

A new approach for solving resource constrained project scheduling problems using differential evolution algorithm

Arian Eshraghi*

Sharif University of Technology, Tehran, Iran

CHRONICLE

Article history:

Received July 22 2015

Received in Revised Format

September 17 2015

Accepted November 5 2015

Available online

November 8 2015

Keywords:

Resource constrained

Metaheuristic algorithms

Project scheduling

Differential evolution

Investment

ABSTRACT

One subcategory of project scheduling is the resource constrained project scheduling problem (RCPS). The present study proposes a differential evolution algorithm for solving the RCPS making a small change in the method to comply with the model. The RCPS is intended to program a group of activities of minimal duration while considering precedence and resource constraints. The present study introduces a differential evolution algorithm and local search was added to improve the performance of the algorithm. The problems were then solved to evaluate the performance of the algorithm and the results are compared with genetic algorithm. Computational results confirm that the differential evolution algorithm performs better than genetic algorithm.

© 2016 Growing Science Ltd. All rights reserved

1. Introduction

Scheduling of projects can be implemented for optimization planning, consensus planning, milestone schedules, and outage schedules (Klimek, 2011). A resource constrained project scheduling problem (RCPS) is a type of project scheduling that plays a prominent role in project management (Arjmand & Najafi, 2015). RCPS discusses scheduling plans and projects in light of resource constraints. It can be divided into different classifications, the most important of which divides it into single-mode and multi-mode categories. Other classifications for RCPS are non-regular objective functions, and stochastic RCPS, bin-packing-related RCPS, and multi-RCPS problems (Rahmani et al., 2015). Blazewicz (1983) proved that RCPS is NP-hard. Yang et al. (2001) studied the different types of RCPS (Rahmani et al., 2015). Demeulemeester (1995) presented another procedure for RACP and Rodrigues and Yamashita (2010) developed an algorithm for it (Biloliar, 2014). Mohring (1984) introduced the resource investment problem (RIP) in 1984 and considered it a “problem of scarce time” and proposed an exact solution. Biloliar (2014) presented a solution for RCPS based on graph algorithms. RIP has two important features. First, every activity has a fixed duration and, second, every project activity needs

* Corresponding author. Tel: +98 21 77869150
E-mail: arian2001_e@yahoo.com (A. Eshraghi)

a constant amount of resources throughout implementation. Demeulemeester and Herroelen (1995), Zimmermann (1998), and Engelhard and Drexl (2001) expanded the literature about RIP. Hsu and Kim (2005) presented a solution for RIP using MRCPSP and solved the problem using maturity constraints and resource applications. A heuristic procedure was proposed to solve RIP for the first time by Akpan (1997). RIP/max was presented by Nubel (1999), who proposed a depth-first branch and bound procedure as a solution.

Shahdrok and Kianfar (2007) proposed a new RIP in which tardiness is permitted with penalties and called it RIPT and solved it using a genetic algorithm. The RIP with discounted cash flow (RIPDCF) maximizes discounted cash flows for project payments as proposed by Najafi and Niaki (2007). Arjmand and Najafi (2015) solved a multi-mode bi-objective resource investment problem using two modified meta-heuristic algorithms they called NSGA-II and MOPSO. They compared the algorithms using the MADM approach called TOPSIS. Hartmann and Kolisch (1999, 2000, 2006) and Kolisch and Padman (2001) developed meta-heuristic methods to solve large-sized RCPSP. Near-optimal solutions were generated using this method in large-sized RCPSP (Rahmani et al., 2015). Koulinas et al. (2014) presented particle swarm optimization for solving RCPSP based on a hyper-heuristic algorithm. The bee algorithm was used to solve RCPSP on the large scale by Ziarati et al. (2011). Akbari et al. (2012) used the artificial bee colony and bee swarm optimization to solve RCPSP .

Jia and Seo (2013) applied the facility layout problem and integrated the permutation-based artificial bee colony algorithm to efficiently cope with RCPSP. Xiao et al. (2014) presented the activity-list based nested partitions. Merkle et al (2002) used the ant colony as an important meta-heuristic method for solving RCSPS. The differential search (DS) algorithm was proposed to solve RCSPS by Rahmania et al. (2015). Different types of meta-heuristics have been used to solve this type of RCPSP. Price and Storn (1995) developed differential evolution meta-heuristic algorithms. Differential evolution (DE) algorithms solve optimization problems using a population-based probabilistic search algorithm. The basis of the algorithm is the use of distance and direction information from the current population to carry out search operations (Amiri & Barbin, 2015). It appears that DE has the ability to solve complex RCPSP problems. The present study developed a DE for RIP with a single mode resource constrained project. Other types of RCPSP problems can be solved by making slight changes in the DE algorithm.

The problem is defined in Section 2. Section 3 presents the basic concepts of the DE algorithm. Section 4 describes testing of the performance of the DE algorithm after adjustment of the parameters. Section 5 presents the conclusions.

