.09%**	**71.88%**

In our experiments, we used 2040 problem instances from four categories of case studies. To view the overall performance of the proposed algorithm, its ability in solving all the problem instances is considered. Fig. 4 presents the average percentage of the problem instances which are successfully solved by the algorithms after 10, 50, and 500 iterations. From the results, it can be seen that the proposed algorithm surpasses the PSO+, ANGEL, ACOSS, and Neurogenetic algorithms and successfully outperforms the other ones.

One property of the figure is that as the number of iterations increases the gap between performance of the proposed algorithm and the other ones increases too. In general, we can see that ABC algorithm provides an efficient way for solving RCPSP problems. The overall performance shows

that ABC outperforms other meta-heuristics investigated in this paper. This happens due to the ability of ABC algorithm in providing better diversity throughout the execution of the algorithm. Providing appropriate level of diversity helps the algorithm to alleviate the deficiencies of meta-heuristic algorithm such as stagnation and premature convergence and consequently provide the ability to explore further regions of the search space to find better solutions.

Table 4
The results of using GA, PSO, ACO, ANGEL, GAPS, OOP-GA, PSO+, ACOSS, Neurogenetic, and ABC for j90 cases study

	Number of iterations		
Method	10	50	500
GA (Hartmann, 1998)	39.95%	44.30%	53.96%
PSO (Zhang et al., 2005)	42.63%	49.34%	55.27%
ACO (Chen et al., 2006)	54.25%	57.92%	59.80%
ANGEL(Tseng & Chen, 2006)	53.08%	58.23%	61.49%
GAPS (Mendes et al., 2009)	45.85%	48.72%	54.21%
OOP-GA (Montoya-Torres et al., 2010)	47.44%	50.58%	56.84%
PSO+ (Chen et al., 2010)	**61.65%**	**65.24%**	67.17%
ACOSS(Chen et al., 2010)	55.72%	58.10%	62.03%
Neurogenetic(Agrawal et al., 2010)	57.64%	60.53%	64.82%
ABC (This paper)	60.13%	65.21%	**68.75%**

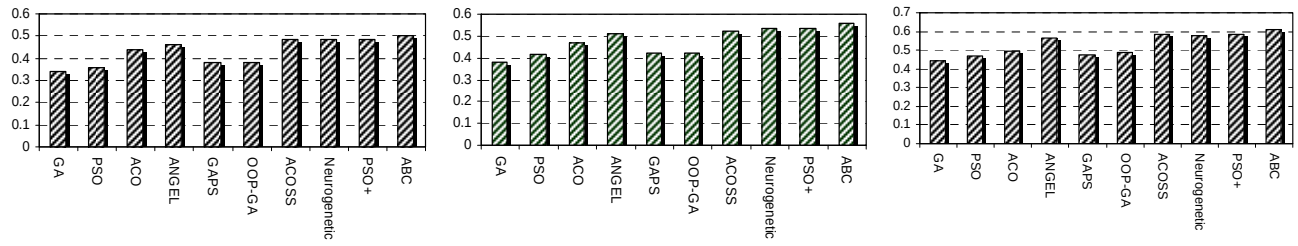


Fig. 4. Overall performance of the ABC and the other algorithms investigated in this paper

Table 5
The results of using GA, PSO, ACO, ANGEL, GAPS, OOP-GA, PSO+, ACOSS, Neurogenetic, and ABC for j120 cases study

	Number of iterations		
Method	10	50	500
GA (Hartmann, 1998)	7.15%	9.18%	16.33%
PSO (Zhang et al., 2005)	9.44%	11.89%	20.42%
ACO (Chen et al., 2006)	14.67%	17.67%	21.17%
ANGEL(Tseng & Chen, 2006)	14.85%	15.36%	19.91%
GAPS (Mendes et al., 2009)	10.12%	12.83%	18.78%
OOP-GA (Montoya-Torres et al., 2010)	10.39%	13.51%	19.82%
PSO+ (Chen et al., 2010)	14.60%	16.48%	21.26%
ACOSS(Chen et al., 2010)	14.56%	17.72%	20.69%
Neurogenetic(Agrawal et al., 2010)	14.32%	16.85%	21.08%
ABC (This paper)	**15.34%**	**18.97%**	**22.84%**

6. Conclusions

In this paper we have considered the performance of the artificial bee colony meta-heuristic on resolving the single-mode resource constrained project scheduling problem. The ABC-based meta-heuristic starts with a set of initial schedules and tries to improve them cycle by cycle by applying four-step strategy as described in the paper. We have evaluated the performance of ABC strategy on PSPLIB case studies against other meta-heuristics. Our experimental results prove that ABC provides an efficient way for solving RCPSP. Moreover, the better performance can be obtained using ABC strategy for large-sized case studies. The competitive results obtained by the ABC-based meta-heuristic on solving RCPSP may encourage one to study alternatives for improving the performance of ABC-based approach as a newly emerged meta-heuristics.

