2019

Greedy Binary Particle Swarm Optimization for multi- Objective Constrained Next Release Problem

Abeer Hamdy
The British University in Egypt, abeer.hamdy@bue.edu.eg

Ali Mohamed
The British University in Egypt

Greedy Binary Particle Swarm Optimization for multi-Objective Constrained Next Release Problem

A. Hamdy and A. A. Mohamed

***Abstract*—Software companies, that adopt agile methodologies in the development of a large and complex software product, encounter the problem of selecting the subset of requirements to be included in the next release of the product. This problem is known as the next release problem (NRP). NRP is identified as an NP-hard problem as it involves a set of conflicting objectives that need to be addressed. These objectives are as follows: (1) maximizing the customer satisfaction taking into consideration that each customer has an importance level to the company, and (2) minimizing the development cost such that it does not exceed the allocated budget. Furthermore, the requirements have dependency and precedence relationships, and each requirement has a priority for each customer according to the customer's needs. Therefore, metaheuristic algorithms are good candidate for tackling this problem. This paper proposes a hybrid approach to address the multi-objective constrained NRP. The proposed approach is based on adapting an improved binary particle swarm optimization (IBPSO) algorithm. Additionally, a greedy methodology was utilized for swarm initialization to seed the swarm with good solutions. Experimental results, of over six small and large NRP instances, demonstrated that the proposed approach converges much faster to solutions better than the ones discovered by the original binary PSO.**

***Index Terms*—Next release problem, binary PSO, swarm intelligence, requirements engineering.**

I. Introduction

During the last decade agile methodologies [1] have gained a substantial popularity among software development companies. In these methodologies, the software product is developed through releases that have to be produced within iterative cycles. Each release includes a new subset of the requirements that should meet the needs of the customers and should be developed within the allocated budget for each iteration [2]. The following constraints must be taken into consideration: (1) The customers have different importance levels to the company, (2) the precedence and dependency relationships among the requirements such that some requirements cannot be developed before others, (3) each requirement has a development cost and (4) requirements have different priority levels to the different customers. This problem is known in the literature as the next release problem (NRP) [3], [4].

The larger the set of requirements, the more complex the NRP gets; due to the huge number of possible combinations that can be formed by all the features. In addition to considering the aforementioned constraints, this context leaves the decision maker with 2^n possible combinations to decide from. This is why the NRP is considered one of the complex combinatorial optimization problems [3]. Maximizing the total satisfaction of the customers, and minimizing the company's development cost/effort, are two conflicting objectives that categorize the NRP as a multi-objective optimization problem which is best tackled by metaheuristic algorithms [4]. Some researchers tackled the NRP problem using PSO [5]-[9], ant colony optimization (ACO) [5], [10]-[12], genetic algorithms (GA) [5], [16], [13]-[15]. Some previous research proved that the original BPSO (OBPSO) surpasses both of the ant colony and the GA [3] in solving the NRP, especially the large scale NRP.

This paper proposes a solution to the constrained multi-objective NRP using a hybrid approach that adapts an improved version of the BPSO (IBPSO) [16] and a greedy methodology for the swarm initialization.

A. Research Questions

The paper aims at answering the following research questions:

RQ1: Does the performance of the IBPSO surpass the performance of the OBPSO in solving the multi objectives, constrained NRP?

The aim of this question is to compare the performance of the adapted IBPSO algorithm and the adapted OBPSO in solving different sizes of the NRP instances.

RQ2: Does the proposed greedy-random swarm initialization method improve the performance of the IBPSO and the OBPSO?

The aim of this question is to assess the ability of the proposed greedy initialization methodology on guiding the BPSO to discover better solutions.

RQ3: Is the hybrid greedy-random IBPSO scalable?

The aim of this question is to compare the performance of the proposed approach over small and large NRP instances.

The rest of the paper is organized as follows: Section II discusses the previous studies that addressed the NRP. Section III describes the multi-objective constrained NRP. Section IV introduces the PSO algorithm. Section V discusses the proposed approach. While, Section VI discusses the experimental design and the datasets. Section VII discusses the experimental results and provides answers

Manuscript received May 14, 2019; revised August 1, 2019.

A. Hamdy is with the Faculty of Informatics and Computer Science, British University in Egypt, El-Sherouk City, Egypt, on secondment from Computers and Systems Department, Electronics Research Institute, Giza, Egypt (e-mail: abeer.hamdy@bue.edu.eg).

A. A. Mohamed is with the Faculty of Informatics and Computer Science, British University in Egypt, El-Sherouk City, Egypt (e-mail: ali.alaa.eldin@outlook.com).

doi: 10.18178/ijmlc.2019.9.5.840

to the research questions. Finally, Section VIII concludes the paper and suggests further extensions to this work.

II. Related Work

NRP was proved to be NP-complete [17]; so several research studies have been conducted to solve the NRP or to prioritize the requirements using metaheuristic algorithms [4]. Bagnall *et al.* [3] compared greedy randomized adaptive search procedure (GRASP), hill climber and simulated annealing (SA) algorithms in solving different sizes of the NRP; and they found that the SA outperformed both the GRASP and the hill climber especially with large scale NRP instances. Glauber *et al*. [5] investigated solving the large scale NRP using two metaheuristic algorithms, the ant colony optimization (ACO) and the particle swarm optimization (PSO). The researchers aimed to identify which metaheuristic algorithm is more suitable for handling the large scale NRP. They proved that PSO algorithm achieved much better performance than ACO and the PSO's best results were obtained with the higher number of particles. Moreover, their experiments showed that ACO can achieve same performance of PSO for small sizes of the NRP. However, the work of [3] and [5] did not consider the interactions among the requirements. Three research studies considered the interactions among the requirements [4], [11], [12]. Jose *et. al.* [4] used differential evolution algorithm [18], while Sagrado *et al*. [11] and Souza *et al*, [12] used ACO algorithms.