2. Problem formulation

Problem can be defined in the single-mode form as follows:

A project with n activities is assumed. The node network has $n + 2$ nodes with no loops. The first and final nodes are dummies; consequently their duration and resource requirements equal zero. Constant C_d , is defined as the cost of unit of time for the project. The duration of activity i is denoted as D_i . The predecessor list of activity j is denoted as P_{rj} . There are $k = \{1, \dots, \rho\}$ renewable resource types. Factor m_{jk} determines the resource requirements of activity j with respect to resource type k . M_k denotes the total resource availability for resource k during project scheduling and C_k is the resource cost of for each unit of available capacity. The aim of the problem is to schedule each activity and find its start time and determine the amount necessary of resource of M_k to minimize the total cost of the project for all resources and the tardiness penalty. Assume that $S = \{S_0, S_1, \dots, S_{n+1}\}$ is a feasible schedule where S_i is the start time of activity i and x_{it} is defined as:

$$x_{it} = \begin{cases} 1 & \text{if activity } i \text{ is started at time } i. \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

The mathematical model is:

$$\min \left\{ \sum_{k=1}^{\rho} C_k M_k + C_d \times S_{n+1} \right\} \quad (2)$$

subject to

$$\sum_{t=0}^T t \times x_{it} \geq D_j + \sum_{t=0}^T t \times x_{jt}, \quad j \in p_i, i = 1, \dots, n+1, \quad (3)$$

$$\sum_{i=1}^n \sum_{u=t-D_i+1}^t m_{ik} \times x_{iu} \leq M_k, \quad t = 0, \dots, T, \quad k \in K = \{1, \dots, \rho\} \quad (4)$$

$$\sum_{t=0}^T x_{it} = 1, \quad i = 1, \dots, n+1, \quad (5)$$

$$x_{01} = 1, \quad (6)$$

$$x_{it} \in \{0, 1\}, i = 0, \dots, n+1, t = 0, \dots, T, \quad (7)$$

$$M_k \geq 0, \quad k \in K = \{1, \dots, \rho\} \quad (8)$$

In this model, Objective (2) is to minimize the makespan and total cost of the resource capacities. Eqs. (3-8) represent the constraints of the problem. Eq. (3) represents the precedence constraints, i.e., the start time of j is always greater than or equal to the finish time of predecessor activity i , which belongs to predecessor set \mathcal{P}_j of j . Constraint (4) states that the usage of each resource within each period should not exceed the total capacity of the resource. Eq. (5) assumes that each activity is assigned exactly one start time. Constraints (7) and (8) states that the decision variables are binary and positive integer variables, respectively.

3. Main section

The DE algorithm is meta-heuristic (Storn & Pierce, 1995) and finds the optimal solutions using an efficient search process in the direction of optimistic variables; it can change wrong directions into correct directions. This feature means that DE has speed (Amiri & Barbin, 2015). There are four main types of DE: generation of initial population, mutation, crossover, and selection.

The opposition-based differential evolution (ODE) was proposed by Rahnamayan and Wang (2008) to produce better results from DE on the large scale. Wang et al. (2012) used an orthogonal genetic algorithm (Leung, 2001) and proposed DE through orthogonal crossover. Deng et al. (2015) combined ODE and the orthogonal genetic algorithm to develop DE and proposed a novel local search operation for large-scale optimization problems. The DE standard framework is shown in Fig. 1 (Wang et al., 2012).

Variables $x_{i,G}$ and $a_{i,G}$ are the target resultant and its mutant resultant, respectively; $b_{i,G}$ is the vector that performs the crossover and inherits properties from $x_{i,G}$. For selection and replacement, $b_{i,G}$ is chosen if it has a better objective function than $x_{i,G}$. It is evident that this method cannot be used in RCSPS problems; thus, some of concepts, such as mutation, were changed to achieve the ability to solve the problems.

