Application of the Bee Swarm Optimization BSO to the Knapsack Problem

Marco Aurelio Sotelo-Figueroa, Rosario Baltazar, Juan Martín Carpio

Abstract:

Swarm Intelligence is the part of Artificial Intelligence based on study of actions of individuals in various decentralized systems. The optimization algorithms which are inspired from intelligent behavior of honey bees are among the most recently introduced population based techniques. In this paper, a novel hybrid algorithm based in Bees Algorithm and Particle Swarm Optimization is applied to the Knapsack Problem. The Bee Algorithm is a new population-based search algorithm inspired by the natural foraging behavior of honey bees, it performs a kind of exploitative neighborhood search combined with random explorative search to scan the solution, but the results obtained with this algorithm in the Knapsack Problem are not very good. Although the combination of BA and PSO is given by BSO, Bee Swarm Optimization, this algorithm uses the velocity vector and the collective memories of PSO and the search based on the BA and the results are much better. We use the Greedy Algorithm, which it's an approximate algorithm, to compare the results from these metaheuristics and thus be able to tell which is which gives better results.

Keywords: *swarm optimization, PSO, BA, BSO, Knapsack Problem.*

1. Introduction

Evolutionary and meta-heuristic algorithms have been extensively used as search and optimization tools during this decade in several domains from science and engineering to the video game industry, and others.

Many demanding applications that involve the solution of optimization problems of high complexity, a lot of these belonging to a special class of problems called NP-hard have been solved by various methods [8]. Metaheuristic algorithms are now considered among the best tools must to find good solutions with a reasonable investment of resources.

As described by Eberhart and Kennedy [4] Particle Swarm Optimization or PSO algorithm is part of the Swarm Intelligence and is a metaheuristics that use the social-psychological metaphor; a population of individuals, referred to as particles, adapts by returning stochastically toward previously successful regions. The PSO simulate a society where each individual contributes with his knowledge to the society. These metaheuristics have proved their ability to deal with very complicated optimization and search problems.

The behavior of a single ant, bee, termite or wasp often is too simple, but their collective and social behavior is of paramount significance. The collective and social behavior of living creatures motivated researchers to undertake the study of today what is known as Swarm Intelligence [5]. Two fundamental concepts, self-organization and division of labor, are necessary and sufficient properties to obtain swarm intelligent behavior.

The Bee Algorithm or BA [10] is also part of the Swarm Intelligence and this mimics the honey bees and who this search their food foraging. This algorithm is based on a random search on the neighborhood for combinatorial and functional optimization.

The Knapsack Problem is a classical combinatorial problem [3], [15] can be described as follows: "Imagine taking a trip to which you can only carry a backpack that, logically, has a limited capacity. Given a set of items, each with a weight and a value, determine the number of each item to include in a bag so that the total weight is less than a given limit and the total value is as large as possible", this problem can be considerate as NP-easy problem but some studies show that the Knapsack Problem is an NP-Hard problem [2].

Actually the Knapsack Problem can be modeled by different ways [16] for example multi-dimensional and multiple Knapsack problem [1], [14], [17], quadratic Knapsack problem [6], [13].

In the present paper we introduce the Bee Swarm Optimization or BSO. This algorithm is a hybrid metaheuristic between the PSO and the BA. The BSO use the better characteristics from both algorithms, the Social Metaphor from the PSO and the random search in the neighborhood from the BA, and give us a better result. The experiments were made on seven types of instances from uncorrelated, to uncorrelated similar weight. All these instances probe the algorithm varying the parameters of the profits and the weight. The algorithms were probed with different methods for generating the initial populations [9] like random solutions, uniformly distributed solutions and greedy base solutions. Another important characteristic is the fitness's Confidence Interval for saying what metaheuristic with who initial population is better. It was necessary to have a comparison point for the metaheuristics and was use the Greedy Algorithm [3], this is a deterministic algorithm who gives an approximate result for the Knapsack Problem.