III. Multi-Objective Next Release Problem

In this paper the NRP is formulated as follows:

A software product has a set of requirements of size *m* $R = \{r_1, r_2, \ldots, r_m\}$. Each of these requirements has an associated cost value e_i, which represents the amount of resources or the development cost/effort needed for this requirement. The dependency and precedence among these requirements are represented as a directed acyclic graph, in which there exist a set of nodes and a set of edges, where each node represents a requirement and each edge represents a dependency between the nodes.

Given an edge (r_j, r_i) in which r_i is the child node of the parent r_j; this means that the requirement r_i depends on the requirement r_j. In this paper we consider one functional interdependence among the requirements which is REQUIRE (r_i, r_j). This interdependency was selected as it is the most popular in software projects [19]. A REQUIRE (r_i, r_j) interdependency indicates that requirement a r_i cannot be selected to the next release if r_j is not selected also or has been developed in a previous release [4], [11], [12]. But requirement r_j can be included in the next release without r_i. Fig. 1 shows an example to the dependencies among the requirements and the customers' requests for certain requirements. In this example, r_2 is in the first level, which means that r_2 is the parent of all the requirements and is required to be developed first.

Furthermore, there is a set of customers of size *k* (C_1, C_2, …, C_k); each customer has an importance to the company. Each customer requests a subset of requirements and maybe more than one customer is interested in the same requirement. So there will be a value v_{ij} associated with each requirement *i* and customer *j* denotes the satisfaction of the customer when including this requirement in the next release. The cost of a release is the cost of all the requirements selected to be developed in the next release. The objective of the NRP is to determine the subset of requirements that will be included in the product's next release, such that the total satisfaction of the customers is maximized and the release's development cost is minimized; furthermore, the cost does not exceed the company's budget.

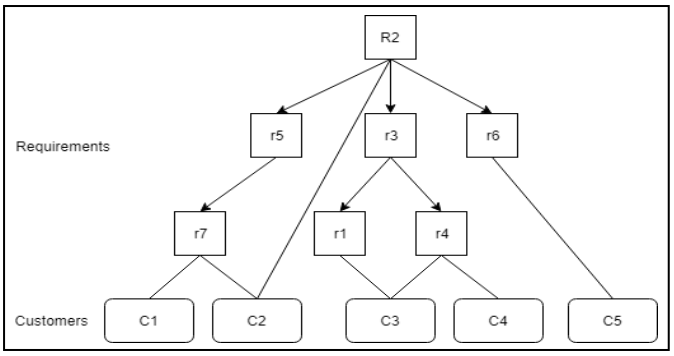

Fig. 1. Requirements precedence and customers request example.

IV. Particle Swarm Optimization

PSO is one of the swarm intelligence algorithms that was designed by Eberhart and Kennedy in 1995 [6-9]. It is inspired by the social behavior of the bird flocks that collaboratively work together to reach the point that has the most resources. The whole flock is called the swarm, and each bird in the swarm is called a particle. Each particle represents a solution in the search space and has three attributes, namely, velocity, position and best explored position by the particle. The velocity attribute guides the particle to move to its next position. The particle's position is updated every iteration according to the particle's current velocity value, the global best position that was found by the swarm, and the best explored position found by the particle. The PSO algorithm iterates for a predetermined number of iterations or until a minimum error value is achieved.

V. Proposed BPSO Strategies for the NRP

The proposed strategy starts by initializing the swarm with solutions to the NRP in the feasible search space, where the NRP constraints are valid. Then, each particle in the swarm is evaluated and their positions are updated according to their velocity vectors. The velocity vectors guide the particles toward better solutions. These steps are repeated until the swarm reaches a point where no better solutions are found, or the iteration count is reached. The following subsections discuss the formal description of the fitness function used in evaluating the particles, the particles encoding, the swarm initialization strategies and the steps of both of the IBPSO and OBPSO.

A. Multi-Objective Fitness Function Formulation

Assume that $R_{NR} = \{r_1, r_2, \ldots, r_n\}$ is a subset of the set of requirements R, which are selected to be developed for the

next release of a software package and they satisfy the requirements' precedence constraints. Each of these requirements r_i has an associated cost e_i, that represents the development effort needed for this requirement. Let $E = \{e_1, e_2, \ldots, e_n\}$ be the set of costs that are associated to the n requirements. These requirements are enhancements or extensions to the current software system that are endorsed by a set of k clients $C = \{c_1, c_2, \ldots, c_k\}$. Each client has a grade of importance for the company that is represented by a weight. The weights of the set of k clients are denoted by $W = \{w_1, w_2, \ldots, w_k\}$. Each requirement r_j has an importance to the client c_i, and this importance is defined by a value $v_{ij} > 0$. A zero value for v_{ij} means that the client c_i has not suggested the requirement r_j for the next release. The total satisfaction s_j of a requirement r_j is calculated as the weighted sum of its importance values for all the clients that suggested it, which is expressed by (1):

$$s_j = \sum_{i=1}^{k} v_{ij} * w_i \quad (1)$$

The overall satisfaction produced by the set of requirements selected for the next release is calculated as the sum of the satisfactions of this set of requirements R_{NR} which is calculated by (2):

