The Project Portfolio Selection and Scheduling Problem: Mathematical Model and Algorithms

Bahman Naderi*

Assistant Professor, Young Researchers and Elite Club, Qazvin Branch, Islamic Azad University, Qazvin, Iran

Received 19 May , 2012; Revised 26 July, 2012; Accepted 15 October, 2012

Abstract

This paper investigates the problem of selecting and scheduling a set of projects among available projects. Each project consists of several tasks and to perform each one some resource is required. The objective is to maximize total benefit. The paper constructs a mathematical formulation in form of mixed integer linear programming model. Three effective metaheuristics in form of the imperialist competitive algorithm, simulated annealing and genetic algorithm are developed to solve such a hard problem. The proposed algorithms employ advanced operators. The performance of the proposed algorithms is numerically evaluated. The results show the high performance of the imperialist competitive algorithm outperforms the other algorithms.

Keywords: Project portfolio selection and scheduling, imperialist competitive algorithm, simulated annealing, genetic algorithm, mixed integer programming.

1. Introduction

A classical project scheduling problem deals with planning a set of tasks that need to be executed in order to complete the project. It is commonly assumed that these tasks have fixed duration and need to meet some certain precedence relations. The objective is to specify a set of feasible schedules (starting or finishing times) for the tasks that minimizes the project completion time.

There is also a resource feature to the classical project scheduling problem. That is, the jobs commonly consume resources while being executed. The simple project scheduling problem, which considers unlimited available resources, can be easily solved to optimality by the critical path method (CPM) or the program evaluation and review technique (PERT). CPM is a deterministic model while PERT accepts the fact that activity durations are random. The simple project scheduling problem can be extended to the resource-constrained project scheduling problem (RCPSP) when imposed by limited renewable resources. That is, limited resources are available during every period in which the project needs to be executed. It is known that the RCPSP is NP-hard in the strong sense (Hartmann and Briskorn, 2010).

Both exact and heuristic algorithms are extensively studied (Brucker et al., 1998; Brucker et al., 1999; Kolisch and Hartmann, 2006). Among exact methods, the depth-first search implicit enumeration algorithm by Demeulemeester and Herroelen (1992, 1997) provides a current standard for solving the classical RCPSP problem.

Heuristic algorithms have also been studied by researchers. The pioneer study by Davis and Patterson (1975) proposes the minimum job slack (MINSLK) and resource scheduling method (RSM). Metaheuristics found to perform best are the methods of Montoya-Torres et al. (2010) and Zhang et al. (2006).

Although project planning and control have attracted considerable attention from both researchers and practicing managers, most of this attention focuses on individual projects rather than coordinated decision across multiple projects. On the other hand, project portfolio selection or capital budgeting problem is an active research topic in the field of project management and engineering management (Aaker and Tyebjee, 1978; Peng et al., 2008; Talias, 2007; Stummer et al., 2009; Henriksen and Palocsay 2008).

Project portfolio selection becomes a harder problem if the project interactions are studied (Yu et al., 2010). Considering its practical importance, several algorithms have been proposed to solve this problem (Mavrotas et al. 2006; Peng et al. 2008; Stummer et al. 2009).

In accordance with the above analyses, it can be found that the existing papers rarely consider both project portfolio selection and resource constrained project scheduling simultaneously. This paper studies the problem where multiple projects are available. The decision maker aims at selecting a subset of these projects and scheduling their tasks subject to limited resources.

* Corresponding author Email: bahman.naderi@aut.ac.ir

The objective is to maximize total benefit. The problem is first mathematically formulated by a mixed integer linear programming model. To effectively solve large sized problems, the paper develops three metaheuristics based on imperialist competitive algorithm, genetic algorithm and simulated annealing.

The rest of the paper is organized as follows. Section 2 presents the mathematical model of the problem. Section 3 describes the imperialist competitive algorithm. Section 4 presents the experimental performance evaluation. Finally, the conclusions are presented in Section 5.

2. Problem Formulation

The problem of selecting and scheduling projects can be described as follows. There are a set of n projects where each project has its own benefit. To perform each project a set of m tasks has to be operated. To operate each task, some resource types are required. All available resource from one type is limited. There are some precedence relationships among the tasks of a project. The objective is to first select among the projects, a subset of projects with a maximum total benefit, and then schedule them so as to complete the projects in a given time horizon. The process should be done in such a way resource constraints are met.

