Combinatorial versus Priority Based Optimization in Resource Constrained Project Scheduling Problems by Nature Inspired Metaheuristics

Silviu-Ioan BEJINARIU[1], Hariton COSTIN[1,2], Diana COSTIN[3]
[1]*Institute of Computer Science, Romanian Academy, Iaşi, 700481, Romania*
[2]*Faculty of Medical Bioengineering, Grigore T. Popa University of Medicine and Pharmacy, Iaşi, 700454, Romania*
[3]*Faculty of Medicine, Grigore T. Popa University of Medicine and Pharmacy, Iaşi, 700115, Romania*
hcostin@gmail.com

***Abstract*—This paper explores the behavior of the Flower Pollination Algorithm (FPA) and Particle Swarm Optimization (PSO) metaheuristic algorithm in resolving Resource Constrained Project Scheduling Problems (RCPSP) that can model certain practical issues in distributed applications. A RCPSP type problem has at the input a set of activities between which there are precedence relationships and for whose execution it is necessary to allocate resources that are limited. The solution determines the order of execution of the activities with respect to the precedence relations between them and the allocation of the available resources so that the total duration is minimal. The experimental results showed that a near optimal solution can be obtained faster than with other traditional algorithms, mainly for optimization problems in the continuous space. Two versions of FPA and PSO were used, namely combinatorial and priority based optimization. Because during evolution the individuals' position changes do not guarantee the precedence order preservation, a new tasks reordering procedure is proposed in this paper.**

***Index Terms*—biological information theory, evolutionary computation, optimization, particle swarm optimization, scheduling algorithms.**

I. Introduction

In economy, one of the key issues to be optimally solved is resources allocation in order to manage costs, working time, broaden access and improve general efficiency. For instance, an overview of recent Operating Research models developed for home health care routing and scheduling problem (HHCRSP) is presented in [1]. The HHCRSP is an extension of the vehicle routing problem with constraints that make it difficult to solve. The routes used by care workers to provide care to patients who live in the same geographic area and who must be treated at home have to be optimized in HHCRSP. Different objectives, as travel costs minimization or quality of services maximization, are used in this class of problems.

The use of biological inspiration algorithms is a relatively new approach to solve optimization problems. These algorithms are inspired by the strategies of living beings in the feeding process, for survival or perpetuation of the species. Also, some algorithms in this class are inspired by other phenomena or natural or artificial processes. Being part of the metaheuristic algorithms, the biological inspirational algorithms often allow a quick solution to be obtained close to the optimal solution, in complexity problems. So, a new research on using such algorithms in Resource Constrained Project Scheduling Problem (RCPSP) makes sense.

In [2] a novel Particle Swarm Optimization (PSO) based approach for RCPSP is proposed. It is based on two rules: delay local search rule and bidirectional scheduling rule which facilitate finding global minimum. The first rule enables some delayed activities by altering the starting time being capable of escaping from local minimum. The second rule combines forward and backward scheduling to expand the searching area for obtaining potential optimal solution. A hybrid combinatorial version of PSO is proposed in [3]. It is designed for the flowshop scheduling problem in which the makespan criterion has to be minimized. Different priority rules defined by experimental studies and statistical analysis are used in the initialization step of PSO. Another PSO based approach for RCPSP solving is presented in [4]. In [5] it is proposed a Discrete Flower Pollination Algorithm (DFPA) used to solve RCPSP. The DFPA is an adaptation of FPA for solving combinatorial optimization problems. In DFPA, some of the algorithm's core concepts, such as flower, global pollination, Lévy flight, local pollination, were redesigned.

The FPA [6] is used to solve various optimization problems, including medical image processing, processes optimization or structural engineering. A retinal blood vessels localization approach which uses the multi-objective version of FPA for image optimal clustering and Pattern Searching algorithm for segmentation results enhancement is proposed in [7]. In [8] FPA is used to find the optimal thresholds by maximization of between-class variance for multi-threshold image segmentation. A FPA-based approach for structural design optimization is proposed in [9]. The problems to be solved there have various design constraints concerning structural security measures and are related to pin-jointed plane frames, truss systems, deflection minimization of I-beams, tubular columns, and cantilever beams.

Digital Object Identifier 10.4316/AECE.2019.01003

In [10] the optimization objectives are: 1) minimizing the total transportation, and 2) maximizing the satisfaction of demand points (an interesting criterion).

Finally, it is worth to be mentioned the paper [11], which introduces a novel metaheuristic framework that allows multi-objective optimization of combinatorial problems, that would be an useful tool for an extension of our proposed work.

This paper is dedicated to the FPA and PSO metaheuristic algorithm in resolving RCPSP, and an objective comparison between them is made in order to better model certain practical issues in distributed applications.

The work presented in this paper continues the research of the authors in the nature-inspired (NI) optimization field. Mainly, NI algorithms were used for image registration procedures which require computing the parameters of a geometric transform. In this case the number of parameters is reduced and good results were obtained by applying the Bacterial Foraging Optimization Algorithm [12-14], Cuckoo Search and Bat algorithms [15]. Other approaches, as multispectral image fusion using Particle Swarming, Cuckoo Search and Fireworks algorithms were presented in [16]. A study of the nature inspired algorithms performances is presented in [17]. Because in image registration procedures the fitness function computing is time consuming, the speed can be improved by using parallel versions of the optimization algorithms [12].

The objective of the RCPSP problem is to compute the minimum time to complete a set of activities by respecting the precedence relations between them and allocating the required resources up to the limit of their availability.

The following information is used to define a RCPSP [5]:

$A=\{a_1,a_2,...,a_n\}$ – a set of n activities; a_1 and a_n are dummy activities representing the beginning and the end of the project,

$D=\{d_1,d_2,...,d_n\}$ – the duration of each activity; obviously the duration of the dummy activities is $d_1=d_n=0$,

$P=\{(a_i,a_j),1\le i,j\le n,i\ne j\}$ – a set of precedence relationships meaning that activity a_j can be started only after the a_i is completed; obviously, a_1 has no predecessors and a_n has no successors. The set of precedence relationships can be represented as a graph.

$R=\{r_1,r_2,...r_q\}$ – a set of q resources available in quantities $r_i,1\le i\le q$,

$B=\{\{b_{11},...b_{1q}\},...\{b_{n1},...b_{nq}\}\}$ – the resources required to complete each activity, where $b_{ij},1\le i\le n,1\le j\le q$ is the quantity of resource j required to complete activity i. The dummy activities do not need resources.

The problem is to determine the execution order of activities in A so that the total time is minimal, respecting the precedence relations and allocating the resources within the available limit, namely the set $T=\{t_1,t_2,...t_n\}$, where t_i is the planned time to start $a_i,1\le i\le n$. In fact, $t_1=0$ and t_n is the total time, because $d_n=0$.