$$Satisf(R_{NR}) = \sum_{j=1}^{n} s_j \quad (2)$$

The cost of developing R_{NR} is calculated as the summation of the cost of developing all the requirements included in R_{NR} which is given by (3):

$$Cost(R_{NR}) = \sum_{j=1}^{n} e_j \quad (3)$$

The fitness function is formulated in a way to maximize the global satisfaction and minimize the development cost of the release which is given by (4):

$$\text{Particle's fitness} = 0.7\frac{Satisf(R_{NR})}{Satisf(R)} + 0.3\frac{\text{Cost}(R)}{\text{Cost}(R_{NR})} \quad (4)$$

$$\text{Such that: } Cost(R_{NR}) \leq Budget$$

where, $Budget$ is the budget allocated by the company for developing the release.

B. Solution Encoding

x_{i1}	x_{i2}	x_{i3}	x_{i4}	x_{i5}	x_{i6}	x_{i7}	x_{i8}	x_{i9}	x_{i10}
1	0	1	0	0	1	1	1	0	1

Fig. 2. A sample position vector X_i of a particle P_i.

In PSO each particle P_i has a position vector X_i which represents a solution to the tackled problem. We encoded the solution as a binary string of size m, $Xi = (x_{i1}, x_{i2}, \ldots, x_{im})$. Where each bit x_{ij} corresponds to a requirement r_j, the bit value is equal to zero or one. The value of one means that the requirement r_j is selected for the next release, while the value of zero means that the r_j is not selected. Fig. 2 shows a sample particle of size m=10 in which r_1, r_3, r_6, r_7, r_8, and r_{10} are selected and r_2, r_4, r_5, and r_9 are not selected for the next release. The particle position vector is mutated every iteration depending on the particle's velocity vector according to the BPSO algorithm utilized (the OBPSO or the IBPSO).

C. Swarm Initialization

The swarm holds a population of solutions to the problem. The selection of the initial swarm is very important; it directly affects the quality of the best solution found. In this paper two strategies were used to initialize the swarm which are: Random and Greedy-Random strategies.

1) Swarm random initialization

Each particle's position X_i is randomly initialized within the feasible search space of the NRP, which must comply to the following constraints:

1) Requirement precedence relationship.
2) The particle's cost does not exceed the release's allocated budget.

Initialization procedure starts by adding randomly to the particle a requirement and its precedence requirements. If the total cost of the requirements included in the particle exceeds the release budget, this action is undone. This process is repeated until the particle's cost is equal to the release's budget.

Procedure 1: Random initialization

Input: $X_i = (x_{i1}, x_{i2}, \ldots, x_{im})$, $i \in \{1, swarm\_size\}$

For each particle's position vector X_i

- Repeat till all the requirements are included or

Budget - Cost (X_i) ≤ epsilon

- Generate a random number $j \in (1, m)$
- Set the value of the bit x_{ij} to 1

If precedence constraints of the requirement r_j is not satisfied,

Include all r_j precedencies in X_i

If Cost (X_i) – budget ≤ epsilon

Remove r_j and its added precedencies

(Undo latest changes)

2) Swarm greedy-random initialization

Greedy approach is utilized to seed the initial population with valuable feasible solutions. Valuable solutions are selected based on the values of the requirements. The value of a requirement is calculated as the summation of the satisfactions of this requirement and its precedencies divided by the summation of its cost and the cost of its precedencies. Greedy-random Initialization procedure proceeds as follows:

Procedure 2: Greedy-random initialization

Input: $Xi = (xi_1, xi_2, \ldots, xim)$, $i \in \{1, swarm\_size\}$

1. Calculate the value of each requirement.
2. Arrange the requirements in a descending order based on their values.
3. Split the swarm into three compartments

Initialize the first compartment using greedy as follows:

- For a particle i, add to the particle requirements starting from the i^{th} most valuable one and add its precedencies.
- Check out the budget constraint after each step.

- Stop when budget - Cost (X_i) ≤ epsilon

Initialize the second compartment using Greedy-Random as follows:

- At random add to the particle one of the five most valuable requirements and its precedencies
- Include more requirements randomly till the budget – Cost (X_i) ≤ epsilon

Initialize the third compartment according to the random initialization procedure

D. Binary Particle Swarm Optimization

PSO was originally developed for continuous valued spaces but in this work, we use binary encoding for the particles. Therefore, two binary PSO algorithms were implemented in this paper that is the original BPSO developed by Kennedy and Eberhart [8] and the improved BPSO developed by Mojtaba *et al.* [16]. The following subsections introduce the both algorithms.

1) Original BPSO

The original BPSO (OBPSO) differs from the PSO in that the velocity vector V_i of the i[th] particle P_i represents the probabilities of the corresponding bits of the particle's position vector X_i to mutate from its current state to another. For example, if the j[th] position in the particle's velocity vector v_{ij} equals to 0.40, it means that there is a 40% chance that x_{ij} will be equal to "one" and a 60% chance it will be equal to zero. v_{ij} is updated according to (5):

$$v_{ij}(k+1) = w \cdot v_{ij}(k) + c_1 r_1 \left(Xbest_{ij} - x_{ij}(k)\right) + c_2 r_2 \left(Xgbest_j - x_{ij}(k)\right) \quad (5)$$

where, c_1 and c_2 are two parameters representing the particle's confidence in itself (local behavior) and in the swarm (social behavior) respectively. The values of these two parameters are very important as they control the balance between exploration and inclinations. Higher values of c_1 encourage the particles to move toward their local best positions, while higher values of c_2 cause faster convergence to the global best position.

r_1 and r_2 are two random numbers uniformly distributed in the range (0, 1), while, w is the inertia weight. An upper bound V_{max} was set to the components of the velocity vector. V_{max} and w maintain the balance between the local and the global search [8], [9]. $Xbest_i$ and $Xgbest$ are the best positions experienced so far by the *i*[th] particle and the whole swarm, respectively.