2. Knapsack Problem

The Knapsack problem [15] is the typical combinatorial problem that has been studied since many years ago and was proved that it is a NP-Hard problem [12]. The basic problem is the 0-1 Knapsack Problem or Binary Knapsack Problem and it have a search space of $2^n - 1$

possible solutions.

The Knapsack Problem can be described as follows: "there are *n* objects, each of this objects have a profit and weight, and needs to select those whose sum of their benefits is maximized subject to the sum of the weight of the same objects should not exceed an amount determined". It can be formulated mathematically by numbering each of its objects or items from 1 to *n* and introducing it to a vector of binary variables $x_j = 1,2,3,...,n$, where each variable represented here will take the value 1 or 0 depending on whether it is selected or not.

The solution to the Knapsack Problem is select a subset of objects from the binary vector *x*, solution vector, that satisfies the constraint on the equation (2) and the same time maximize the objective function on the equation (1).

$$z = \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\sum_{j=1}^{n} w_j x_j \le c \qquad x_j = \begin{cases} \text{If the } j \text{ object is selected} \\ 0 \quad \text{otherwise} \end{cases} \tag{2}$$

where:

- z represents the profit.
- j represents the j-th object.
- x_j indicates whether the j object is part of the solution.
- p_j is the j-th object profit.
- w_j is the j-th object weight.
- c is the volume or capacity of the knapsack.

2.1. Types of Knapsack Problem Instances

The Knapsack Problem is affected by the relationship between the profit and the weight of the objects; these types of instances are the following:

Uncorrelated: the profits and Weight are distributed uniformly between one and a maximum *V* number.

$$p_j \in [1,V];\ w_j \in [1,V] \tag{3}$$

Weakly correlated: the Weight is distributed uniformly between one and a maximum *V* number and the profits are distributed uniformly around the weight and an *R* ratio.

$$w_j \in [1,V];\ p_j \in [w_j - R,\ w_j + R] \tag{4}$$

Strongly correlated: the Weight is uniformly distributed between one and a maximum *V* number; the profits are the Weight plus one *K* constant.

$$w_j \in [1,V];\ p_j = w_j + K \tag{5}$$

Inverse strongly correlated: the profits are distributed uniformly between one and a maximum *V* number and the Weight is the profits plus one *K* constant.

$$p_j \in [1,V];\ w_j = p_j + K \tag{6}$$

Almost strongly correlated: the Weight is distributed uniformly between one and a maximum *V* number and the profits are the Weight plus one random number between one and a maximum *S* number.

$$S \in [1,U];\ w_j \in [1,V];\ p_j = w_j + S \tag{7}$$

Subset-sum: the profits and Weight have the same value and are distributed uniformly between one and a maximum *V* number.

$$w_j \in [1,V];\ p_j = w_j \tag{8}$$

Uncorrelated similar Weight: the profits are distributed uniformly between one and a maximum *V1* number and the Weight is distributed uniformly between one and a maximum *V2* number plus a *K* constant.

$$p_j \in [1,V_1];\ w_j \in [1,V_2] + K \tag{9}$$

3. The Greedy Algorithm

This algorithm gives a intuitive approach considering the profit and weight of each item, it is know as efficiency and it's based on the Equation (10) and the main is to try to put the items with highest efficiency into the Knapsack. It is necessary sort all the items based on the efficiency, using the Equation (11), before to apply the Greedy to the problem.

$$e_j = \frac{p_j}{w_j} \tag{10}$$

$$\frac{p_0}{w_0} \ge \frac{p_1}{w_1} \ge \ \ldots \ \ge \frac{p_n}{w_n} \tag{11}$$

4. Bee Algorithm (BA)

The Bee Algorithm [10] or BA is a bio-inspired metaheuristic behavior of honey bees and how they searching for plant to obtain the necessary pollen for honey production.