This section formulates the problem under consideration by a mixed integer linear programming (MILP) model. The parameters and indices of the proposed model are as follows.

n	The number of projects
m	The number of resources
T	The planning time horizon.
i	Index for projects, $i, \in \{1, 2, \dots, n\}$
n_i	The number of tasks of project i
l	Index for tasks, $l \in \{1, 2, \dots, n_i\}$
k	Index for resources, $k \in \{1, 2, \dots, m\}$
t, j	Index for time, $t, j \in \{1, 2, \dots, T\}$
$d_{i,l}$	The duration of task l of project i.
$S_{i,l}$	The set of prerequisite tasks of task l of project i
$E_{i,l}$	The earliest possible starting of task l of project i $E_i = \max\limits_{(j,r)\epsilon S_{i,l}} \{E_{j,r} + d_{j,r}, 0\}$
$L_{i,l}$	The latest possible starting of task l of project i $L_{i,l} = \max\limits_{(j,r)\vert(i,l)\epsilon S_{j,r}} \{L_{j,r} - d_{i,l} + 1, T - d_{i,l} + 1\}$
$r_{k,t}$	The available resource k at time period t.
b_i	The benefit of project i.
$a_{i,l,k}$	The resource k necessary for task l of project i at each time period.
$U_{i,l}$	The set of all tasks must be performed after task l of project i.

Decision variables used in the model are

$$X_{i,l,t} = \begin{cases} 1, & \text{if task } l \text{ of project } i \text{ starts at time } t \\ 0, & \text{otherwise} \end{cases}$$ where $E_{i,l} \le t \le L_{i,l}$

$$Y_i = \begin{cases} 1, & \text{if project } i \text{ is selected} \\ 0, & \text{otherwise} \end{cases}$$

The MILP model is as follows.

$$Max\ Z = \sum_{i=1}^{n} b_i Y_i$$

Subject to:

$$\sum_{t=E_{i,1}}^{L_{i,1}} X_{i,1,t} = Y_i \qquad \forall_i \qquad (2)$$

$$\sum_{i=1}^{n}\sum_{l=1}^{n_i}\left(\sum_{j=\max\{E_{i,l},t-d_{i,l}+1\}}^{\min\{t,L_{i,l}\}} X_{i,l,j}\right)a_{i,l,k} \le r_{k,t} \qquad \forall_{k,t} \qquad (3)$$

$$X_{j,r,t} \le 1 - Y_i + \sum_{k=\max\{E_{i,l},t+d_{j,r}\}}^{L_{i,l}} X_{i,l,k} \qquad \forall_{i,l,(j,r)\epsilon S_{i,l},t} \qquad (4)$$

$$X_{i,l,t} \in \{0,1\} \qquad \forall_{i,l,E_{i,l}\le t\le L_{i,l}} \qquad (6)$$

$$Y_i \in \{0,1\} \qquad \forall_i \qquad (7)$$

Equation (1) is the objective function that maximizes the total benefit of selected projects. Constraint set (2) specifies which projects are selected. Constraint set (3) ensures that the limitations of resources are met. Constraint set (4) assures that precedence relations of tasks are satisfied if a task is for a selected project. Constraint sets (6) and (7) define the decision variables.

3. The Proposed Algorithms

This section develops three metaheuristics to solve the problem based on imperialist competitive algorithm, genetic algorithm and simulated annealing.

3.1. The imperialist competitive algorithm

The imperialist competitive algorithm (ICA) is a novel population based evolutionary algorithm to solve various optimization problems. Very recently, several papers report the great performance from ICA. Atashpaz and Lucas (2007) and (2008) use ICA to solve the continuous optimization problems. Bagher et al. (2011) uses ICA to solve assembly line balancing problems. Banisadr et al. (2012) employs ICA to solve single machine scheduling problems. Zhou et al. (2012) develop ICA to deal with assembly sequence planning problems.

This algorithm contains a population of agents, known as countries where they are classified as imperialists and colonies. A collection of one imperialist and several colonies is called an empire. The basis of ICA is to simulate three sociopolitical processes among the empires: imperialistic behavior, imperialistic competition and independence. The idea behind the imperialistic behavior is that the imperialist attempts to penetrate the colony by attracting the culture and the social structure of each colony toward itself. During the imperialist competition, weak empires collapse and powerful ones take possession of their colonies. There is always a probability for some colonies to jointly separate from their empires and constitute a new empire. The outline of the proposed ICA is shown in Figure 1.