In order to keep v_{ij} in the range (0, 1), a sigmoid transformation function is applied as given by (6):

$$V'_{ij} = sig(V_{ij}) = \frac{1}{1 + e^{-V_{ij}}} \quad (6)$$

The *j*[th] position of the *i*[th] particle is updated according to (7):

$$x_{ij}(t+1) = \begin{cases} 1 & \text{, if } r_{ij} < sig\left(v_{ij}(t+1)\right) \\ 0 & \text{, otherwise} \end{cases} \quad (7)$$

where, r_{ij} is a random uniform number in the range (0, 1).

The algorithm of the OBPSO to solve NRP can be summarized as follows:

Procedure 3: OBPSO for NRP

1. Initialize the BPSO swarm using either Random or Greedy – Random strategies.
2. For each particle *i* do the following:
   - Compute the fitness of the particle using its current position vector $F(X_i)$
   - If the particle's current fitness is higher than its best experienced position, ($F(X_i) > F(Xbest_i)$), then replace the particle's best position with the current position ($Xbest_i = X_i$).
   - If particle's best position is better than the global best position (F ($Xbest_i$) > F ($Xgbest$)), then replace the global best position with this particle's best-position ($Xgbest = Xbest_i$).
   - Update the particle's velocity vector V_i according to (5) and (6).
   - Update the particle's position vector X_i according to (7).
   - If the particle is not in the feasible search space, mutate the particle to meet the precedencies and the budget constraints.
3. Go to step 2 and repeat until convergence or for a predetermined number of iterations.

2) Improved BPSO

The improved BPSO (IBPSO) defines two more velocity vectors for each particle V_i^0 , V_i^1. Where, V_i^0 ,V_i^1 are the probabilities of the bits of the particle's position vector to change to 0, or to change to 1, respectively. V_i^0 *and* V_i^1 are not complement. The velocity vector of the particle V_i^c is defined by (8):

$$V_{ij}^c = \begin{cases} V_{ij}^1 & \text{, if } x_{ij} = 0 \\ V_{ij}^0 & \text{, if } x_{ij} = 1 \end{cases} \quad (8)$$

where, V_{ij}^c is the probability of change in the *j*[th] bit of the *i*[th] particle position vector X_i.

The vectors V_i^0 , V_i^1 are updated according to the particle's best position ($Xbest_i$) and the global best position ($Xgbest$) vectors. Such that if the *j*[th] bit in $Xgbest$ or $Xbest_i$ is equal to zero the velocity V_{ij}^0 is increased and the probability of changing to one V_{ij}^1 is decreased with the same rate. Similarly, if the *j*[th] bit in $Xgbest$ or $Xbest_i$ is equal to one V_{ij}^1 is increased and V_{ij}^0 is decreased. The advantage of the IBPSO over the OBPSO is that the bits of the particle's position vector benefit from the previously found direction of change to one or to zero. Equations 9 and 10 show the calculation of the j[th] bit of the velocity vectors V_i^1 and V_i^0.

$$V_{ij}^1 = w \cdot V_{ij}^1 + d_{ij,1}^1 + d_{ij,2}^1 \quad (9)$$

$$V_{ij}^0 = w \cdot V_{ij}^0 + d_{ij,1}^0 + d_{ij,2}^0 \quad (10)$$

where, w is the inertia weight and the values of $d_{ij,1}^0$, $d_{ij,1}^1$, $d_{ij,2}^0$,and $d_{ij,2}^1$ are calculated according to (11), (12), (13)

and (14):

$$\text{if } x\text{best}_i^j = 1 \text{ Then } d_{ij,1}^1 = c_1 r_1 \text{ and } d_{ij,1}^0 = -c_1 r_1 \quad (11)$$

$$\text{if } x\text{best}_i^j = 0 \text{ Then } d_{ij,1}^0 = c_1 r_1 \text{ and } d_{ij,1}^1 = -c_1 r_1 \quad (12)$$

$$\text{if } xg\text{best}^j = 1 \text{ Then } d_{ij,2}^1 = c_2 r_2 \text{ and } d_{ij,2}^0 = -c_1 r_1 \quad (13)$$

$$\text{if } xg\text{best}^j = 0 \text{ Then } d_{ij,2}^0 = c_2 r_2 \text{ and } d_{ij,2}^1 = -c_1 r_1 \quad (14)$$

where, r_1 and r_2 are two random numbers in the range (0,1). c_1 and c_2 are defined parameters.

The velocity vector V_i^c is calculated and normalized using the sigmoid function defined by (6); then the particle position vector is updated according to (15):

$$x_{ij}(t+1) = \begin{cases} \overline{x}_{ij} & ,if\ r_{ij} < Sig\ (V_{ij}^c) \\ x_{ij} & ,if\ r_{ij} > Sig\ (V_{ij}^c) \end{cases} \quad (15)$$

where, r_{ij} is a random uniform number in the range (0, 1). Equation 15 implies that if r_{ij} is less than the current velocity, then the current position bit is changed to the 2nd complement of itself; that is, if x_{ij} is 0 then $\overline{x}_{ij}$ is 1 or vice versa. On the other hand, if r_{ij} is greater than the current velocity, then the current position bit remains the same. The algorithm of IBPSO proceeds similarly to the OBPSO except the approach of calculating the position and the velocity vectors.