The mathematical formulation of the RCPSP problem is:

$$\begin{cases}\min t_n & (1)\\ t_j-t_i\ge d_i,\forall i,j\ with\ (a_i,a_j)\in P & (2)\\ \sum_{a_i\in A_t}b_{ik}\le r_k,\forall k,1\le k\le q,where\\ \qquad A_t=\{a_i:t_i\le t\le t_i+d_i\} & (3)\end{cases}$$

where (1) is the objective function to be minimized, in restrictions (2) the precedence relationships are encoded, and (3) represents the set of restrictions related to the available resources allocation, where A_t is the set of activities a_i in progress at time t.

This is the simplest formulation of a RCPSP, in which the duration of each activity is specified by a single value (Single Time Estimate). In practice, the duration of activities is not fixed, so that the minimum, maximum and most probable time required to complete each activity are considered (Triple Time Estimate). The classical methods of RCPSP solving are part of graph theory: CPM (Critical Path Method) or PERT (Program Evaluation and Review Technique).

RCPSP can be solved as a combinatorial problem by computing the permutation $Z=\{z_1,z_2,...z_n\}$ of $\{1,2,...n\}$ which specifies the launching order of the activities in set A with the following meaning: if $z_i=k$ then activity a_k is in the i^{th} position in the launching order. Obviously, the dummy activities a_1 and a_n keep their original positions $z_1=1$ and $z_n=n$. The moment when each activity is launched $T=\{t_1,t_2,...t_n\}$ can be determined by a single parsing of the list of activities in the order specified by Z as it will be described in the following sections. Obviously, there are $n!$ possible solutions for a RCPSP, value which can be enough large in case of a great number of activities, so, the usage of nature-inspired metaheuristics is justified.

II. Optimization by Nature Inspired Metaheuristics

In the following paragraphs it is analysed the usage of two nature-inspired (NI) optimization algorithms for RCPSP solving. The NI algorithms are inspired form the nature intelligence and model the strategies used by life forms for survival, feeding or species perpetuation. Other NI algorithms are inspired from natural or artificial phenomena. Most of these algorithms were initially developed for optimization problems in the continuous space, where they are able to find near optimal solutions faster than classical algorithms [18]. For most of the NI algorithms, versions for multi-objective optimization, integer and binary programming were also developed.

In case of the single-objective optimization, the general form of the problem is

$$\min_{x\in S} f(x), S\subset R^d, \quad (4)$$

where S is the problem domain and it can also specify a set of constraints. The domain number of dimensions d is given by the number of parameters of the objective function f.

The structure of NI optimization algorithms is similar to most evolutionary algorithms (Fig. 1):

```
Initialize the set of possible solutions
Initialize the value of the best solution
```

```
while stop condition is not met perform
evolution loop
  build the next generation of solution
    candidates by applying the evolution
    strategy
  evaluate the value of the objective function
    for all new solution candidates
  apply the selection strategy to build the next
    generation of possible solutions
  update the value of the best solution
end while
```

Figure 1. General structure of nature-inspired algorithms

The NI algorithms are based on a set of individuals (*possible solutions*) which evolve in the *problem domain* using the *evolution strategy* that models the evolutionary strategy of the species from which the algorithm is inspired. The optimization problem's parameters are encoded as position of individuals. The *objective function* is evaluated in all the positions achieved by individuals during the evolutionary process. The algorithm continues until the *stop condition* is met. The solution of the problem is the position in which the best value of the objective function was reached. Usually, during evolution, the individuals move in the problem domain trying to optimize the objective. In some algorithms new individuals (descendants) are created in new positions so a *selection strategy* is applied to keep the populations size constant.

A. Particle Swarm Algorithm

The Particle Swarm Optimization (PSO) algorithm is one of the most efficient nature-inspired metaheuristic which is derived from the bird or fish swarming intelligence [19]. The *evolution strategy* is collaborative and individuals tend to move together toward the solution of the problem. This is done by using the personal and global experience when the moving direction is computed. In every step the particles move in directions which are mainly random, but components which direct them toward their personal best and global best positions are used.

Let x_i^t be the position of the i^{th} particle $x_i \in S$ in the t^{th} iteration. The new position is computed as:

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{5}$$

$$v_i^{t+1} = w \times v_i^t + \left(c_1 \times r_1 \times \left(x_i^{best} - x_i^t\right)\right) + \left(c_2 \times r2 \times \left(x^{best} - x_i^t\right)\right), \tag{6}$$

where v_i^t is the displacement of the i^{th} particle in the t^{th} iteration, w is the moving inertia coefficient, c_1 and c_2 are the local (personal) and global (social) learning coefficients, respectively, x_i^{best} is the best position reached by the i^{th} particle, x^{best} is the best position reached by any particle in the population and r_1, r_2 are random values. Using this strategy, the particles tend to swarm towards the solution of the problem. Multi Swarm Optimization [20] is an extension of PSO which is useful for multi-modal optimization problems. The particles are organized into swarms and the experience of the group is also used in the evolution strategy.

B. Flower Pollination Algorithm (FPA)

The FPA, proposed in [21] is inspired by the pollination process of flowering plants. Pollination can be achieved by self-pollination (pollen of the same flower or another flower of the same plant) or cross-pollination (pollen of another plant). The cross-pollination can be biotic when it is achieved by pollen vectors (insects or birds which fly a long distance), or abiotic when the pollination is helped by wind or water, usually on short distances. The pollination process was modelled [21-22] by the following rules used in FPA:

- biotic cross-pollination is considered as global pollination in which the vectors perform Lévy flight;
- self-pollination and abiotic pollination are considered as local pollination;
- the reproduction probability (flower constancy) is proportional to the similarity of the two flowers;
- local and global pollination is controlled by a switch probability.

Let $X = \{x_i, i = 1,...,n\}$ the individuals in the population, in this case flowering plants which are specified by their position in the optimization problem domain, $x_i \in S \subset R^d$, where d is equal to the number of parameters of the objective function. As it is described in the algorithm in Fig. 2, during a number of generations, each individual x_i is replaced by a new individual whose position is computed using the local or global pollination model, the selection being made using the switch probability, only if the value of the objective function in the new position is better. Global pollination is modelled [21] by:

$$x_i^{t+1} = x_i^t + L\left(x^{best} - x_i^t\right), \tag{7}$$

where x_i^t şi x_i^{t+1} are positions of the i^{th} individual in the t^{th} and $(t+1)^{th}$ iterations respectively, x^{best} is the position in which the best value of the objective function was computed and L is the step length computed using the Lévy distribution [21]:

$$L \cong \frac{\lambda \Gamma(\lambda) \sin(\pi \lambda/2)}{\pi s^{\lambda+1}}. \tag{8}$$

In (8), Γ is the gamma function, λ is constant and $s >> 0$ is a constant used to establish the step length. The local pollination is described by:

$$x_i^{t+1} = x_i^t + \varepsilon\left(x_j^t - x_k^t\right), \tag{9}$$

where ε is a random value with uniform distribution, x_j and x_k are two randomly chosen individuals with $j \neq k$. In Fig. 2 the general structure of the FPA is described.

```
generate X= {x_i, i = 1, ... n} the initial population in
    random positions
compute the objective function for each
    individual x_i
compute x^best the best solution in the initial
    population
for  t =1 to number of generations
  for each individual x_i ∈ X
    if  random < selection likelihood
      compute L step length with Lévy
        distribution
      Global pollination: x_i^{t+1} = x_i^t + L(x^best − x_i^t)
    else
      generate ε ∈ [0,1] a random value with
        uniform distribution
      randomly choose 2 individuals x_j şi x_k
      Local pollination: x_i^{t+1} = x_i^t + ε(x_j^t − x_k^t)
    end if
```

```
    evaluate the new solutions x_i^{t+1}
    if  the new solutions are better
      update the elements in X
    end if
  end for
  update the best solution x̄
end for
```

Figure 2. General structure of Flower Pollination algorithm [21]

The FPA was developed mainly for global optimization in the continuous space and it proved to be more efficient than PSO and Genetic Algorithms (GA). In literature, multiple variants of FPA are presented, including versions for integer or binary programming.