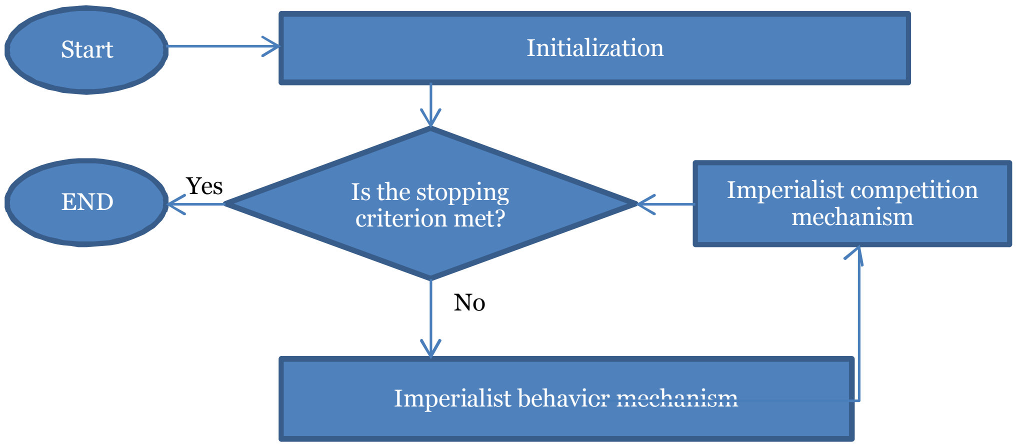

Start
Initialization
END
Yes
Is the stopping criterion met?
Imperialist competition mechanism
No
Imperialist behavior mechanism

Fig. 1. The outline of the proposed ICA

3.1.1. The Initialization Mechanism

ICA starts with a number of countries each of which represents a possible solution for the problem. It selects those with relative high fitness to be the imperialist, and the remaining becomes the colonies of these imperialists. The number of the colonies in each empire depends on the power of its imperialist. Hence, powerful imperialists have greater number of colonies while weaker ones have less.

To encode a solution and form initial countries, we use the permutation representation that determines the selected projects as well as their schedule. For example, consider a problem with 7 projects. One possible encoded solution is {4,7,1,3,2,6,5}. In this scheme, project 4 is scheduled according to a modified critical path method. In this rule, tasks of projects, one by one from left to right, are scheduled so as to start as early as possible while the precedence relations and resource constraints are met. Note that projects that cannot be completed in the given time horizon are not selected.

The number of countries is the population size indicated by *pop*. The initial countries are randomly generated from the feasible solutions. To define the initial imperialists, the first *I* best countries of the population are selected as the imperialist and the rest as colonies. Therefore, there are *I* empires. To rank the countries, we need to calculate the fitness. The fitness of a country (an encoded solution) is set to its objective function.

To assign colonies to imperialist, a stochastic procedure in which more chance is given to more powerful imperialists. To chance of empire k to hold each colony is as follows.

$$p_k = \frac{fit(k)}{\sum_{h=1}^{pop} fit(h)}$$

3.1.2. The Imperialist Behavior Mechanism

After forming initial empires, the imperialist behavior mechanism commences and the colonies of an empire move towards their imperialist. While a colony approaches its imperialist, it might become more powerful (better fitness) than its imperialist. In this case, the colony overcomes the imperialist and takes the control of the whole empire. In fact, the colony and the imperialist swap their positions. Then, the procedure continues by the new imperialist and colonies change their path and start moving toward this new imperialist. After the exchanging step, the total power of each empire is recalculated which depends on both the power of the imperialist and its colonies.

To take a colony towards its imperialist, we define a new country that inherits from both the colony and imperialist. In fact, we combine the colony and imperialist to form a new country. This is done through a cyclic operator with the following steps.

Step 1: Find the first project number in the imperialist that is not equal with the colony.

Step 2: Go to the same position in the colony and find its project number.

Step 3: Find the same project number in the imperialist, go to its position.
Step 4: If this project number is not reached before, go step 2; otherwise, go to step 5.
Step 5: The reached project numbers in the imperialist are copied into new country.
Step 6: The remaining project numbers are inserted into empty positions according to the colony.