VI. Experimental Methodology

All the experiments conducted in this paper have been run under the same environment. On the hardware side, we used Intel(R) Core(TM) i5-6402P CPU @ 2.80GHz (4CPUs), ~2.8GHz, Memory: 8192MB RAM. On the software side we used Operating System: Windows 10 Pro 64-bit (10.0, Build 17134). As the BPSO is a stochastic algorithm, 25 independent runs have been carried out for each experiment. The results provided in the paper are the arithmetic mean of the results of these independent runs. The arithmetic mean is a valid statistical measurement because the results follow a normal distribution.

Two NRP datasets were used in the experiments to assess the proposed approach. Furthermore, the budget of the release (total development effort of the release) was restricted to three limits 30%, 50%, and 70% of the total development effort $Cost(R)$ for each dataset. So, it could be said that 6 instances of an NRP were utilized in assessing the proposed approach. The datasets are copied from [4], [11], and [20]. The first dataset comprises 30 requirements and 5 clients. Table I shows the development effort/cost of each requirement, the priority level assigned to each requirement by each client, and the requirement interactions. Priority levels take values from 1 to 5: Value "1": means not important requirement, value "2": minor requirement, value "3": important requirement, value "4": highly important requirement, and value "5": tremendously important. These priority levels were used in measuring the total satisfaction of a requirement. The development effort of each requirement is defined on a scale from 1 to 10. It should be pointed out that this dataset is not large however we used it to assess the effectiveness of the proposed approach over small-scale NRP. The second dataset is more complex than the first one. It comprises 100 requirements, 5 clients and forty four requirement interactions. Table II lists the development effort of each requirement, the priority level assigned to each requirement by each client, and the interactions. In dataset#2, the development costs of the requirements are taken from real agile software projects. The development effort of a requirement is established in 20 effort units, which can be translated into 4 weeks. The values of the priority levels in this dataset are between one and three, as the clients prefer to use a coarse-grained scale to indicate the benefit of the inclusion of a requirement. The value 'one' indicates inessential, 'two' indicates desirable and 'three' indicates mandatory [21], [22]. Furthermore, for the two datasets, each client is assigned a relative importance in the decision making of the company. A client's importance in the two datasets has values in the range of 1 to 5, where value of '1' means the less important client, while value of '5' means the most important client. Table III shows the clients' relative importance for both datasets.

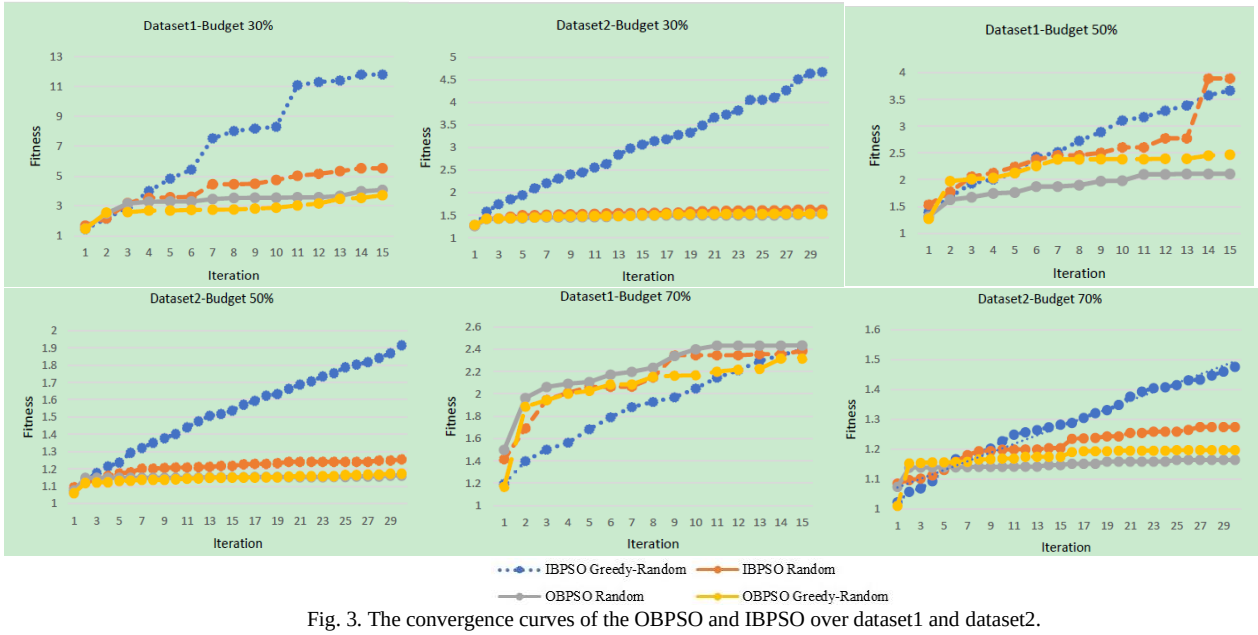

Fig. 3. The convergence curves of the OBPSO and IBPSO over dataset1 and dataset2.

The experiments were designed to answer the previously stated three research questions. The setup of the experiments is listed in Table IV.