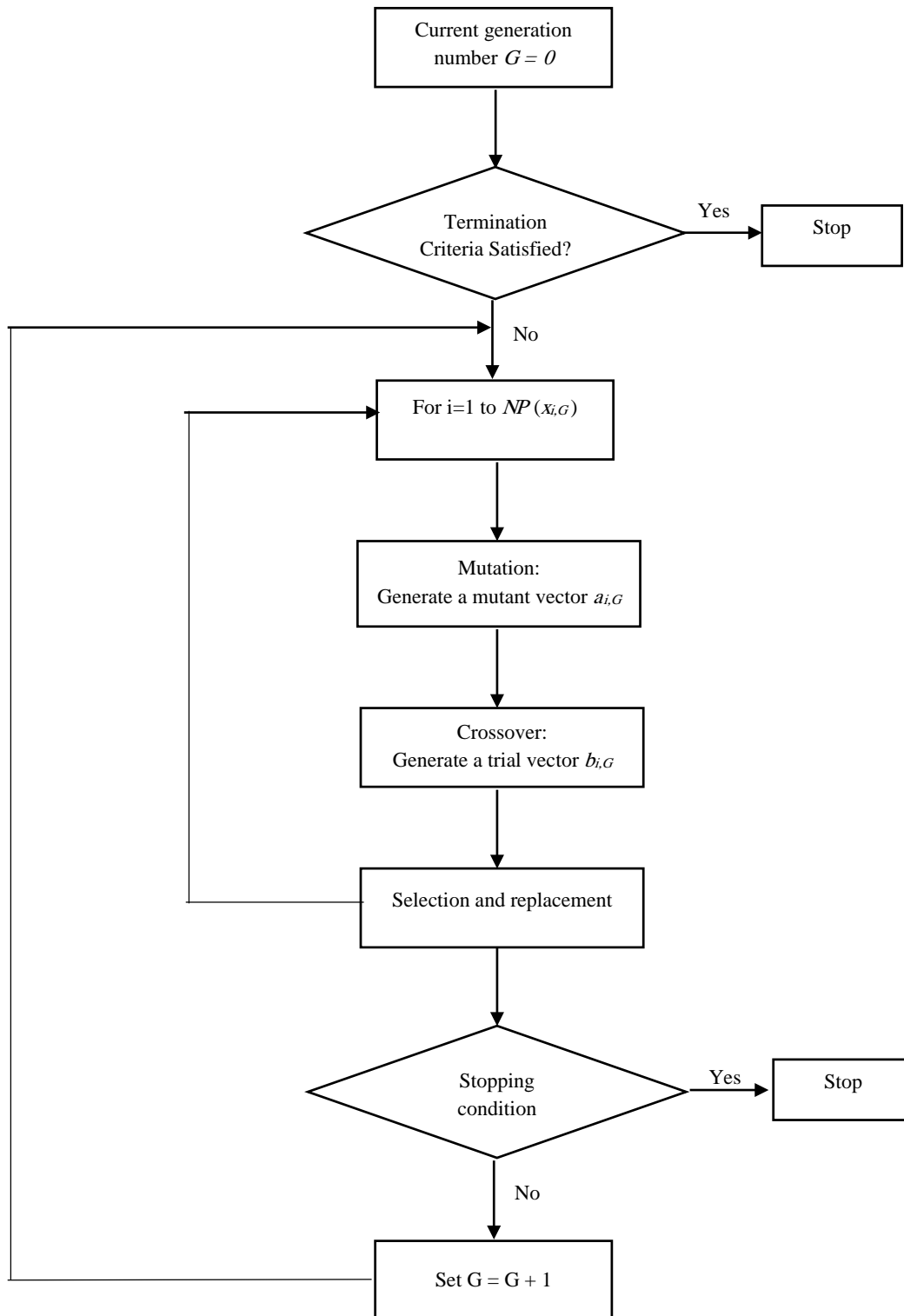


Fig. 1. DE standard framework

3.1 Chromosome representation and objective function

The chromosomes designed in the first step were those used by Shahdrok and Kianfer (2007). The design of the first part of the chromosomes was based on the activity sequence proposed by Hartmann (2000). For MRCPSPP problems, the chromosomes should consist of three parts (Arjmand & Najafi, 2015). A list of available resource capacities has been proposed for second part of the present approach for SMRCPSPP as:

$$I = ((i_o^I, \dots, i_{n+1}^I), (M_1^I, \dots, M_\rho^I)). \tag{9}$$

3.2. Chromosome generation

The chromosome activity list can be produced by permutation of n activities; however, it is possible that the precedence list constraint makes the activity sequence infeasible. Different methods have been proposed to remove infeasible solutions (Arjmand & Najafi, 2015). The repair approach implemented by Shahdrok and Kianfer (2007) changes infeasible solutions to feasible solutions and was applied here. To generate the second part, resource capacities M_k^I , ($k = 1, \dots, \rho$) were created for each chromosome I as:

$$M_k^I = M_k^{\min} + rand(0,1) \times (M_k^{\max} - M_k^{\min}). \tag{10}$$

The capacity list in the second part should be a feasible number; thus, M_k^I should be a number that lies between the defined lower and upper bounds. The lower and upper bounds are described as:

$$\overline{M}_k = \sum_{i=1}^n m_{ik} \tag{11}$$

$$\underline{M}_k = Max \{ \sum_{i=1}^n (m_{ik} \times D_i) / T, Max_{i=1, \dots, n} \{ m_{ik} \} \} \tag{12}$$

The repair operator was used to change infeasible solutions for the capacity list. For this operator, if the j^{th} value of M_j^I of the chromosomes is outside search region $[\underline{M}_k, \overline{M}_k]$, then M_j^I is updated as:

$$M_j^I = \begin{cases} \min\{\overline{M}_k, 2\underline{M}_k - M_j^I\} & \text{if } M_j^I \leq \underline{M}_k \\ \min\{\underline{M}_k, 2\overline{M}_k - M_j^I\} & \text{if } \underline{M}_k \leq M_j^I \end{cases} \tag{11}$$

In each generation, \overline{M}_k and \underline{M}_k should be updated so that they equal the maximum values of the parameters in the current population.