Acknowledgment

The authors would like to gratefully thank the anonymous referees for their constructive comments on earlier version of this work.

References

Abbasi, B., Shadrokh, S., & Arkat, J.(2006). Bi-objective resource-constrained project scheduling with robustness and makespan criteria. *Journal of Applied Mathematics and Computation*, 180, 146–152.

Agarwal, A., Colak, S., & Erenguc, S.(2010). A Neurogenetic approach for the resource-constrained project scheduling problem. *Computers & Operations Research*, doi:10.1016/j.cor.2010.01.007.

Akbari, R., Mohammadi, M., & Ziarati, K.(2010). A novel bee swarm optimization algorithm for numerical function optimization. *Journal of Communications on Nonlinear Sciences and Numerical Simulation*, 15, 3142-3155.

Ashtiani, B., Leus, R., & Aryanezhad, M. B.(2009). New competitive results for the stochastic resource-constrained project scheduling problem: exploring the benefits of pre-processing. *Journal of Scheduling*, doi: 10.1007/s10951-009-0143-7.

Alatas B.(2010). Chaotic bee colony algorithms for global numerical optimization. *Expert Systems with Applications*, 37, 5682-5687.

Blazewicz J., Lenstra J. K., & Rinnooy Kan A. H. G.(1983). Scheduling projects to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5, 11–24.

Boctor, F. F.(1990). Some efficient multi-heuristic procedures for resourceconstrained project scheduling. *European Journal of Operational Research*, 49, 3–13.

Boctor, F. F.(1996). An adaptation of the simulated annealing algorithm for solving resource-constrained project scheduling problems. *International Journal of Production Research*, 34, 2335–2351.

Bouleimen, K., & Lecocq, H.(1993). A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research*, 149, 268–281.

Chen, R. M., Wu, C. L., Wang, C. M., & Lo, S. T.(2010). Using novel particle swarm optimization scheme to solve resource-constrained scheduling problem in PSPLIB. *Expert Systems with Applications*, 37, 1899–1910.

Chen, R. M., Lo, S. T., Wang, C. J., & Wu, C. L.(2006). Multiprocessor system scheduling with precedence and resources constraints by ant colony system. *Proceeding of ICS Conference*, 292-297.

Chen, R. M., Wu, C. L., Wang, C. M., & Lo, S. T.(2010). Using novel particle swarm optimization scheme to solve resource-constrained scheduling problem in PSPLIB. *Expert Systems with Applications*, 37, 1899–1910.

Chen, W., Shi, Y. J., Teng, H. F., Lan, X. P., & Hu, L. C.(2010). An efficient hybrid algorithm for resource-constrained project scheduling. *Information Sciences*, 180, 1031–1039.

Damak, N., Jarboui, B., Siarry, P., & Loukil, T.(2009). Differential evolution for solving multi-mode resource-constrained project scheduling problems. *Computers & Operations Research*, 36, 2653 – 2659.

Debels, D., & Vanhoucke, M.(2004). An Electromagnetism Meta-Heuristic For The Resource-Constrained Project Scheduling Problem. *Lecture Notes on Computer Science*, 3871, 259-270.

Debels, D., De Reyck, B., Leus, R., & Vanhoucke M.(2006). A hybrid scatter search /Electromagnetism meta–heuristic for project scheduling. *European Journal of Operational Research*, 169, 638-653.

Fekete, S. P., & Schepers, J.(1998). New classes of lower bounds for bin-packing problems. *Lecture Notes in Computer Science*, 1412, 257–270.

Karaboga, D., & Basturk, B.(2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization*, 39, 459–471.

Karaboga, D., & Akay, B.(2009). A comparative study of Artificial Bee Colony algorithm. *Applied Mathematics and Computation*, 214, 108-132.