VII. Results and Discussion

This section This section discusses the results of the experiments and provides answers to the research questions. Tables V, VI and VII list the mean results of the experiments, while Fig. 3 shows the average convergence curves of the experiments over the six NRP instances.

As observed from Table V, the IBPSO algorithm surpasses the OBPSO in solving the NRP across the six NRP instances. It could be observed also that the proposed greedy random initialization algorithm enhanced the performance of the IBPSO especially across the difficult NRP instances (the two instances with budget limit 30% of the total effort), while it improved the performance of the OBPSO across two instances of the NRP. However, as observed from Table VI, the greedy random initiation boosted the robustness of both the OBPSO and the IBPSO over the small and the large datasets. The robustness of the evolutionary algorithms is measured using the standard deviation. The smaller the standard deviation, the more robust the algorithm. Small standard deviation means that the algorithm is capable of finding acceptable solutions in the different runs, with small discrepancy.

We also analysed the execution time required by the algorithms to find the best solution. The execution time values are listed in Table VII, which shows the average execution time in seconds. As observed, the running time of the IBPSO, to find the best solution, is smaller than the OBPSO. Furthermore, the greedy random initialization assisted the IBPSO to converge faster to the best solution, while it did not benefit the OBPSO.

VIII. Conclusions and Future Work

An improved approach for BPSO was adapted to solve the constrained multi-objective NRP taking into considerations the dependency relations among the requirements. Moreover, the original BPSO algorithm was adapted to solve the same problem. The two algorithms were assessed using three NRP instances of a small dataset and three NRP instances of a large dataset.

It was found that the IBPSO surpasses the OBPSO and not only that but also its execution time was extremely faster than the OBPSO.

It was found that the greedy random initialization speeded up the search for the best solution in the case of utilizing the IBPSO, while its influence on the BPSO was minimal.

As a further extension to this work, the IBPSO approach could be extended to find the Pareto front instead of a particular best solution. Also, the performance of the IBPSO could be compared with other recent computational intelligence techniques like the bat and cuckoo search algorithms.

Appendix

TABLE I: Dataset1: Requirements Development Cost, Dependencies and Importance (Priority) for Each Customer

Effort		r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}
		1	4	2	3	4	7	10	2	1	3	2	5	8	2	1	4	10	4	8	4
Priority Level	c_1	4	2	1	2	5	5	2	4	4	4	2	3	4	2	4	4	4	1	3	2
	c_2	4	4	2	2	4	5	1	4	4	5	2	3	2	4	4	2	3	2	3	1
	c_3	5	3	3	3	4	5	2	4	4	4	2	4	1	5	4	1	2	3	3	2
	c_4	4	5	2	3	3	4	2	4	2	3	5	2	3	2	4	3	5	4	3	2
	c_5	5	4	2	4	5	4	2	4	5	2	4	5	3	4	4	1	1	2	4	1

Interactions

Requirement	→	Dependencies
r_4	→	r_8, r_{17}
r_8	→	r_{17}
r_9	→	r_6, r_{12}, r_{19}
r_{11}	→	r_{19}

TABLE II: Dataset2: Requirements Development Cost, Dependencies and Importance (Priority) for Each Customer

Effort		r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}
		16	19	16	7	19	15	8	10	6	18	15	12	16	20	9	4	16	2	9	3
Priority Level	c_1	1	2	1	1	2	3	3	1	1	3	1	1	3	2	3	2	2	3	1	3
	c_2	3	2	1	2	1	2	1	2	2	1	2	3	3	2	1	3	2	3	3	1
	c_3	1	1	1	2	1	1	1	3	2	2	3	3	3	1	3	1	2	2	3	3
	c_4	3	2	2	1	3	1	3	2	3	2	3	2	1	3	2	3	2	1	3	3
	c_5	1	2	3	1	3	1	2	3	1	1	2	2	3	1	2	1	1	1	1	3
Effort		r_{21}	r_{22}	r_{23}	r_{24}	r_{25}	r_{26}	r_{27}	r_{28}	r_{29}	r_{30}	r_{31}	r_{32}	r_{33}	r_{34}	r_{35}	r_{36}	r_{37}	r_{38}	r_{39}	r_{40}
		2	10	4	2	7	15	8	20	9	11	5	1	17	6	2	16	8	12	18	5
Priority Level	c_1	2	1	1	1	3	3	3	3	1	2	2	3	2	1	2	2	1	3	3	2
	c_2	3	3	3	2	3	1	2	2	3	3	1	3	2	2	1	2	3	2	3	3
	c_3	2	1	2	3	2	3	3	1	3	3	3	2	1	2	2	1	1	3	1	2
	c_4	1	1	1	2	3	3	2	1	1	1	1	2	2	2	3	2	2	3	1	1
	c_5	1	1	3	3	3	2	2	3	2	3	1	1	3	3	2	2	1	1	2	1
Effort		r_{41}	r_{42}	r_{43}	r_{44}	r_{45}	r_{46}	r_{47}	r_{48}	r_{49}	r_{50}	r_{51}	r_{52}	r_{53}	r_{54}	r_{55}	r_{56}	r_{57}	r_{58}	r_{59}	r_{60}
		6	14	15	20	14	9	16	6	6	6	6	2	17	8	1	3	14	16	18	7
Priority Level	c_1	2	2	3	1	1	1	2	2	3	3	3	3	1	3	2	1	3	1	3	1
	c_2	3	3	1	1	3	2	2	2	1	3	3	3	1	2	2	3	3	2	1	1
	c_3	1	3	1	3	3	3	3	1	3	2	3	1	2	3	2	3	2	1	2	3