Figure 2 shows the cyclic operator applied to a problem with 10 projects.

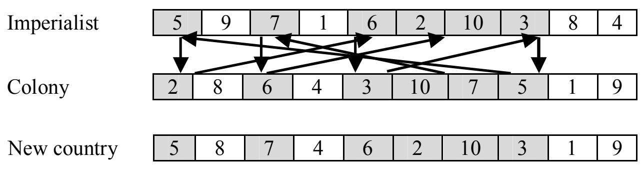


Fig. 2. The cyclic operator applied to a problem with 10 projects

After colonies are taken towards the current imperialist, the imperialist of the empire is updated. In other words, it is checked whether any of the new countries can beat the imperialist or not. If this is the case, the imperialist is replaced with that new country. We check if net improvement of this new country versus its imperialist is positive or not. If it is positive, we conclude that it is more powerful. The net improvement is calculated by the following formula.

$$\Delta = \frac{fit(i) - fit(c)}{fit(i)}$$

Where $fit(i)$ and $fit(c)$ are the makespan of imperialist and new country, respectively. If $\Delta > 0$, then the new country becomes the new imperialist. Then, the total power of empire is reevaluated. It is recommended to use both power of imperialist and colonies to calculate the total power. We use the following formula to obtain the total power (tp) of empire k.

$$tp_k = fit_{imperialist}\left(1 + \frac{\sum_{h=1}^{s_k} fit_h}{\sum_{h=1}^{pop} fit_h}\right)$$

where s_k is the number of countries in empire k.

3.1.3. The imperialist competition mechanism

In the imperialistic competition process, empires endeavor to conquer colonies of other empires and control them. When an imperialist broadens its empire by conquering more colonies, it becomes more enhanced. On the other hand, the imperialist losing its colonies becomes weaker. Once an empire loses all of its colonies, it is collapsed. After a while, all the empires, one by one with exception of the most powerful one, will vanish.

When all the colonies of the single remaining empire have the same position with their imperialist, consequently the same fitness, the algorithm converges to the best solution. To implement this concept, at each iteration, the weakest empire is selected and its weakest colony is given to the most powerful empire.

3.2. The simulated annealing

The simulated annealing (SA) is a local search based metaheuristic simulating the annealing process (Kirlpatrick et al., 1983; Kolon, 1999). SA includes a mechanism, called acceptance criterion, which enables it to partially avoid getting trapped in local optima. The acceptance criterion decides if the new generated solution is accepted or not. In this mechanism, even inferior solutions might be accepted.

3.2.1. The structure and acceptance criterion

Simulated annealing starts from an initial solution, and a series of moves are made until a stopping criterion is met. The basic idea of SAs is to generate a new permutation *s* by an operator from the neighborhood of the current permutation *x*. This new sequence is accepted or rejected by another random rule. A parameter *t*, called the temperature, controls the acceptance rule. The variation between objective values of two candidate solutions is computed $\Delta = fit(s) - fit(x)$. If $\Delta \leq 0$, permutation *s* is accepted. Otherwise, permutation *s* is accepted with probability equal to $exp(\Delta/t_i)$. The algorithm proceeds by trying a fixed number of neighborhood moves at each temperature t_i, while temperature is gradually decreased. We use exponential cooling schedule, $t_i = \alpha \cdot t_{i-1}$ (where $\alpha \in (0, 1)$ is temperature decrease rate). The initial temperature is set to be 50 and $\alpha = 0.97$.

3.2.2. The move operator

In this research, to generate new solution from the current solution an operator based on insertion neighborhood search is utilized. In this operator, one randomly selected project in the permutation is randomly relocated into a new position. Consider a problem with 10 projects. Figure 3 shows the numerical example. In this example. The randomly selected project is 4 and it is relocated into position 8.

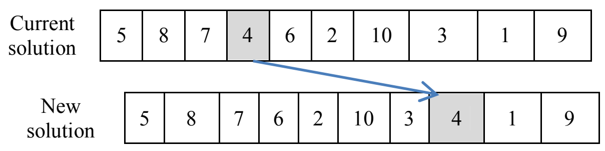


Fig. 3. The example of move operator

3.3. The genetic algorithm

Genetic algorithms (GA) arose towards 70s by the work of Holland (1975). They were intended to tackle some problems of industry which were difficult to solve with methods available that time. Nowadays, GA is considered as one of the typical metaheuristic approaches tackling both discrete and continuous optimization problems. The idea behind GA comes from Darwin's ''survival of the fittest'' concept, meaning that good parents produce better offsprings.