3.3 Mutation

The lineament feature of DE is its mutation operator. An important difference between DE and other meta-heuristic algorithms is the precedence of mutation over crossover. For chromosome $I^p = ((i_o^p, \dots, i_{n+1}^p), (M_1^p, \dots, M_\rho^p))$, apply the DE mutation by selecting three random chromosomes from population, I_a , I_b and I_c . For the activity sequence, create I_{nb} as equal to I_b , but change the places that are same as I_c to empty. This produces $I_{nb} = (i_o^{nb}, \dots, i_{n+1}^{nb}) \exists j, j \in \{0, \dots, n+1\}$ in which i_j^{nb} is empty.

To fill the empty places of I_{nb} , i_j^{nb} , insert $i_j^{nb} = i_l^a$, where l is the lowest index such that $i_l^a \notin \{i_0^{nb}, \dots, i_{n+1}^{nb}\}$. For example, if $n = 6$, then $(i_o^I, \dots, i_{n+1}^I) = (0, 1, 2, 3, 4, 5, 6, 7)$. Three random chromosomes are then chosen from the population, such as $(I_a = (i_o^I, \dots, i_{n+1}^I) = (0, 1, 3, 2, 5, 4, 6, 7))$, $I_b = (j_o^I, \dots, j_{n+1}^I) = (0, 2, 3, 1, 4, 5, 6, 7)$, and $I_c = (i_o^I, \dots, i_{n+1}^I) = (0, 1, 6, 3, 4, 5, 2, 7)$ to produce $I_{nb} = (i_o^{nb}, \dots, i_{n+1}^{nb}) = (0, 2, 3, 1, \square, \square, 6, 7)$. The new child is $I^{newchild} = (i_o^C, \dots, i_{n+1}^C) = (0, 2, 3, 1, 5, 4, 6, 7)$. Because it is possible for the result of the mutation to be infeasible, the repair approach is used to change an infeasible mutation to a feasible one.

Next, mutation operators were used for the second part of the list. Let $M_K = (M_1^p, \dots, M_\rho^p)$ be the capacity list of the parent. Use the standard form of the mutation operators of DE as proposed by Storn and Pierce (1995). Two commonly-used mutation operators are (Wang et al., 2012):

- **DE/rand/1 mutation:** For each resource i , a new solution is generated as:

$$M_{k,i}^{newchil} = M_{k,i}^a + F \cdot (M_{k,i}^b - M_{k,i}^c) \quad (12)$$

where random indices a, b and $c \in \{1, \dots, NP\}$ are integer numbers and F is a real and constant factor which determines the consolidation of different variations of $(M_{k,i}^b - M_{k,i}^c)$ (Biju et al., 2015). Higher diversity is created in the generated population for larger values of F and lower values for F result in faster convergence (Rahnamayan & Wang, 2008).

- **DE/rand/2 mutation:** A new chromosome is generated from the mutation by randomly selecting four chromosomes (I_c, I_b, I_d, I_e) from the population. For each resource i , a new solution is generated as:

$$M_{k,i}^{newchil} = M_{k,i}^a + F \cdot (M_{k,i}^b - M_{k,i}^c) + F \cdot (M_{k,i}^d - M_{k,i}^e), \quad (13)$$

where random indices a, b, c, d and $e \in \{1, \dots, n\}$ are integers that are mutually exclusive. F is a real and constant factor which determines the consolidation of different variations of $(M_{k,i}^b - M_{k,i}^c)$ and $(M_{k,i}^d - M_{k,i}^e)$. It is possible for the result of a mutation of the capacity to be infeasible; thus, the repair operator of the capacity list is used to change an infeasible solution to a feasible solution.

3.4 Crossover

DE employs the crossover operation to generate new solutions to increase diversity in the population. This operator is the same that for a crossover in a GA (Amiri & Barbin, 2015). Two crossover operators are applied in the DE method.

- **Binomial crossover:** Chromosome $I^{CH} = ((M_o^{CH}, \dots, M_{n+1}^{CH}), (M_1^{CH}, \dots, M_\rho^{CH}))$ is generated as:

$$M_i^{CH} = \begin{cases} M_i^{newchil} & \text{if } rand_i(0,1) \leq CR \text{ or } i = i_{rand} \\ M_i^p & \text{otherwise} \end{cases} \quad (14)$$

where index i_{rand} is a random number in $[1, n]$. CR is the crossover constant $\in (0,1]$ and is the user-defined crossover control parameter. Because i_{rand} is used, R_i^{CH} is always different from M_i^p (Wang et al., 2012).

- **Exponential crossover:** Chromosome $I^{CH} = ((i_o^{CH}, \dots, i_{n+1}^{CH}), (M_1^{CH}, \dots, M_\rho^{CH}))$ is generated as:

$$i_i^{CH} = \begin{cases} i_j^{newchil} & \text{for } j = \langle z \rangle_n, \langle z+1 \rangle_n, \dots, \langle z+Z-1 \rangle_n \\ i_j^p & \text{otherwise} \end{cases} \quad (15)$$

where $i = 1, 2, \dots, n, j = 1, 2, \dots, n$ and $\langle \cdot \rangle_n$ define the modulo function for modulus n . The index z is a random number between 1 and n . Integer Z was chosen from $[1, n]$ and has a probability of $p_r(Z \geq u) = CR^{u-1}, u > 0$. Parameters z and Z are then regenerated for each chromosome I^{CH} and the exponential crossover can be used for the capacity list. The first part of equation is the same for the activity sequence. To choose i_j^{CH} from the second part, use i_j^p such that I^{CH} is $(i_o^{CH}, \dots, i_{j-1}^{CH}, \square, i_{j+1}^{CH}, \dots, i_{n+1}^{CH})$ and insert i_f^p , where f is the lowest index such that $i_f^p \notin \{i_0^{CH}, \dots, i_{n+1}^{CH}\}$.