C. Combinatorial optimization using Particle Swarm algorithm

In [23-24] it is proposed an extension of PSO for solving combinatorial problems, named CPSO. Bellow, the proposed changes in the standard PSO are briefly presented.

In CPSO, each possible solution $x_i \in X$ is a permutation of $\{1,2,...n\}$ and it is accompanied by a status vector y_i defined as follows for each of its components:

$$y_{ij} = \begin{cases} 1 & if\ x_{ij} = x_j^{best} \\ -1 & if\ x_{ij} = x_{ij}^{best} \\ -1 or 1 & randomly, if\ x_{ij} = x_{ij}^{best} = x_j^{best} \\ 0 & otherwise, \end{cases} \quad (10)$$

where x_j^{best} is the j^{th} component of x^{best} best global position, and x_{ij}^{best} is the j^{th} component of x_i^{best}, the best position of the i^{th} particle. The position update is performed for each component of the position vector using (11), (12) and (13). First, the velocity is computed for each particle:

$$v_{ij}^{t+1} = w \times v_{ij}^t + \left(c_1 \times r_1 \times \left(-1 - y_{ij}^t\right)\right) + \left(c_2 \times r2 \times \left(1 - y_{ij}^t\right)\right) \quad (11)$$

Then, the values of y_{ij}^t are adjusted to compute the new position:

$$y_{ij}^{t+1} = \begin{cases} 1 & if\ x_{ij}^t + v_{ij}^{t+1} > \alpha \\ -1 & if\ x_{ij}^t + v_{ij}^{t+1} < -\alpha \\ 0 & otherwise, \end{cases} \quad (12)$$

where α is a parameter used for *intensification* and *diversification*. For small values of α, the components of the best global position or best personal position are used (*intensification*). For larger values of α new random values are used (*diversification*). Finally, the new position is computed by:

$$x_{ij}^{t+1} = \begin{cases} x_j^{best} & if\ y_{ij}^{t+1} = 1 \\ x_{ij}^{best} & if\ y_{ij}^{t+1} = -1 \\ random & otherwise \end{cases} \quad (13)$$

It is obvious that in the new position computing step, care should be taken to ensure that it is a permutation of $\{1,2,...,n\}$ and the precedence relations are verified.

D. Combinatorial optimization using Flower Pollination algorithm

One FPA-based RCPSP solving method is the *Discrete Flower Pollination Algorithm* (DFPA) proposed in [5]. Below, the differences between FPA and DFPA are shortly presented. Concerning the model used for individuals evolution in the problem's domain, the following operators are defined: *swap mutation* (the position of two randomly chosen activities are switched), *multiple swap mutation*, *inverse mutation* (the positions of all activities between two randomly selected activities are switched) and *crossover* (combines two randomly selected flowers) [5]. The *local pollination* is accomplished by applying the *crossover* operator: a sequence of tasks from the first flower is selected to create a new flower and all missing tasks are added in the free positions with respect of their order in the second flower. The *global pollination* requires a step length $s \in [0;1]$ to be computed using the Lévy distribution. The interval $[0;1]$ is divided in k subintervals of equal length. If $s \in [0;1/k]$ then *a swap mutation* is executed. If $s \in [i/k;(i+1)/k], i \in \{1,...k-2\}$ then a *multiple swap* which consists of $(i+1)$ *swap mutations*, is executed. For the larger values of the step length, $s \in [(k-1)/k;1]$ an *inverse mutation* is executed.

It must be noticed that in DFPA, in the *global pollination* a single individual is involved, unlike the standard FPA, which uses also the current best position x^{best} to compute the new solution candidate.

E. Precedence relations in solutions candidates

During the evolutionary process, it is obvious that not all possible solutions (permutations) are generated. Some solutions are generated more than once and there are also solutions which do not check the precedence relations. Since these solutions do not solve the problem, it can be assumed that in these cases the execution time of the project is ∞ without modifying the algorithm. But as it will be seen in the following paragraphs, the number of these *invalid* solutions is large enough, which reduces the possibility to find the optimal solution of the RCPSP problem.

The problem is to sort the elements of the input permutation to obtain a valid permutation which meets the precedence conditions, using a minimum number of changes in the original permutations so that the elements that are not related keep as much as possible their relative order. The classical sorting algorithms can't be used for such ordering because the precedence relation is not a total order relation. It is not defined for tasks located on different paths on the associated directed graph. In this case, a topological ordering (Kahn's algorithm) has to be applied. Unfortunately, the topological ordering leads always to the same solution if the successors of an activity are not processed in the order specified by the processed permutation. Even if in the algorithm the successors are processed in the order of their occurrence in the permutation, this is considerably altered, making it difficult to obtain the known solution. To solve these issues, an original sorting procedure is proposed in this paragraph (Fig. 3). It is based on the usage of a temporary array which contains the position of each task in the permutation. All the pairs of tasks between which there is a precedence relation are checked and swapped if their order is incorrect in the permutation. The procedure is resumed from the new

position of the current element, which is a position prior to the original one.

```
SortByPrecedence
Build the Index array which contains for each
activity its position in the permutation
for i=1 to n
  changed = false
  for each succ -successor of sol(i) AND not changed
    if  Index(sol(i)) > Index(succ)
      swap sol(i) and succ in the permutation
      spawn the corresponding elements in the
      index array
      changed = true
    end if
  end for
  if  changed
    i = Index(sol(i))
  end if
end for
```

Figure 3. Sort algorithm for permutation validity checking and reordering

The proposed sorting method allows obtaining a valid permutation in which the precedence conditions are met, using a minimum number of changes in the original permutations.