Kolisch, R., & Hartmann, S.(1999). Heuristic algorithms for solving the resource-constrained project scheduling problem: classification and computational analysis. J. Weglarz (Ed.), *Project Scheduling: Recent Models, algorithms and Applications, Kluwer Academic Publishers*, Berlin, 147–178.

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

Krüger, D., & Scholl, A.(2009). A heuristic solution framework for the resource constrained multi-project scheduling problem with sequence-dependent transfer times. *European Journal of Operational Research*, 197, 492–508.

Hartmann, S., & Briskorn, D.(2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207, 1-14.

Hartmann, S.(1998). A competitive genetic algorithm for resourceconstrained project scheduling. *Naval Research Logistics*, 45, 733– 750.

Mahdi Mobini, M. D., Rabbani, M., Amalnik, M. S., Razmi, J., & Rahimi-Vahed, A. R.(2009). Using an enhanced scatter search algorithm for a resource-constrained project scheduling problem. *Soft Computing*, 13, 597–610.

Mendes, J. J., Gonalves, J. F., & Resende M.G.C.(2009). A random key based genetic algorithm for the resource constrained project scheduling problem. *Computers & Operations Research*, 36, 92–109.

Mendes, J. J., Gonalves, J. F., Resende, M. G. C.(2009). A random key based genetic algorithm for the resource constrained project scheduling problem. *Computers & Operations Research*, 36, 92–109.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., & Bianco, L.(1998). An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation. *Journal of Management Science*, 44, 714–729.

Mobini, M., Mobini Z., & Rabbani M.(2010). An Artificial Immune Algorithm for the project scheduling problem under resource constraints. *Applied Soft Computing*, doi:10.1016/j.asoc.2010.06.013.

Montoya-Torres, J. R., Gutierrez-Franco, E., & Pirachica N-Mayorga, C.(2010). Project scheduling with limited resources using a genetic algorithm. *International Journal of Project Management*, 28, 619–628.

Neumann, K., Schwindt, C., & Zimmermann, J.(2003). Order-based neighborhoods for project scheduling with nonregular objective functions. *European Journal of Operational Research*, 149, 2, 325-343.

Pan, Q. K., M. Tasgetiren F., Suganthan, P. N., & Chua, T. J.(2010) A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem, *Information Sciences*. doi:10.1016/j.ins.2009.12.025.

Pham, D. T., Castellani, M., & Fahmy, A. A.(2008). Learning the inverse kinematics of a Robot manipulator using the bees algorithm. *IEEE international conference on industrial informatics*, 493–498.

Rabbani, M., Fatemi Ghomi, S.M.T., Jolai, F., & Lahiji, N.S.(2007). A new heuristic for resource-constrained project scheduling in stochastic networks using critical chain concept. *European Journal of Operational Research*, 176, 794–808.

Ranjbar, M.(2008). Solving the resource-constrained project scheduling problem using filter-and-fan approach. *Journal of Applied Mathematics and Computation*, 201, 313–318.

Sprecher, A.(2000). Scheduling resource-constrained projects competitively at modest memory requirements. *Management Science*, 46, 710–723.

Stork, F., & Uetz, M.(2005). On the generation of circuits and minimal forbidden sets. *Mathematical Programming*, 102, 185–203.

Teodorovic, D., & Dell Orco, M.(2007). Bee colony optimization–a cooperative learning approach to complex transportation problems. *Advanced OR and AI Methods in Transportation*, 51–60.

Teodorovic, D., Panta, L., Goran M., & Dell, O. M.(2006). Bee colony optimization: principles and applications. Proceeding of eighth seminar on neural network applications in electrical engineering, *Neurel*, 151–156.

Thomas, P., R., & Salhi S.(1998). A tabu search approach for the resource constrained project scheduling problem. *Journal of Heuristics*, 4, 123–139.

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

Tseng, L.Y., & Chen, S. C.(2006). A hybrid metaheuristic for the resource-constrained project scheduling problem, *European Journal of Operational Research*, 175, 707–721.

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

Zhang, H., Li, X., Li, H., & Huang, F.(2005). Particle swarm optimization-based schemes for resource-constrained project scheduling. *Journal of Automation in Construction*, 14, 393– 404.

Zhang, H., Li, H., & Tam, C. M.(2006). Particle swarm optimization for resource-constrained project scheduling, *International Journal of Project Management*, 24, 83–92.