	c_4	3	1	1	3	1	2	1	1	3	2	2	1	3	2	1	3	3	1	2	3
	c_5	3	1	1	2	1	2	3	3	2	2	1	3	3	2	3	1	2	1	3	2
Effort		r_{61}	r_{62}	r_{63}	r_{64}	r_{65}	r_{66}	r_{67}	r_{68}	r_{69}	r_{70}	r_{71}	r_{72}	r_{73}	r_{74}	r_{75}	r_{76}	r_{77}	r_{78}	r_{79}	r_{80}
		10	7	16	19	17	15	11	8	20	1	5	8	3	15	4	20	10	20	3	20
Priority Level	c_1	2	2	3	3	1	3	1	3	2	3	1	3	2	3	1	1	2	3	3	1
	c_2	1	3	2	3	1	2	1	2	3	1	1	3	1	3	2	1	3	3	1	2
	c_3	1	1	2	3	3	1	3	3	3	1	3	1	3	1	1	2	3	3	1	2
	c_4	2	2	3	3	3	1	2	1	2	1	2	3	3	2	2	2	1	3	3	1
	c_5	2	2	1	2	1	3	2	1	2	1	2	2	3	2	1	3	2	3	1	3
Effort		r_{81}	r_{82}	r_{83}	r_{84}	r_{85}	r_{86}	r_{87}	r_{88}	r_{89}	r_{90}	r_{91}	r_{92}	r_{93}	r_{94}	r_{95}	r_{96}	r_{97}	r_{98}	r_{99}	r_{100}
		10	16	19	3	12	16	15	1	6	7	15	18	4	7	2	7	8	7	7	3
Priority Level	c_1	2	1	3	1	2	2	2	1	3	2	2	3	1	1	1	2	1	3	1	1
	c_2	1	2	1	2	2	1	3	2	2	2	3	2	2	3	2	2	1	3	1	1
	c_3	1	2	3	2	3	1	2	2	3	3	3	3	2	1	1	2	3	3	2	3
	c_4	3	1	2	2	2	1	1	1	3	1	1	3	3	1	2	1	2	3	1	3
	c_5	3	2	1	2	2	2	2	1	3	3	3	1	1	3	1	3	3	3	3	3

Interactions

Requirement	→	Dependencies		Requirement	→	Dependencies		Requirement	→	Dependencies
r2	→	*r*24		*r*17	→	*r*43		*r*40	→	*r*64
r3	→	*r*26, *r*27, *r*28, *r*29		*r*29	→	*r*49, *r*50, *r*51		*r*43	→	*r*65
r4	→	*r*5		*r*30	→	*r*52, *r*53		*r*46	→	*r*68
r6	→	*r*7		*r*31	→	*r*55		*r*47	→	*r*70
r7	→	*r*30		*r*32	→	*r*56, *r*57		*r*55	→	*r*79
r10	→	*r*32, *r*33		*r*33	→	*r*58		*r*56	→	*r*80
r14	→	*r*32, *r*34, *r*37, *r*38		*r*36	→	*r*61		*r*57	→	*r*80
r16	→	*r*39, *r*40		*r*39	→	r63		*r*62	→	*r*83, 84
r64	→	*r*87								

TABLE III: CUSTOMERS' RELATIVE IMPORTANCE

	c_1	c_2	c_3	c_4	c_5
Dataset 1	1	4	2	3	4
Dataset 2	1	5	3	3	1

TABLE IV: EXPERIMENTAL SETUP

BPSO parameter settings	w = 0.8, $c1$ = 1.5, $c2$ = 1.5,Vmax =4,Vmin =-4 Swarm size = 15, # of iterations = 15 (for dataset1) Swarm size = 30, # of iterations = 30 (for dataset2)	
Experiment ID	**BPSO version**	**Initialization**
1	Original BPSO (OBPSO)	Random
2		Greedy-Random
3	Improved BPSO (IBPSO)	Random
4		Greedy-Random

TABLE V: BEST SOLUTION AVERAGE FITNESS, $\mathbf{Satisf}(\mathbf{R_{NR}})$ AND $\mathbf{Cost}(\mathbf{R_{NR}})$ OVER DATASET1 AND DATASET2

	Budget limit	OBPSO						IBPSO					
		Random			Greedy-Random			Random			Greedy-Random		
		Fit.	S	Cost	Fit.	S	Cost	Fit	S	Cost	Fit.	S	Cost
Dataset 1	**30%**	3.90	162.91	7.75	3.60	179.08	7.83	4.74	138.25	6.83	11.76	101.75	2.83
	50%	2.68	193.33	11.08	2.15	247	13.5	2.82	221.91	9.91	6.17	184.25	6.75
	70%	2.32	236.41	12.75	2.17	280.25	14.58	2.49	207.91	11.41	3.48	215.66	11.16
Dataset 2	**30%**	1.52	846.83	239.66	1.52	892.25	241	1.62	715.33	217.66	4.67	367.92	77.16
	50%	1.16	1095.83	357.66	1.17	1074.5	351.08	1.25	1005.33	315	1.91	723.25	186.66
	70%	1.17	1021.91	350.16	1.19	994.83	337.83	1.27	977.41	308.33	1.47	920.16	256.41