F. *Transforming the permutations into solutions*

Each permutation specifies the order in which the activities are started. The starting time of each activity is then computed according to the completion of previous activities and resources availability. During the solution computing, a list of events is built up. Each event contains information about the current time, the list of current activities which includes their status in relation to the event time (*start*, *end*, *ongoing*) and the resources availability at that time. A single parsing of the activities in the order specified by the permutation allows the solution to be determined by building the events list (Fig. 4).

```
Build_Solution
Create an event at time 0 and add the dummy
    activity 1 with start and end status, the
    resources are not changed
for each  activity i in the order specified by
    the permutation
  find event_1 with the maximum time at which a
      predecessor activity of i is completed
  while required resources for activity i are not
      free go to next event in the list
  end while
  add activity i in the list of current event
      with status start and update resources
  find event_2 whose time is equal to the end time
      of activity i
  if such an event does not exist, a new event 2
      is created in the proper position
  add activity i in the list of event_2 with
      status end and update resources
  add activity i with status ongoing to all
      events between event_1 and event_2 and update
      resources
end for
for each event in the list of events are
    identified the activities with status start;
  the time of the event is the starting time for
      these activities
end for
the time required to complete the project is the
      starting time of the last activity.
```

Figure 4. Algorithm for transforming a permutation of activities into the solution

It is obvious that the solution computed using the proposed algorithm is not unique. There may be situations when the resources required for an activity are available for a longer period than the duration of the activity. In this case, the activity may be delayed, but the change is local and does not affect the overall duration of the project.

III. Experiments and Results

First, the proposed permutation sorting method to solve RCPSP using PSO and FPA was validated and then the results obtained by the two optimization methods were compared to those reported in [5]. The RCPSP library available on the PSPLIB site of TUM School of Management, Technical University of Munich was used in experiments [25].

A. *Validation of the permutation sorting procedure*

The "Patterson" set [25], which includes 100 problems, each with up to 51 tasks and up to 3 resources to be shared, was used to validate the permutation sorting method proposed in the previous sections. In the next paragraphs, the results obtained for three RCPSP problems (Table I) which were considered representative are presented.

Table I. RCPSP Problems Used for Validation

RCPSP problem	Number of tasks	Number of resources	Solution
pat1	14	3	19
pat100	27	3	33
pat110	51	3	50

Considering that the permutations generated by the algorithm can be *invalid* in the sense that they do not necessarily check the precedence conditions, two tests were carried out, with and without applying the sorting procedure. The results obtained by applying FPA are presented in Table II and Table III. The following parameters of FPA were used: number of iterations – 2000, number of individuals (flowers) – 20 and switch probability – 0.8.

Concerning the contents of the two tables, it should be mentioned that:

- the optimal solution of the problems is known [25];
- the number of optimal permutations of the activities set is not necessarily equal to that of the optimal solutions because for different permutations, identical time allocations of activities can be obtained (case when more activities start simultaneously);
- the valid permutations are those for which the precedence conditions are met.

Table II. Results Obtained in Case of Unsorted Permutations

RCPSP problem	Number of tasks	Total number of permutations	Number of generated permutations			Number of generated optimal permutations		Known solution
			total	unique	valid	total	unique	
pat1	14	479.001.600	40020	8047	1091	31883	1024	19
pat100	27	~1.55e+25	40020	38550	-	-	-	33*
pat110	51	~6.08e+62	40020	39276	-	-	-	50*

The following conclusions can be drawn by analysing the results obtained without applying the sorting procedure (Table II):

- in case of a reduced number of tasks (pat1), the algorithm is able to generate valid permutations, including some

permutations which correspond to the optimal solutions;
- if the number of tasks is greater (pat100 and pat110) the set of generated permutations does not even include a valid permutation. However there is the possibility to obtain solutions by modifying the parameters of the algorithm (a greater number of iterations and/or individuals).

In the second experiment (Table III), because the sorting procedure was applied, all the generated permutations are also valid from the precedence relations point of view.

TABLE III. RESULTS OBTAINED IN CASE OF SORTED PERMUTATIONS

RCPSP problem	Number of tasks	Total number of permutations	Number of generated permutations		Number of generated optimal permutations		Solution
			total	unique	total	unique	
pat1	14	479.001.600	40020	1335	39705	1181	19
pat100	27	~1.55e+25	40020	11196	33119	8860	33
pat110	51	~6.08e+62	40020	19129	33760	15935	50

In this case, the proposed method allows obtaining the optimal solution for all three RCPSP problems, even if the number of tasks is greater. Also the number of unique optimal permutations is enough great, which leads to the idea that the algorithm can be used to solve more complex problems. The optimal solution was obtained for all 110 RCPSP problems in the "Patterson" set with the notice that for some of them it was necessary to increase the number of iterations from 2000 to 5000.

B. Comparison of results obtained by using PSO and FPA

In the experiment presented below the results obtained by using FPA and PSO are compared, using both optimization methods: combinatorial and priority based. The priority based optimization is performed in the continuous space. The coordinates of individual's positions are real numbers in the $[0;1]^n$ hyper-interval, where n is the number of tasks in the optimization problem. The value of each coordinate is considered as the priority of the corresponding task in the project. More specifically, for each solution the coordinates are sorted in increasing order to obtain a permutation of tasks. As in the combinatorial optimization, the obtained permutation does not necessarily verify the precedence conditions, so the sorting by precedence procedure has to be applied.

To compare the results to those reported in [5], the same set of 10 optimization problems from [25] was used. In [25] a large amount of benchmarks with 30, 60, 90 and 120 tasks is available. In all these, the number of resources is 4 and each task has no more than 3 successors. In the last part of this section, some results obtained in case of problems with 120 tasks are presented.

The two optimization algorithms were applied using the following parameters:
- FPA combinatorial (FPA-C): #flowers=50, #iterations=100, switch probability=0.8;
- FPA priority based (FPA-P): #flowers=50, #iterations=100, switch probability=0.8;
- PSO combinatorial (PSO-C): #particles=50, #iterations=100, inertia weight $w = 0.75$, personal cognitive weight $c_1 = 0.2$, social cognitive weight $c_2 = 0.8$;
- PSO priority based (PSO-P): #particles=50, #iterations=100, inertia weight $w = 0.729$, personal cognitive weight $c_1 = 1.49445$, social cognitive weight $c_2 = 1.49445$;

The number of individuals and iterations was determined by experiments and they were chosen so that the number of objective function evaluations is similar in all cases. The other parameter of FPA-C and FPA-P was determined by experiments. The weights used in PSO-C are those proposed in [24] and the weights used in PSO-P are those proposed in [26]. Because the number of iterations and individuals (flowers / particles) have a great impact on the required computing resources, especially on the processing time, their values were chosen to allow obtaining the known solution for most of the test problems. In the case of more complex problems, these values need to be increased, as will be outlined in the next paragraphs. The other parameters influence the convergence speed. For each optimization problem 100 runs of each algorithm were performed, with the results presented in Table IV and Table V. The column *Known solution* contains the optimal solution as it is specified in [25]. The columns *Min*, *Max* and *Avg* show the minimum, maximum and average optimal solutions respectively, determined as results of the 100 runs of each algorithm. The column Err% contains the percentage deviation of the average solution from the optimal solution and it is computed using (14) [5].

$$Err = \frac{average\ solution - optimal\ solution}{optimal\ solution} \times 100 . \quad (14)$$

The last column, *Err%**, contains the deviation percent reported in [5]. These values were obtained using DFPA with 20 individuals in the population and 1000 iterations which means that about four times more solution candidates were generated.