It is better to use binomial crossover for the capacity list and exponential crossover for the schedule list. It is important after each crossover to apply the repair operator.

3.5 Selection

This operation decides whether I^{CH} should become part of the next generation ($G + 1$). For this purpose, it is compared with parent chromosome I^P using greedy selection. Selection is done as follows:

$$I_{i,G+1} = \begin{cases} I_{i,G}^{CH} & \text{if } f(I_{i,G}^{CH}) < f(I_{i,G}^P) \\ I_{i,G}^P & \text{otherwise.} \end{cases} \quad (16)$$

where $I_{i,G+1}$ is the selection chromosome for the next generation. These steps are repeated until a stopping criterion is met or the best solution is achieved. The best fitness values have, thus been obtained and the matrix of the best solution consisting of total overtime work, duration, and cost can be obtained (Biju et al., 2015).

3.6 Local search

The performance of DE depends on the dimensionality of the optimization problems. As the search space increases dimensionality, the cases increase the complexity of the problem exponentially. This increase in complexity can reduce performance; however, the performance of DE will increase quickly for problems with a large number of variables. Most problems have a large number of variables; thus, support of scalability is a valuable characteristic for any optimization method. A local search is proposed to address this issue. A number of studies have examined local search in DE. The opposition-based search proposed by Rahnamayan and Wang (2008) and orthogonal crossover proposed by Wang et al. (2012) were used here. Deng et al. (2015) combined these two operators and proposed a novel local search operation for large scale optimization problems.

3.6.1 Opposition-based DE

This operator was proposed by Rahnamayan and Wang (2008), who proposed quasi-opposition-based learning (ODE) and proved that isotropy speed and solution correctness of DE can be improved. A quasi-opposite point is more likely to be closer to the solution of the optimization problem than the opposite one (Deng et al., 2015). Opposition-based DE was carried out only for the capacity list. This operator works based on jumping rate J_r . ODE forces the process to jump to a new chromosome candidate that may be better than the current one (Rahnamayan and Wang, 2008). This operation is applied after generating a new population by selection, crossover, and mutation. After calculating the opposite population (N_p : number of population), the best chromosomes are selected from the union of the current population and the opposite population. The opposite point of (M_1^I, \dots, M_ρ^I) for $I^P = ((i_o^p, \dots, i_{n+1}^p), (M_1^p, \dots, M_\rho^p))$ is defined by its components as:

$$OM_k^p = MinM_k^p + MaxM_k^p - M_k^p, \quad (17)$$

where $k \in K = \{1, \dots, \rho\}$, $MaxR_k^p$ is the maximum number of resources k in N_p , and $MinM_k^p$ is the minimum number of resources k in N_p . In the first generation, $MaxM_k^p = \overline{M}_k$ and $MinM_k^p = \underline{M}_k$. Comprehensive experiments by Rahnamayan and Wang (2008) revealed that J_r should be a number in $[0, 0.4]$. It is important after each local search repair to apply the operator.

3.6.2 Orthogonal crossover

Zhang and Leung (1999) proposed orthogonal crossover to improve the performance of a GA. They proposed a quantization technique called QOX which uses the orthogonal crossover operators in the DE algorithm (Leung and Wang, 20001). Hui et al. (2013) inserted opposition-based learning (OBL) into an

orthogonal crossover. This combination improved the ability of orthogonal crossover. An orthogonal array for K factors with Q levels and M combinations can be denoted as $(K), L_M(Q^K)$. For example, $L_9(3^4)$ is used in DENSL as follows (Deng et al, 2015):

$$L_9(3^4) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 \\ 1 & 3 & 3 & 3 \\ 2 & 1 & 2 & 3 \\ 2 & 2 & 3 & 1 \\ 2 & 3 & 1 & 2 \\ 3 & 1 & 3 & 2 \\ 3 & 2 & 1 & 3 \\ 3 & 3 & 2 & 1 \end{bmatrix} \quad (18)$$

Zhang and Leung (1999) believed each trial solution in a search algorithm could be regarded as an experiment. Another version of OX was introduced by Leung and Wang (2001) which proposed a quantization technique for OX that was called QOX by Wang et al. (2012). QOX was employed in the present algorithm. Orthogonal crossover was used only for the capacity list to improve the search ability of DE. Note that it is not logical to apply QOX to every pair of mutant and target chromosomes because it would consume too much computing time; thus, for every generation, QOX was performed for only one member of the population (Wang et al., 2012).