3.3.1. The general structure

GA searches a problem space with a population of *chromosomes* each of which represents an *encoded* solution. A *fitness value* is assigned to each chromosome according to its performance. The more desirable the chromosome is, the higher this value becomes. The population *evolves* by a set of operators until some *stopping criterion* is visited. A typical iteration of a GA, *generation*, proceeds as such: The best chromosomes of current population are directly copied to next generation (*reproduction*). A *selection mechanism* chooses chromosomes of the current population in such a way that a chromosome with the higher fitness value has more probability to be selected. The selected chromosomes mate and generate new offspring (*crossover*). After mating process, each offspring might mutate by another mechanism called *mutation*. Afterwards, the new population is evaluated again and the whole process is repeated.

3.3.2. The crossover and mutation

New solutions or offsprings are produced by crossing two other parents through an operator called *crossover*. The crossover operators must avoid generating infeasible solutions. The purpose is to generate "better" offsprings, i.e. to create better sequences after combining the parents. We use two-point crossover which can be described as follows.
Step 1: Select two cut points randomly.
Step 2: Copy directly the permutation before cut point 1 and after cut point 2 from parent 1 into offspring.
Step 3: Copy the remaining projects into offspring according parent 2

Figure 4 shows the two-point crossover applied to a problem with 10 projects.

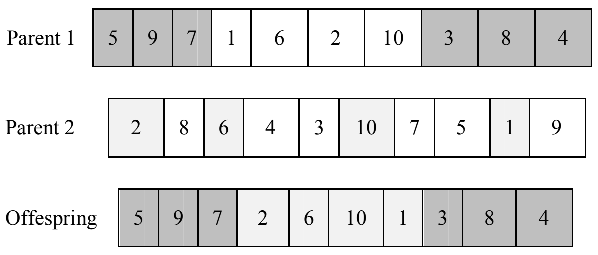


Fig. 4. The two-point crossover applied to a problem with 10 projects.

A mutation operator is utilized to slightly change the sequence, i.e. generating a new but similar sequence. The main purpose of applying mutation is to avoid convergence to a local optimum and diversify the population. Mutation operator can also be seen as a simple form of local search. In this research, we use the following mutation operator. The positions of two randomly selected projects are swapped. Figure 5 shows the procedure of the mutation applied to a problem with 10 projects.

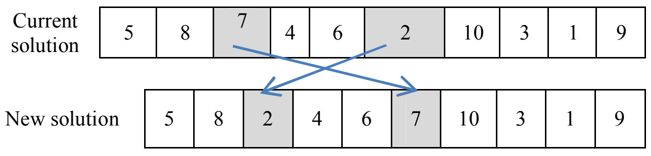


Fig. 5. The example of move operator

4. Numerical Experiment

This section evaluates the performance of the proposed algorithms. We first tune the parameters of the tested algorithms, then, a set of instances are generated and performance of the algorithms are compared. The algorithms are implemented in C++ and ran on a PC with 2.0 GHz Intel Dual Core CPU and 2 GB of RAM memory. We set the stopping criterion used when testing the algorithms is set to a fixed time limit of *n/2* seconds. To generate a set of instances we consider n = {10, 30, 50, 100} and m={2,4}. The duration of tasks is generated from a uniform distribution between [5 30]. The benefit of each project is randomly generated from a uniform distribution between [10 30]. For each combination of n and m, we generate 10 different instances. It sums up to 80 instances.