TABLE IV. RESULTS OBTAINED USING FPA

Problem	Known solution	Solutions - combinatorial				Solutions – Priority based				Err%*
		Min	Max	Avg	Err%	Min	Max	Avg	Err%	
J3006_2	51	51	51	51.00	0	51	51	51.00	0	1.06
J3015_4	48	48	48	48.00	0	48	48	48.00	0	0.29
J3020_1	57	57	57	57.00	0	57	57	57.00	0	0.24
J3026_6	53	53	54	53.66	1.25	54	55	54.10	2.07	0.34
J3029_4	103	104	104	104.00	0.97	104	105	104.22	1.18	0.47
J3034_4	67	67	67	67.00	0	67	67	67.00	0	0.42
J3039_3	54	54	54	54.00	0	54	54	54.00	0	0.22
J3042_8	82	82	82	82.00	0	82	82	82.00	0	0.41
J3045_2	125	125	125	125.00	0	125	126	125.01	0.01	0.56
J3048_2	54	54	54	54.00	0	54	54	54.00	0	0.33

TABLE V. RESULTS OBTAINED USING PSO

Problem	Known solution	Solutions - combinatorial				Solutions – Priority based				Err%*
		Min	Max	Avg	Err%	Min	Max	Avg	Err%	
J3006_2	51	51	52	51.18	0.35	51	52	51.13	0.25	1.06
J3015_4	48	48	48	48.00	0	48	48	48.00	0	0.29
J3020_1	57	57	57	57.00	0	57	57	57.00	0	0.24
J3026_6	53	53	55	54.07	2.02	53	55	54.04	1.96	0.34
J3029_4	103	104	105	104.15	1.11	104	105	104.16	1.12	0.47
J3034_4	67	67	67	67.00	0	67	67	67.00	0	0.42
J3039_3	54	54	54	54.00	0	54	54	54.00	0	0.22
J3042_8	82	82	82	82.00	0	82	82	82.00	0	0.41
J3045_2	125	125	126	125.08	0.06	125	126	125.06	0.05	0.56
J3048_2	54	54	54	54.00	0	54	54	54.00	0	0.33

By analysing the results presented in Table IV and Table V the following conclusions can be drawn:

- The optimal solution was reached by FPA-C, PSO-C and PSO-P for 9 benchmark problems and by FPA-P for 8 problems;
- The optimal solution was obtained in all 100 executions of the algorithm by FPA-C for 8 problems and by FPA-P for 7 problems;
- For 6 problems, PSO-C and PSO-P have obtained the optimal solutions in all 100 executions of the optimization algorithm;
- For the *J3029_4* problem, the optimal solution was never obtained.
- The results obtained in all 4 experiments are better than those reported in [5] in terms of percentage of the deviation for 8 benchmark problems.

For problems *J3026_6* and *J3029_4* the results were improved by increasing the number of iterations (Table VI) and in case of FPA-C, the error became lower than that reported in the mentioned paper. Unfortunately, none of FPA-P, PSO-C and PSO-P was able to obtain the known optimal solution for problem *J3029_4*.

TABLE VI. RESULTS OBTAINED AFTER THE NUMBER OF ITERATIONS WAS INCREASED TO 400

Alg	Problem	Known solution	Solutions - combinatorial				Solutions – Priority based				Err %*
			Min	Max	Avg	Err%	Min	Max	Avg	Err%	
FPA	J3026_6	53	53	54	53.10	0.19	53	54	53.80	1.51	0.34
	J3029_4	103	103	104	103.99	0.96	104	104	104.00	0.97	0.47
PSO	J3026_6	53	53	55	53.72	1.35	53	55	53.84	1.58	0.34
	J3029_4	103	104	105	104.09	1.05	104	105	104.10	1.07	0.47

In Table VII an analysis on the number of unique (U), optimal (O) and unique optimal (OU) solutions generated during the evolution by all four algorithms used in the experiment is presented. The unicity was considered in terms of the time established to start each task, not of generated permutations, because different permutations can lead to the same solution. Because the number of individuals (50) and iterations (100) was the same in all cases, the total number of generated solutions was 5050 and is not shown. The permutation sort procedure was applied, so, all solutions are valid. It must be mentioned also that all the values in the Table VII were obtained for a single execution of the algorithms.

TABLE VII. UNIQUE, OPTIMAL AND UNIQUE OPTIMAL SOLUTIONS

	FPA-C			FPA-P			PSO-C			PSO-P		
Problem	U	O	OU	U	O	OU	U	O	OU	U	O	OU
J3006_2	288	3411	14	394	198	8	438	4062	7	657	1439	13
J3015_4	936	5025	927	746	4976	736	170	5044	64	1157	4875	1021
J3020_1	1	5050	1	1	5050	1	1	5050	1	1	5050	1
J3026_6	374	253	2	411	10	1	608	3450	7	2334	339	7
J3029_4	600	2848	206	601	204	10	1230	3622	67	4373	68	14
J3034_4	10	4951	3	10	4966	3	11	4995	3	14	2817	3
J3039_3	23	4761	1	21	4283	1	27	4854	1	55	1355	1
J3042_8	964	4834	848	704	4351	542	358	4813	139	2835	1704	531
J3045_2	1398	3811	955	1235	17	1	1551	3704	284	4241	824	451
J3048_2	1	5050	1	1	5050	1	1	5050	1	1	5050	1

The following conclusions can be also drawn:

- First of all, there is no rule linking the number of solutions to the algorithm used. Rather it seems that the number of solutions depends on the optimization problem and on how favourable or not are the possible solutions generated during the execution of the algorithms;
- Comparing the number of solutions generated by the combinatorial and priority based versions of the two algorithms it seems that the second one generates less optimal solutions even if in some cases the number of unique optimal solution is greater than in the first case;
- For the problems *J3020_1* and *J3048_2*, all the 5050 generated solutions are optimal and there is only one unique solution which is also optimal. From this, we can conclude that for each of these optimization problems the precedence conditions lead to a unique solution which is optimal. In fact for all generated sequences of tasks, the sorting procedure offers always the same result;
- Concerning the problem *J3029_4* for which the known optimal solution was not obtained, we can conclude that the optimization processes are trapped in some local solutions and the result can be improved only by a better adjustment of the algorithms parameters.

All the algorithms were implemented in C++ and the experiments were made using a Core i3 2.4 GHz processor based computer with 6 GB RAM. The average processing time for the experiments presented in Table IV and Table V is presented below.

TABLE VIII. AVERAGE PROCESSING TIME, IN SECONDS

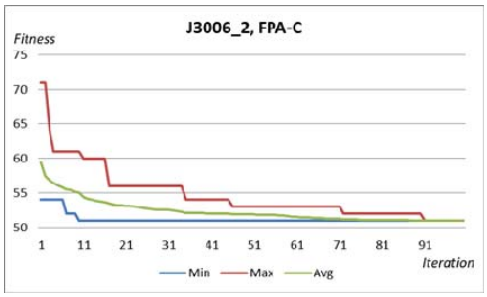
Optimization algorithm	FPA		PSO	
Optimization type	Combinatorial	Priority based	Combinatorial	Priority based
Average processing time (sec)	0.39	0.45	0.35	0.40

By analysing the values in Table VIII, it is obvious that on the one hand PSO is faster than FPA and on the other hand the combinatorial versions are faster than the priority based versions. First, PSO is faster because in the evolution loop the particles new position computing is based on arithmetic operation only while in FPA the new position computing requires an exponential and a sine function evaluation. Concerning the difference between combinatorial and priority based optimization, it is caused by the fact that in the second case, each fitness function evaluation requires the priorities list to be sorted in order to compute the permutation of tasks. Even if the solution was known, in all experiments the stop condition of the algorithm was to reach the maximum number of iterations.