Let chromosome $I^P = ((i_o^p, \dots, i_{n+1}^p), (M_1^p, \dots, M_\rho^p))$ be a selection member for QOX. First, mutation and crossover operations are carried out as explained above. The capacity list is given two parent individuals $M_K^P = (M_1^P, \dots, M_\rho^P)$ and $M_K^{newchil} = (M_1^{newchil}, \dots, M_\rho^{newchil})$ where M_K^P is the capacity list from the parent and $M_K^{newchil}$ is the capacity list from the mutation operator. QOX first quantizes the search range and defines Q levels $l_{i1}, l_{i2}, \dots, l_{iQ}$ based on $L_M(Q^K)$ as follows:

$$l_{i,j} = \min(M_i^P, M_i^{newchil}) + \frac{j-1}{Q-1} \bullet (\max(M_i^P, M_i^{newchil}) - \min(M_i^P, M_i^{newchil})), \quad j = 1, \dots, Q. \quad (19)$$

Eq. (18) and Eq. (19) can generate nine children given two parents. A general explanation of QOX has been provided by Wang et al. (2012) After generating nine children, the activity sequence achieved from crossover and mutation is combined with the nine children. The evaluation unfitness function is carried out for these nine children and the best solution is chosen (Wang et al., 2012).

3.6.3 Novel local search operation

Deng et al. (2015) combined the orthogonal crossover and opposition operator to propose a new local search. This local search can generate 18 experimental chromosomes to locate more space. Because of the large number of chromosomes, this operation cannot be used for each pair of mutation chromosomes; thus, one individual is chosen from every population for a local search. This method is similar to orthogonal crossover except that, that after producing nine children using orthogonal crossover, it generates the opposite of each set of nine children. Next, the unfitness function is carried out on the 18 children and the best solution is chosen from among them.

4. Testing and Comparison of DE Algorithm

A set of solved problems is necessary for analysis of the computational performance of the DE algorithm. Shadrokh and Kianfar (2007) used the branch-and-bound procedure proposed by Demeulemeester and

Herroelen (1992) to solve 690 problems. The proposed DE method was used on the single mode test cases in the PSPLIB library. Evaluation of the results requires determination of the initial parameters in meta-heuristic algorithms. The DE algorithm is susceptible to its input data, which has a major effect on DE performance. A flexible and efficient solution can be achieved using the appropriate parameters. One important parameter is the number of populations in the meta-heuristic algorithms. This problem suffers from immature convergence with a low population size and will not reach an eligible response that is approaches the global optimum. With a large population size, more time for computation can be expected. This means that determining the appropriate population size to achieve the optimal solution is important. Table 1 shows the parameters that most affect the performance of the proposed model (Amiri and Barbin, 2015).

Table 1

Performance parameters

Parameters	Values
No.Population	100
CR	0.9
F	0.9
Selection	Randomize
Jumping rate	0.3
Generation	1000, 5000, 50000

The j30 and j60 sets of problems were selected from the PSPLIB library. The algorithms were run through 10 iterations. All the results were calculated by using Matlab 2014. To analyze the effect of a scheduling scheme, the SSS and PSS scheduling schemes were considered. The DE/rand/1 and DE/rand/2 and their crossover, binomial crossover, and exponential crossover mutations were used. The opposition-based DE, orthogonal crossover, and novel local search operations were the local searches used.

Table 2

Effect of schedule generation scheme, mutation operators, and selection method on average deviation

Scheduling scheme	Selection	Mutation	Crossover	Local search	1000	5000	50000
SSS	Random	DE/rand/1	CB/CE	-	0.54	0.23	0.07
SSS	Random	DE/rand/1	CB/CE	ODE	0.42	0.11	0.02
SSS	Random	DE/rand/1	CB/CE	NLSO	0.39	0.12	0.04
SSS	Random	DE/rand/1	CB/CE	QOX	0.30	0.12	0.02
SSS	Random	DE/rand/2	CB/CE	-	0.57	0.28	0.09
SSS	Random	DE/rand/2	CB/CE	ODE	0.44	0.17	0.05
SSS	Random	DE/rand/2	CB/CE	NLSO	0.34	0.13	0.06
SSS	Random	DE/rand/2	CB/CE	QOX	0.30	0.11	0.04

We tried to change the existing unfitness chromosomes to those with less unfitness by applying local searches. The effect of DE in a different mode is proposed in Table 2 on the average deviation for j30. We tested both SSS and PSS. Also results achieved were the same as Shadrokh et al. (1996) confirmed that SSS was used for each composition of operations and produced better results than all compositions where PSS was used. For this reason, only the SSS compositions are shown. Table 2 shows that the algorithms that used DE/rand/1 performed better than DE/rand/2. Table 2 shows that schemes with local search performed better. Of the local search methods, QOX performed the best. Table 3 shows the success rate of problems solved successfully using the DE algorithm. As seen, methods with local search performed better than the others. It is important to note that the DE without local search in problems with less activity (10 and 20) performed quickly during computation of the average percentage of deviation from the optimal solution. It appears that extra solutions were calculated in the local search.