TABLE VI: MEAN FITNESS AND STANDARD DEVIATION OF THE BEST GLOBAL SOLUTION OVER DATASET1 AND DATASET2

	Effort limit	OBPSO				IBPSO			
		Random		Greedy		Random		Greedy	
		Mean	SD	Mean	SD	Mean	SD	Mean	SD
Dataset 1	**30%**	3.90	1.5847	3.60	0.8615	4.74	6.9645	11.76	2.9889
	50%	2.68	0.8148	2.15	0.3691	2.82	6.3995	6.17	0.4919
	70%	2.32	0.6735	2.17	0.6327	2.49	3.0620	3.48	0.6064
Dataset 2	**30%**	1.52	0.0706	1.52	0.0592	1.61	1.8729	4.67	0.0435
	50%	1.16	0.0405	1.17	0.0400	1.25	0.3209	1.91	0.0340
	70%	1.16	0.0665	1.19	0.0854	1.27	0.1352	1.47	0.0697

TABLE VII: AVERAGE EXECUTION TIME (SEC) OF THE ALGORITHMS OVER DATASET1 AND DATASET2

	Budget limit	OBPSO		IBPSO	
		Random	Greedy	Random	Greedy
Dataset1	**30%**	2.32	2.21	0.46	0.56
	50%	2.38	2.29	0.68	0.52
	70%	2.43	2.40	0.79	0.62
Dataset2	**30%**	278.74	279.53	76.99	50.26
	50%	292.69	297.61	130.13	58.93
	70%	307.35	306.70	140.05	61.98

REFERENCES

[1] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*, Prentice Hall, 2001.

[2] R. Fuchshuber and M. de O. Barros, "Improving heuristics for the next release problem through landscape visualization," *Lecture Notes in Computer Science*, Springer, vol. 8636, pp. 222-227, 2014.

[3] A. J. Bagnall, V. J. Rayward-Smith, and I. Whittley, "The next release problem," *Information and Software Technology*, vol. 43, no. 14, pp. 883–890, 2001.

[4] B. Glauber, R. Arthur, B. Andre, and S. Leila, "Investigating bioinspired strategies to solve large scale next release problem," in *Proc. 18th Ibero American Conference on Software Engineering*, Lima, Peru, April 22-24, 2015.

[5] J. M. Chaves-González and M. A. Pérez-Toledano, "Differential evolution with Pareto tournament for the multi-objective next release problem," *Applied Mathematics and Computation*, vol. 252, pp. 1–13, 2015.

[6] R. Eberhart and J. Kennedy, "A new optimizer using particles swarm theory," in *Proc. Sixth International Symposium on Micro Machine and Human Science*, Nagoya, Japan, 1995, pp. 39-43.

[7] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE International Conference on Neural Networks*, Perth, Australia, 1995, pp. 1942-1948.

[8] J. Kennedy and R. Eberhart, *Swarm Intelligence*, Morgan Kaufmann Publishers, Inc., San Francisco, CA, 2001.

[9] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*, Wiley, 2005.

[10] M. Dorigo and T. Stützle, *Ant Colony Optimization*, MIT Press, Cambridge, 2004.

[11] J. del Sagrado, I. M. del Águila, and F. J. Orellana, "Multi-objective ant colony optimization for requirements selection," *Journal of Empirical Software Engineering*, vol. 20, issue 3, pp. 577–610, June 2015.

[12] J. T. Souza, C. L. B. Maia, T. N. Ferreira, R. A. F. do Carmo, and M. M. A. Brasil, "An ant colony optimization approach to the software release planning with dependent requirements," in *Proc. the 3th Int. Symposium on Search Based Software Engineering*, 2011, pp. 142–157.

[13] D. Goldberg, *Genetic Algorithms in Search Optimizationand Machine Learning*, Addison-Wesley Publishing, 1989.

[14] A. Hamdy, "Genetic fuzzy system for enhancing software estimation models," *International Journal of Modeling and Optimization*, vol. 4, no. 3, June 2014

[15] A. A. Araújo, M. Paixao, I. Yeltsio, A. Dantas, and J. Souza, "An Architecture based on interactive optimization and machine learning applied to the next release problem," *Automated Software Engineering, Springer*, vol. 24, issue 3, pp. 623-671, September 2017.

[16] A. K. Mojtaba, T. Mohammed, and A. S. Mahdi, "A novel binary particle swarm optimization," in *Proc. 2007 Mediterranean Conference on Control and Automation*, Athens, 2007.

[17] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP Completeness*, Freeman, New York, 1990.

[18] K. Price and R. Storn, "Differential evolution – a simple evolution strategy for fast optimization," *Dr. Dobb's J.*, vol. 22, no. 4, pp. 18–24,1997.

[19] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell *et al*., "An industrial survey of requirements interdependencies in software product release planning," in *Proc. Fifth IEEE International Symposium on Requirements Engineering*, 2001, pp. 84–91.

[20] D. Greer and G. Ruhe, "Software release planning: An evolutionary and iterative approach," *Information and Software Technology*, vol. 46, no. 4, pp. 243–253, 2004.

[21] E. Simmons, "Requirements triage: What can we learn from a ''medical'' approach?" *IEEE Software*, vol. 21, no. 4, pp. 86–88, 2004.

A. Hamdy is an associate professor in the Faculty of Informatics and Computer Science, the British University, Egypt. She earned her B.Sc., M.Sc. and Ph.D degrees in electronics and electrical communications from the Faculty of Engineering, Cairo University in 1992, 1998, and 2003 respectively.

Her research focuses on software engineering and machine learning.