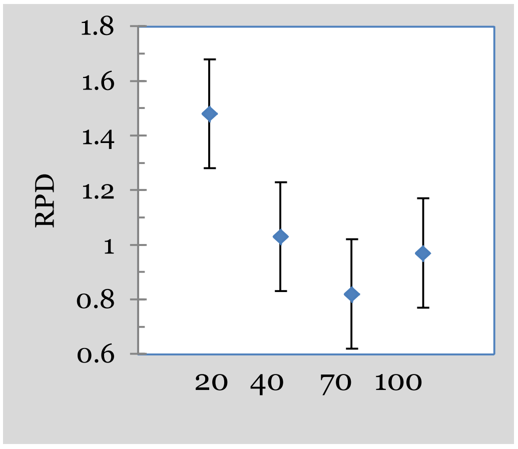
To compare the methods, we use relative percentage deviation (RPD). This is a common performance measure which is calculated as follows:

$$RPD = \frac{Max_{sol} - Alg_{sol}}{Max_{sol}} \times 100 \qquad (8)$$

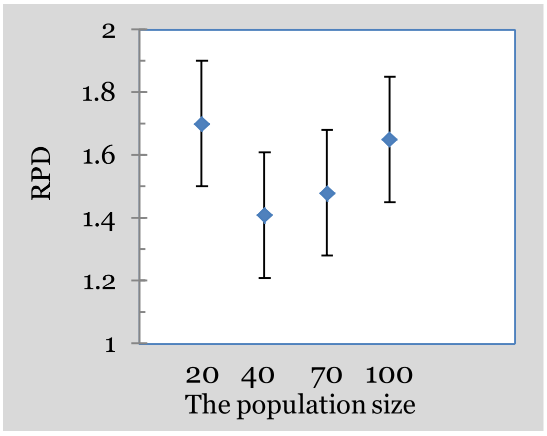
where Max_{sol} and Alg_{sol} are the best objective value obtained for each instance and solution of an algorithm in that given instance.

5.1. Parameter setting

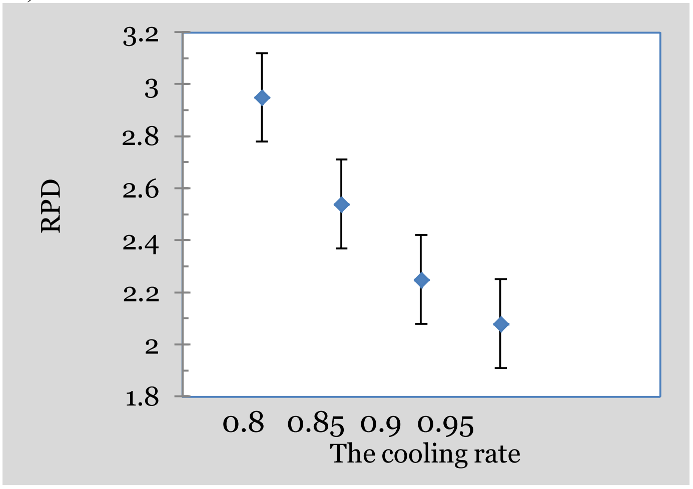
The parameter of ICA and GA is the population size. The parameter of SA is cooling rate. The considered population sizes are {20, 40, 70, 100}. The considered levels for cooling rate are {0.95, 0.90, 0.85, 0.8}. We generate 20 different instances. Then we solve them by the obtained algorithms. Figure 6 shows the results. As it can be seen, for ICA the best population size is 70 while this value for GA is 40. The best cooling rate is also 0.9.



a) ICA



b) GA



c) SA

Fig. 6. The average RPD and LSD intervals for the tested algorithms

5.2. Comparative experiment

This section the proposed algorithms are evaluated and compared on the set of instances mentioned earlier. Table 1 shows the results obtained by the algorithm, averaged by each combination n and m. Figure 7 shows the average RPD and least significant difference (LSD) intervals for the three tested algorithms. The best performing algorithm is ICA with the average RPD of 0.68%. GA obtains the second rank with the average RPD of 1.56% while the worst performing algorithm is SA with average RPD of 2.60%.

Table 5
The average RPDs obtained by the algorithms

n	m	Algorithms		
		GA	ICA	SA
10	2	1.11	0.72	1.92
	4	1.23	1.06	1.84
30	2	2.26	1.04	3.06
	4	1.02	0.94	2.29
50	2	1.49	0.17	2.92
	4	1.86	0.45	3.27
100	2	1.56	0.57	2.95
	4	1.93	0.49	3.25
average		1.56	0.68	2.69