Table 3

Effect of schedule generation scheme, mutation operators, and selection method on average deviation

Scheduling scheme	Selection	Mutation	Crossover	Local search	1000	5000	50000
SSS	Random	DE/rand/1	CB/CE	-	86.3%	91.22%	97.12%
SSS	Random	DE/rand/1	CB/CE	ODE	88.43%	92.32%	96.2%
SSS	Random	DE/rand/1	CB/CE	NLSO	84.32%	92.43%	98.31%
SSS	Random	DE/rand/1	CB/CE	QOX	88.86%	93.44%	98.43%
SSS	Random	DE/rand/2	CB/CE	-	86.1%	90.92%	95.42%
SSS	Random	DE/rand/2	CB/CE	ODE	86.8%	90.22%	94.19%
SSS	Random	DE/rand/2	CB/CE	NLSO	85.33%	91.62%	97.12%
SSS	Random	DE/rand/2	CB/CE	QOX	87.39%	92.02%	98.02%

Over time, the algorithms performing the local search showed the best average percentage of deviation from the optimal solution rather. The results obtained using DE were compared with the results of a GA (Table 4). For each method, the minimum, maximum, and average results were obtained in 10 iterations. The standard deviation was also calculated.

Table 4

Comparison of performances of GA and DE

Name	GA				DE			
	Min	Avg.	Max	SDev	Min	Avg.	Max	SDev
j301_1	512	512	512	0	512	512	512	0
j305_2	1214	1219.5	1246	11.2	1214	1214	1214	0
j3010_1	698	698	698	0	698	698	698	0
j3015_1	1054	1054	1054	0	1054	1054	1054	0
Average	869.5	870.8	877.5	2.8	869.5	869.5	869.5	0
j601_1	409	415	438	12.33	409	409	409	0
j605_1	1018	1096	1167	19.19	918	926.33	984.64	21.59
j6010_1	1281	1284	1291	2.43	1264	1266	1268	2.12
j6015_1	1480	1480	1480	0	1480	1480	1480	0
Average	1047	1068.75	1094	8.48	1017	1020.25	1035.41	5.92

Table 4 shows that the minimum, maximum, and average results achieved from DE are better than those obtained results from the GA. The average standard deviation for the j30 problems for the DE was 0, which is better than that for the GA (2.8). Table 4 also lists the results for the j60 problems. The increase in the size of the problems increased the standard deviation, but DE continued to obtain better results than GA. Table 5 shows the success rates for the j30, j60, and j120 problems for 1000 and 5000 schedule generations. Tables 4 and 5 demonstrate that DE performed better and produced more appropriate results than did GA. All answers prove that the DE algorithm performed well when solving RCSPS.

Table 5

Success rates for j30, j60, and j120

Generations	GA		DE	
	1000	5000	1000	5000
j30	81.50%	91.30%	89.50%	94.67%
j60	74.23%	73.33%	79.32%	84.32%
j120	29.84%	34.45%	45.53%	52.67%

5. Conclusion

The present study has examined the resource investment problem using a DE algorithm. This algorithm has many parameters. To test the proposed DE algorithm, the j30 and j60 problems from the PSPLIB library were solved and the results have been compared. It was demonstrated that DE performed

differently for different problems. Local searches were proposed to avoid being trapped in the local optima for large-scale optimization problems and to increase the ability of the algorithm to solve RCPS problems. The effectiveness and speed of the proposed model was demonstrated by comparison with a sample project that was previously solved using GA. The proposed model solved the RCSPS with accurate convergence to optimal solutions, which was the main objective of this work. The results of DE have revealed that the proposed algorithms performed better than the genetic algorithms with less error. The DE algorithm will be proposed for different types of scheduling problems in the future.

Acknowledgements

This research was done by help of Amirhoseyn Khalili and Amin Sameti.