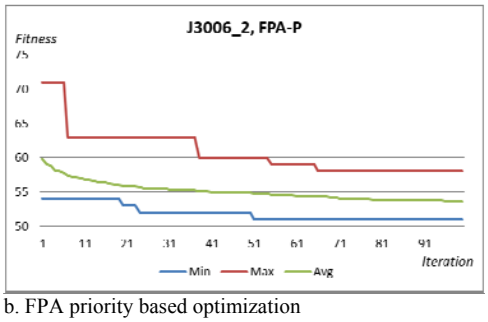
In Fig. 5 and Fig. 6 the evolution of fitness during the 100 iterations is presented for two problems with 30 tasks: J3006_2 and J3042_8. The minimum, maximum and average fitness of the 50 flowers / particles are presented.

Analysing the results obtained for problem J3006_2, the following conclusions can be drawn:

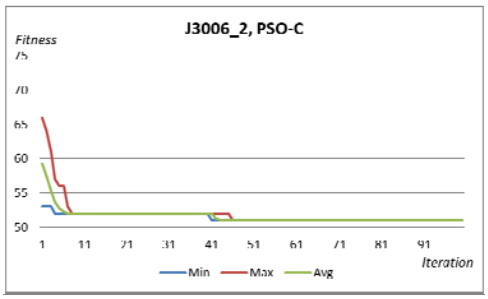
- the swarming tendency is obvious, excepting the case of priority based FPA; the fitness value of all individuals reaches the optimum value in iteration 91 for FPA-C, iteration 46 for PSO-C and iteration 29 for PSO-P;



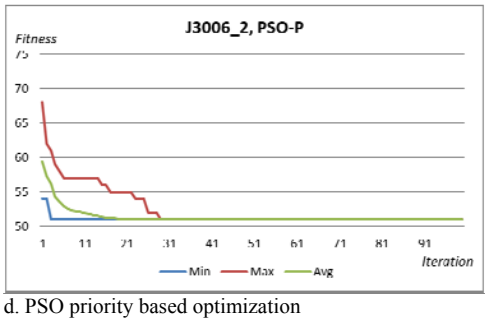
a. FPA combinatorial optimization



b. FPA priority based optimization

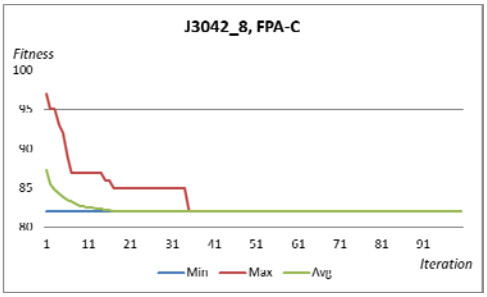


c. PSO combinatorial optimization

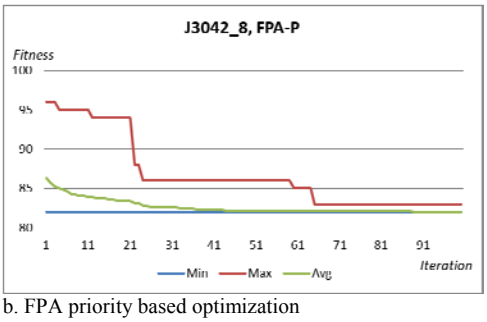


d. PSO priority based optimization

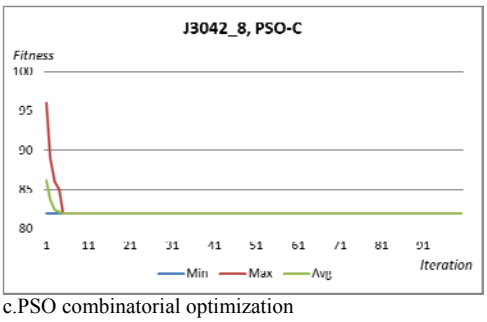
Figure 5. Problem J3006_2, evolution of the minimum, maximum and average value of individuals fitness function, respectively



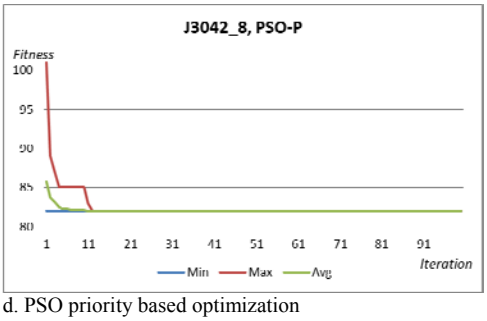
a. FPA combinatorial optimization



b. FPA priority based optimization



c.PSO combinatorial optimization



d. PSO priority based optimization

Figure 6. Problem J3042_8, evolution of the minimum, maximum and average value of individuals fitness function, respectively

- the optimum fitness is reached in iteration 51 by FPA-P, but many other individuals remain trapped up to end in other local solutions of the problem;
- the three values are equal between iterations 8 and 40 of PSO-C but the value is not optimum; this demonstrates that it is not advisable to stop the algorithm before reaching the maximum number of iterations, at least in case of optimization problems in the discrete space;
- PSO-P is the first algorithm that reaches the optimum value, in the 3rd iteration;
- for this problem, it seems that PSO offers the best results.

The results obtained for problem J3042_8 lead to the following conclusions:

- the optimum solution of the problem is found by all algorithms in the first iteration; this is explained by the large number or optimum solutions, as can be seen in Table VII;
- the optimum value is reached faster in case of PSO by all individuals: iteration 5 for PSO-C, iteration 12 for PSO-P and iteration 35 for FPA-C; also in this case, after 100 FPA-P iterations does not reach the optimal value with all individuals, even the average value is close to the optimum;
- also for this problem, it seems that PSO offers the best results;

By analysing the charts in Fig. 5 and Fig. 6 it can be concluded that the combinatorial versions of the two algorithms offer better results than the priority based versions.

In the following paragraphs, some conclusions related to more complex problems are presented. As it was mentioned above PSPLIB include RCPSP problems with 30, 60, 90 and 120 tasks. We made some test with problem which contains 120 tasks each one with at most 3 successors, and 4

resources which must be shared. In this case, the number of permutations is 120! which is about $6.69*10^{198}$. Besides the number of activities, the complexity of the problems arises from the number of required resources. If only one resource is required to complete a task, the scheduling can be computed using the four methods (FPA-C, FPA-P, PSO-C and PSO-P) with the same parameters as in the case of problems with 30 tasks (50 individuals and 100 iterations). Some examples of such problems are X04_8, X24_2 and X44_7 [25] whose results are presented in tables below.