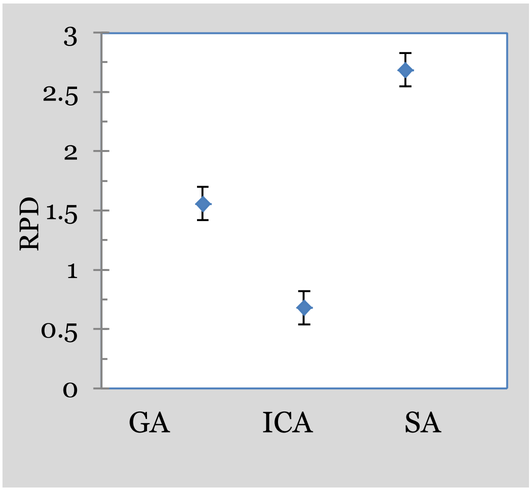


Fig. 7. Means plot and LSD intervals (at the 95% confidence level) for the different algorithms

To further analyze the results, we study the effect of problem characteristics such As the number of projects and the number of resources on the performance of the tested algorithms. Figures 8 and 9 show the performance of algorithms versus the number of projects and the number of resources, respectively. The performance of ICA becomes better while increasing the number of projects. Regarding the number of resources, ICA keeps its robust performance in different sizes.

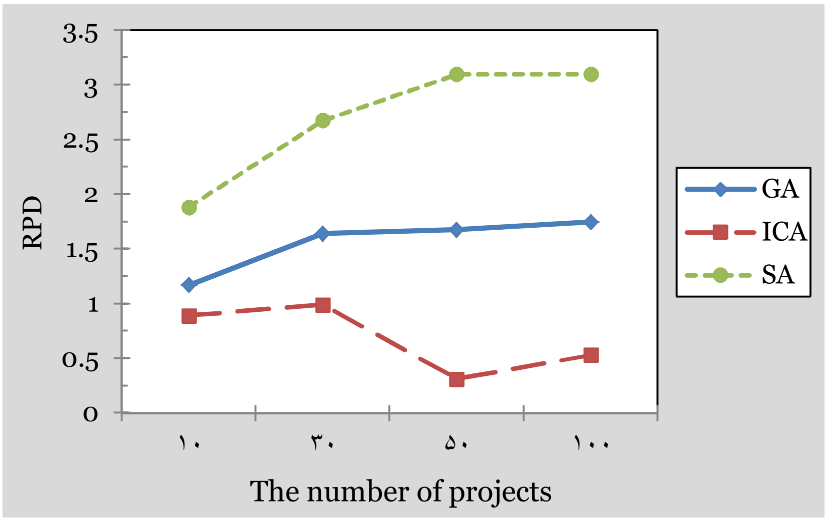


Fig. 8. The average RPD of the algorithms versus the number of projects

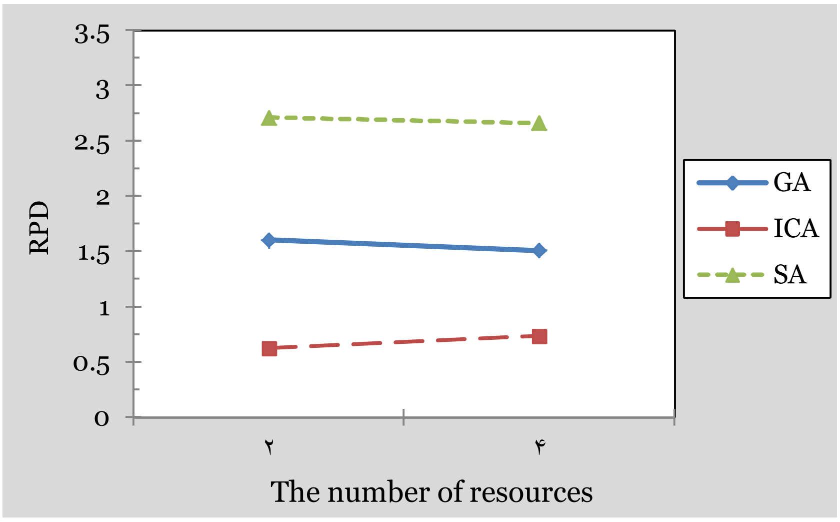


Fig. 9. The average RPD of the algorithms versus the number of resources

6. Conclusion and Future Research

This paper studied the problem of resource constrained project selection and scheduling. Each project requires consists of a set of tasks each of which consumes some resource to complete. There is a given time limit and the decision maker should select and schedule a subset of available projects that maximize the total profit. We first formulate the problem by a mixed integer linear programming model. Then to solve the problem, we developed three algorithms based on imperialist competitive algorithm, simulated annealing and genetic algorithm. An experiment was conducted and the performances of the algorithms are compared. The results showed that the imperialist competitive algorithm outperforms the other ones.