References

- Akbari, R., Zeighami, V., & Ziarati, K. (2011). Artificial bee colony for resource constrained project scheduling problem. *International Journal of Industrial Engineering Computations*, 2(1), 45-60.
- Akpan, E. O. (1997). Optimum resource determination for project scheduling. *Production Planning & Control*, 8(5), 462-468.
- Amiri, M., & Barbin, J. P. (2015). New approach for solving software project scheduling problem using differential evolution algorithm.
- Arjmand, M., & Najafi, A. A. (2015). Solving a multi-mode bi-objective resource investment problem using meta-heuristic algorithms. *I*, 41-58.
- Blazewicz, J., Lenstra, J. K., & Kan, A. R. (1983). Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1), 11-24.
- Bilollikar, V. S., Fr, C. R., & Jain, M. K. (2014). An adaptive crossover genetic algorithm for multi-mode resource constrained project scheduling with discounted cash flows. 1-9. Proceeding.
- Biju, A. C., Victoire, T., & Mohanasundaram, K. (2015). An Improved Differential Evolution Solution for Software Project Scheduling Problem. *The Scientific World Journal*, Article ID 232193.
- Hsu, C. C., & Kim, D. S. (2005). A new heuristic for the multi-mode resource investment problem. *Journal of the Operational Research Society*, 56(4), 406-413.
- Demeulemeester, E., & Herroelen, W. (1992). A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management science*, 38(12), 1803-1818.
- Deng, C., Dong, X., Yang, Y., Tan, Y., & Tan, X. (2015). Differential Evolution with Novel Local Search Operation for Large Scale Optimization Problems. In *Advances in Swarm and Computational Intelligence* (pp. 317-325). Springer International Publishing.
- Demeulemeester, E. (1995). Minimizing resource availability costs in time-limited project networks. *Management Science*, 41(10), 1590-1598.
- Drexl, A., & Kimms, A. (2001). Optimization guided lower and upper bounds for the resource investment problem. *Journal of the Operational Research Society*, 340-351.
- Hartmann, S., & Kolisch, R. (2000). Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 127(2), 394-407.
- Jia, Q., & Seo, Y. (2013). Solving resource-constrained project scheduling problems: conceptual validation of FLP formulation and efficient permutation-based ABC computation. *Computers & Operations Research*, 40(8), 2037-2050.
- Klimek, M. (2011, January). A genetic algorithm for the project scheduling with the resource constraints. In *Annales UMCS, Informatica* (Vol. 10, No. 1, pp. 117-130).
- Kolisch, R. (1996). Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90(2), 320-333.
- Kolisch, R., & Hartmann, S. (1999). *Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis* (pp. 147-178). Springer US.

- Kolisch, R., & Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European journal of operational research*, 174(1), 23-37.
- Kolisch, R., & Padman, R. (2001). An integrated survey of deterministic project scheduling. *Omega*, 29(3), 249-272.
- Koulinas, G., Kotsikas, L., & Anagnostopoulos, K. (2014). A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem. *Information Sciences*, 277, 680-693.
- Leung, Y. W., & Wang, Y. (2001). An orthogonal genetic algorithm with quantization for global numerical optimization. *Evolutionary Computation, IEEE Transactions on*, 5(1), 41-53.
- Merkle, D., Middendorf, M., & Schmeck, H. (2002). Ant colony optimization for resource-constrained project scheduling. *Evolutionary Computation, IEEE Transactions on*, 6(4), 333-346.
- Möhring, R. H. (1984). Minimizing costs of resource requirements in project networks subject to a fixed completion time. *Operations Research*, 32(1), 89-120.
- Najafi, A. A., & Niaki, S. T. A. (2006). A genetic algorithm for resource investment problem with discounted cash flows. *Applied Mathematics and Computation*, 183(2), 1057-1070.
- Nübel, H. (1999). *A branch and bound procedure for the resource investment problem subject to temporal constraints*. Inst. für Wirtschaftstheorie und Operations-Research.
- Nübel, H. (2001). The resource renting problem subject to temporal constraints. *OR-Spektrum*, 23(3), 359-381.
- Rahmani, N., Zeighami, V., & Akbari, R. (2015). A study on the performance of differential search algorithm for single mode resource constrained project scheduling problem. *Decision Science Letters*, 4(4), 537-550.
- Rahnamayan, S., Tizhoosh, H. R., & Salama, M. M. (2008). Opposition versus randomness in soft computing techniques. *Applied Soft Computing*, 8(2), 906-918.
- Storn, R., & Price, K. V. (1996, May). Minimizing the Real Functions of the ICEC'96 Contest by Differential Evolution. In *International Conference on Evolutionary Computation* (pp. 842-844).
- Shadrokh, S., & Kianfar, F. (2007). A genetic algorithm for resource investment project scheduling problem, tardiness permitted with penalty. *European Journal of Operational Research*, 181(1), 86-101.
- Wang, H., Rahnamayan, S., & Wu, Z. (2013). Parallel differential evolution with self-adapting control parameters and generalized opposition-based learning for solving high-dimensional optimization problems. *Journal of Parallel and Distributed Computing*, 73(1), 62-73.
- Wang, Y., Cai, Z., & Zhang, Q. (2012). Enhancing the search ability of differential evolution through orthogonal crossover. *Information Sciences*, 185(1), 153-177.
- Xiao, L., Tian, J., & Liu, Z. (2014, June). An Activity-List based Nested Partitions algorithm for Resource-Constrained Project Scheduling. In *Intelligent Control and Automation (WCICA), 2014 11th World Congress on* (pp. 3450-3454). IEEE.
- Yang, B., Geunes, J., & O'brien, W. J. (2001). Resource-constrained project scheduling: Past work and new directions. *Department of Industrial and Systems Engineering, University of Florida, Tech. Rep.*
- Zhang, Q., & Leung, Y. W. (1999). An orthogonal genetic algorithm for multimedia multicast routing. *Evolutionary Computation, IEEE Transactions on*, 3(1), 53-62.
- Ziarati, K., Akbari, R., & Zeighami, V. (2011). On the performance of bee algorithms for resource-constrained project scheduling problem. *Applied Soft Computing*, 11(4), 3720-3733.
- Zimmermann, J., & Engelhardt, H. (1998). Lower bounds and exact algorithms for resource leveling problems. *Report WIOR-517, University Karlsruhe*.