TABLE IX. RESULTS OBTAINED FOR PROBLEMS WITH 120 TASKS

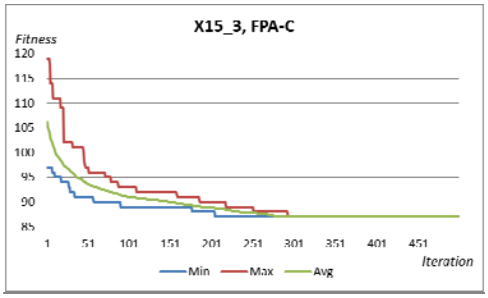
Alg.	Problem	Known solution	Solutions - combinatorial				Solutions – Priority based			
			Min	Max	Avg	Err%	Min	Max	Avg	Err%
FPA	X04_8	90	90	90	90.00	0	90	90	90.00	0
	X24_2	91	91	94	93.60	2.86	91	95	93.94	3.23
	X44_7	98	98	98	98.00	0	98	99	98.03	0.03
PSO	X04_8	90	90	90	90.00	0	90	90	90.00	0
	X24_2	91	91	95	93.82	3.10	91	95	93.69	2.96
	X44_7	98	98	98	98.00	0	98	102	98.08	0.08

TABLE X. UNIQUE, OPTIMAL AND UNIQUE OPTIMAL SOLUTIONS DETERMINED DURING A SINGLE RUN

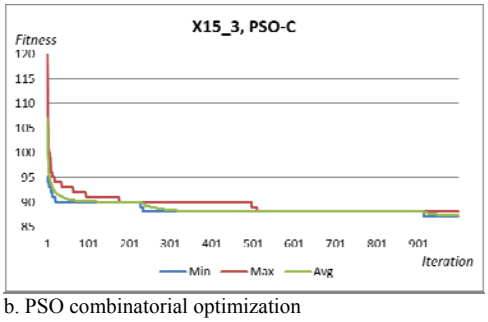
	FPA-C			FPA-P			PSO-C			PSO-P		
Problem	U	O	OU	U	O	OU	U	O	OU	U	O	OU
X04_8	2515	4987	2478	3080	4917	3002	336	5018	311	69	5027	50
X24_2	994	2638	352	858	182	7	504	4686	342	276	45	3
X44_7	1804	2925	939	1428	176	24	997	4562	781	236	3625	47

By analysing the values presented in Table IX and Table X it is obvious that none of the methods used can be considered as the best. The second problem (X24_2) for which the error percent in not 0, the fewest unique solutions were determined, except for PSO-C. However it must be noticed that the values in Table X refer a single run of the algorithms which are based on randomly chosen initial possible solutions.

Fig. 7 shows the results obtained for a problem in which almost all tasks require all four resources to can be completed.



a. FPA combinatorial optimization



b. PSO combinatorial optimization

Figure 7. Problem X15_3, evolution of the minimum, maximum and average value of individuals fitness function, respectively

To obtain these results the number of iterations and individuals were increased as follows: FPA-C: 500 iterations and 100 individuals; PSO-C: 1000 iterations and 100 individuals. FPA-C reaches the best solution in iteration 205 and PSO-C in iteration 915. Also for this problem, PSO-C has a large number of iterations, between 511 and 914 in which all the individuals have the same best fitness value. So, for more than 400 iterations the value of all individual best fitness is not enhanced without being equal to the problem's solution. It must be mentioned also that for FPA-C, the personal best fitness of all individuals became equal to the best fitness in iteration 293, but this does not happen in 1000 iterations of PSO-C. In Table XI the numbers of unique, optimal and unique optimal solutions are presented, respectively. The number of generated possible solutions is: 500.000 in FPA-C and 100.000 in PSO-C. As in previous problems, FPA-C generates more unique optimal solutions than PSO-C.

TABLE XI. UNIQUE, OPTIMAL AND UNIQUE OPTIMAL SOLUTIONS DETERMINED DURING A SINGLE RUN

	FPA-C			PSO-C		
Problem	U	O	OU	U	O	OU
15_3	3657	23846	201	1447	4780	29

Concerning the execution time of the optimization procedures for RCPSP problems with 120 tasks, as it can be seen in Table XII, it is much higher than in case of problems with 30 tasks. This is caused by the fact that each generated possible solution (permutation of 120 values) requires to be reordered to meet the precedence relations.

TABLE XII. AVERAGE PROCESSING TIME, IN SECONDS

Optimization algorithm	**FPA**				**PSO**			
Optimization type	Combinatorial		Priority based		Combinatorial		Priority based	
Iterations	500	1000	500	1000	500	1000	500	1000
Average time (sec)	26.32	49.75	31.75	63.23	26.86	54.21	29.28	58.85

In the table above, the processing time is presented for all four methods with both 500 and 1000 iterations even if not all these combinations led to the known optimal solution. Also, the execution time was obtained using a sequential implementation in C++ of the algorithms, as Windows application and the processing unit was shared with other running applications.

It must be noticed also that for some RCPSP problems with 120 tasks which are more complex, the number of iterations and individuals should be greatly increased making the algorithms difficult to execute on a usual computer. Optimal implementations of the algorithms should be considered in this case.

IV. CONCLUSIONS

In this paper, the capability of two Nature-Inspired algorithms - Flower Pollination and Particle Swarm Optimisation - for RCPSP problems solving is studied. The advantage of these NI algorithms is that a near optimal solution can be obtained faster than for other classical algorithms. Mainly this is true for optimization problems in the continuous space. For optimization in the discrete space or combinatorial optimization problems, these algorithms require some modification not only in data encoding and storage but also in the evolutionary strategies used. In the

experiments, two versions of FPA and PSO were used, namely: combinatorial and priority based optimization, respectively. When applied for RCPSP problems, in the individual's position it is encoded the launching order of the tasks. Because during evolution the individuals' position change does not guarantee the precedence order preservation, a tasks reordering procedure is proposed in this paper. The comparison with other results presented in literature has revealed that at least for small RCPSP problems with 30 tasks, the proposed procedure offers clearly better results:

- the deviation error is lower, in fact the optimal solution was obtained in all executions of the optimization algorithm for 8 problems by FPA-C, for 7 problems by FPA-P, and for 6 problems by PSO-C and PSO-P;
- for the other problems the results were improved by increasing the number of individuals and / or iterations;
- sometimes the minimum and maximum fitness values are equal for all the solution candidates in a large number of consecutive iterations and then the fitness value begins to decrease again to the best solution; this demonstrates that it is not advisable to stop the algorithm before reaching the maximum number of iterations – at least in case of combinatorial optimization;
- for higher dimensional problems, in which the number of each task required resources is higher, the number of individuals / iterations has to be increased, case in which the processing time increases significantly;
- none of the methods used can be considered as being the best; it seems that FPA is a little bit more precise, and also the priority based versions are slower due to the required conversions from floating point representation of individuals positions (priorities) into integer values that specifies the permutations.

The work will be continued by optimizing the procedures to allow faster processing in case of higher dimensional problems and also by studying other nature inspired algorithms capabilities for RCPSP problem solving.