As a future research lead, it is interesting to develop the problem with some additional assumptions such as the interaction among the projects since commonly projects can share the resources. The multi-mode case is also interesting to work on. That is, a project can be performed in different model where each model needs different levels of resource and provide different profits.

7. References

[1] Aaker D.A., Tyebjee T.T., (1978). A model for the selection of interdependent R&D projects. IEEE Transactions on Engineering Management, 25, 30–36.

[2] Atashpaz-Gargari E., Lucas C., (2007). Imperialist competitive algorithm: an algorithm for optimization

inspired by imperialistic competition. IEEE Congress Evolutionary Computers, Singapore, 4661–4667.

[3] Atashpaz-Gargari E., Hashemzadeh F., Rajabioun R., Lucas C., (2008), Colonial competitive algorithm, a novel approach for PID controller design in MIMO distillation column process. International Journal of Intelligent Computation and Cyberntic, 1(3), 337–355.

[4] Bagher M., Zandieh M., Farsijani H., (2010). Balancing of stochastic U-type assembly lines: an imperialist competitive algorithm. International Journal of Advanced Manufacturing Technology, 54, 271–285.

[5] Banisadr A.H., Zandieh M., Mahdavi I., (2013), A hybrid imperialist competitive algorithm for single-machine scheduling problem with linear earliness and quadratic tardiness penalties. DOI 10.1007/s00170-012-4233-x.

[6] Brucker P., Knust S., Schoo A., Thiele O., (1998). A branch and bound algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research 107(2), 272–288.

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

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

[9] Demeulemeester E., Herroelen W., (1997). A branch-and-bound procedure for the generalized resource-constrained project scheduling problem. Operations Research 45 (2), 201–212.

[10] Davis, E.W., Patterson, J.H., (1975). A comparison of heuristic and optimum solutions in resource-constrained project scheduling. Management Science 21 (8), 944–955.

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

[12] Henriksen, A.D.P., Palocsay S.W., (2008). An Excel-based decision support system for scoring and ranking proposed R&D projects, International Journal of Information Technology and Decision Making, 7(3), 529–546.

[13] Holland J., (1975). Adaptation in natural and artificial systems. Ann Arbor: University of Michigan Press.

[14] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., (1983). Optimization by Simulated Annealing. Science 220, 671–680.

[15] 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.

[16] Kolon M., (1999). Some new results on simulated annealing applied to the job shop scheduling problem. European Journal of Operational Research, 113, 123–136.

[17] Mavrotas G., Diakoulaki D., Caloghirou Y. (2006), Project prioritization under policy restrictions: a combination of MCDA with 0–1 programming. European Journal of Operational Research, 171, 296–308.

[18] Montoya-Torres J.R., Gutierrez-Franco E., Pirachicán-Mayorga C., (2010). Project scheduling with limited resources using a genetic algorithm, International Journal of Project Management, 28(6) 619-628.

[19] Peng Y., Kou G., Shi Y., Chen Z., (2008). A descriptive framework for the field of data mining and knowledge discovery. International Journal of Information Technology and Decision Making, 7(4), 639–682.

[20] Stummer C., Kiesling E., Gutjahr W.J. (2009). A multi criteria decision support system for competence driven project portfolio selection, International Journal of Information Technology and Decision Making, 8(2), 379–401.

[21] Talias M.A., (2007). Optimal decision indices for R&D project evaluation in the pharmaceutical industry: Pearson index versus Gittins index. European Journal of Operational Research 177, 1105–1112.

[22] Yu L., Wang S. Wen F., Lai K.K., (2010). Genetic algorithm-based multi-criteria project portfolio selection, Annals of Operations Research, DOI 10.1007/s10479-010-0819-6.

[23] Zhang H., Li H., Tam C.M., (2006), Particle swarm optimization for resource-constrained project scheduling, International Journal of Project Management, 24(1) 83-92.

[24] Zhou W., Yan J., Li Y., Xia C., Zheng J., (2013). Imperialist competitive algorithm for assembly sequence planning. International Journal of Advanced Manufacturing Technology, DOI 10.1007/s00170-0124641-y.