REFERENCES

[1] M. Cissé, S. Yalçındağ, Y. Kergosien, E. Şahin, C. Lenté, A. Matta, "OR problems related to home health care: a review of relevant routing and scheduling problems", Operations Research for Health Care, Vols. 13-14, pp. 1–22, 2017, doi:10.1016/j.orhc.2017.06.001.

[2] R. M. Chen, C. L. Wub, C. M. Wang, S. T. Lo, "Using novel particle swarm optimization scheme to solve resource-constrained scheduling problem in PSPLIB", Expert Systems with Applications, Vol. 37, pp. 1899–1910, 2010, doi:10.1016/j.eswa.2009.07.024.

[3] M. Eddaly, B. Jarboui, P. Siarry, "Combinatorial particle swarm optimization for solving blocking flowshop scheduling problem", Journal of Computational Design and Engineering, Vol. 3, pp. 295–311, 2016, doi:10.1016/j.jcde.2016.05.001.

[4] H. Zhang, H. Li, C.M. Tam, "Particle swarm optimization for resource-constrained project scheduling", International Journal of Project Management, Vol. 24, pp. 83–92, 2006, doi:10.1016/j.ijproman.2005.06.006.

[5] K. Bibiks, J. P. Li, F. Hu, "Discrete flower pollination algorithm for resource constrained project scheduling problem", International Journal of Computer Science and Information Security, Vol. 13(7), pp. 8-19, 2015.

[6] X.-S. Yang, "Flower pollination algorithm for global optimization", in Unconventional Computation and Natural Computation, Lecture Notes in Computer Science, Vol. 7445, pp. 240-249, 2012, doi:10.1007/978-3-642-32894-7_27.

[7] E. Emary, H. M. Zawbaa, A. E. Hassanien, B. Parv, "Multi-objective retinal vessel localization using flower pollination search algorithm with pattern search", Advances in Data Analysis and Classification, Vol. 11, No. 3, pp. 611-627, 2017, doi:10.1007/s11634-016-0257-7.

[8] R. Wang, Y. Zhou, C. Zhao, H. Wu, "A hybrid flower pollination algorithm based modified randomized location for multi-threshold medical image segmentation", Bio-Medical Materials and Engineering, Vol. 26, pp. 1345-1351, 2015, doi:10.3233/BME-151432.

[9] S. M. Nigdeli, G. Bekdaş, X.-S. Yang, "Application of the flower pollination algorithm in structural engineering", in X.-S. Yang et al. (Eds.), Metaheuristics and Optimization in Civil Engineering, Modeling and Optimization in Science and Technologies, Vol. 7, Springer, pp. 25-42, 2016, http://doi.org/10.1007/978-3-319-26245-1_2.

[10] A. Goli, A. Aazami, A. Jabbarzadeh, "Accelerated cuckoo optimization algorithm for capacitated vehicle routing problem in competitive conditions", International Journal of Artificial Intelligence, vol. 16, no. 1, pp. 88-112, Mar. 2018.

[11] J. Ruiz-Rangel, C. J. Ardila Hernandez, L. M. Gonzalez, D. J. Molinares, "ERNEAD: training of artificial neural networks based on a genetic algorithm and finite automata theory", International Journal of Artificial Intelligence, vol. 16, no. 1, pp. 214-253, Mar. 2018.

[12] S.-I. Bejinariu, H. Costin, F. Rotaru, R. Luca, C. Niţă, C. Lazăr, "Parallel processing and bio-inspired computing for biomedical image registration", Computer Science Journal of Moldova, Vol. 22, No. 2(65), pp. 253-277, 2014.

[13] H. Costin, S.-I. Bejinariu, "Medical image registration by means of a bio-inspired optimization strategy", Computer Science Journal of Moldova, Vol. 20, No. 2(59), pp. 178-202, 2012.

[14] H. Costin, S.-I. Bejinariu, D. Costin, "Biomedical image registration by means of bacterial foraging paradigm", International Journal of Computers, Communications & Control, Vol. 11, No. 3, pp. 329-345, 2016, doi:doi:10.15837/ijccc.2016.3.1860.

[15] S.-I. Bejinariu, H. Costin, F. Rotaru, R. Luca, C. Niţă, "Image processing by means of some bio-inspired optimization algorithms", Proc. of the IEEE 5th Int. Conference on E-Health and Bioengineering – EHB 2015, Iaşi, România, 2015, pp. 1-4, doi:10.1109/EHB.2015.7391356.

[16] S.-I. Bejinariu, R. Luca, H. Costin, "Nature-inspired algorithms based multispectral image fusion", Proc. of the 2016 International Conference and Exposition on Electrical and Power Engineering, Iaşi, România, pp. 1-5, 2016, doi:10.1109/ICEPE.2016.7781293.

[17] S.-I. Bejinariu, H. Costin, F. Rotaru, R. Luca, C. Niţă, "Performance analysis of artificial bee colony optimization algorithm", in Proc. of the 13-th Int. Symposium on Signals, Circuits and Systems, ISSCS 2017, Iaşi, România, pp. 1-4, 2017, doi:10.1109/ISSCS.2017.8034903.

[18] X.-S. Yang, Nature-Inspired Optimization Algorithms. Elsevier Inc., pp. 23-173, 2014, ISBN: 0124167438 9780124167438.

[19] J. Kennedy, R. Eberhart, "Particle swarm optimization", Proc. of the IEEE Int. Conference on Neural Networks, Perth, WA, Australia, Vol. 4, pp. 1942-1948, 1995, doi:10.1109/ICNN.1995.488968 .

[20] T. Hendtlass, "WoSP: a multi-optima particle swarm algorithm", Proc. of the IEEE Congress on Evolutionary Computation, Edinburgh, Scotland, UK, pp. 727–734, 2005, doi:10.1109/CEC.2005.1554755.

[21] X.-S. Yang, M. Karamanoglu, X.S. He, "Flower pollination algorithm: a novel approach for multiobjective optimization", Engineering Optimization, Vol. 46, No. 9, pp. 1222-1237, 2014, doi:10.1080/0305215X.2013.832237.

[22] X.-S. Yang, M. Karamanoglu, X.S. He, "Multi-objective flower algorithm for optimization", Procedia Computer Science, Vol. 18, pp. 861-868, 2013, doi:10.1016/j.procs.2013.05.251.

[23] B. Jarboui, M. Cheikh, P. Siarry, A. Rebai, "Combinatorial particle swarm optimization (CPSO) for partitional clustering problem", Applied Mathematics and Computation, Vol. 192, pp. 337–345, 2007, doi:10.1016/j.amc.2007.03.010.

[24] B. Jarboui, N. Damak, P. Siarry, A. Rebai, "A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems", Applied Mathematics and Computation, Vol. 195, pp. 299-308, 2008, doi:10.1016/j.amc.2007.04.096.

[25] PSPLIB, Project Scheduling Problem Library – PSPLIB, http://www.om-db.wi.tum.de/psplib/main.html (Accessed 4 December 2017).

[26] R. C. Eberhart. Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization", Proc. of the Congress on Evolutionary Computation, La Jolla, CA, USA, Vol. 1, pp. 84-88, 2000, doi:10.1109/CEC.2000.870279.