A colony of bees search in a large territory looking for new sources of food and begins to thrive on discovering new food sources. When these sources have much pollen are visited by large numbers of bees and when the pollen decreases the number of bees collected from these sources decreases too.

When season for recollecting pollen start, the colony sent so many bees, which are called scouts bees to reconnoiter randomly the territory to inform at the colony where are the best food sources. Once the harvesting season starts the colony maintains a certain percentage of their scout bees in order to detect new and better sources of food. When scout bees have returned to the colony and found a better food source than the currently is using the colony, makes the Dance by means of which transmits the exact position of the source food and then the colony began to send more bees to the food source.

Require: *ps* population size, *n* neighbourhood.
1: Initialise population with random solutions.
2: Evaluate fitness of the population.
3: **while** stopping criterion not met **do**
4: Select the *n* sites for neighbourhood search.
5: Recruit bees for *n* selected sites and evaluate fitnesses.
6: Select the fittest bee from each site.
7: Assign remaining bees to search randomly and evaluate their fitnesses.
8: **end while**

Fig. 1. The BA Algorithm applied to the 0-1 Knapsack Problem.

5. Particle Swarm Optimization (PSO)

The Particle Swarm Optimization [4], [7] or PSO is a Bio-inspired metaheuristic in flocks of birds or schools of fish. It was developed by J. Kennedy and R. Eberthart based on a concept called social metaphor, this metaheuristics simulates a society where all individuals contribute their knowledge to obtain a better solution, there are three factors that influence for change in status or behavior of an individual:

- The Knowledge of the environment or adaptation which speaks of the importance given to the experience of the individual.
- His Experience or local memory is the importance given to the best result found by the individual.
- The Experience of their neighbors or Global memory referred to how important it is the best result I have obtained their neighbors or other individuals.

In this metaheuristic each individual is called particle and moves through a multidimensional space that represents the social space or search space depends on the dimension of space which depends on the variables used to represent the problem.

For the update of each particle using something called velocity vector which tells them how fast it will move the particle in each of the dimensions, the method for updating the speed of PSO is given by equation (12), and it is updating by the equation (13).

$$v_i = \varphi_0 * v_i + \varphi_1(x_i - B_{Global}) + \varphi_2(x_i - B_{Local}) \quad (12)$$

$$x_i = x_i + v_i \quad (13)$$

where:

- v_i is the velocity of the i-th particle
- φ_0 is adjustment factor to the environment.
- φ_1 is the memory coefficient in the neighborhood.
- φ_2 is the coefficient memory.
- x_i is the position of the i-th particle.
- B_{Global} is the best position found so far by all particles.
- B_{Local} is the best position found by the i-th particle

Require: φ_0 adaptation to environment coefficient, φ_1 neighborhood memory coefficient, φ_2 memory coefficient, n particles.
1: Start the swarm particles.
2: Start the velocity vector for each particle in the swarm.
3: **while** stopping criterion not met **do**
4: **for** $i = 1$ to n **do**
5: If the i-particle's fitness is better than the local best then replace the local best with the i-particle.
6: If the i-particle's fitness is better than the global best then replace the global best with the i-particle.
7: Update the velocity vector.
8: Update the particle's position with the velocity vector.
9: **end for**
10: **end while**

Fig. 2. The PSO Algorithm applied to the 0-1 Knapsack Problem.

6. Bee Swarm Optimization (BSO)

The Bee Swarm Optimization or BSO is a hybrid metaheuristic population between the PSO and the BA. The main idea of BSO is based on taking the best of each metaheuristics to obtain better results than they would obtain.

The BSO use the velocity vector and the way to updating it, equation (12), and applies the social metaphor to get better results from the PSO and use the exploration and a search radius from the BA to indicate which is where they look for a better result.

The first thing that the BSO do is update the position of the particles through the velocity vector and then select a specified number of particles, in this case it is proposed to select the best as being the new food supply that the scout bees discovered, and conducted a search of the area enclosed by the radius search and if there are a better solution in this area than the same particle around which are looking for then the particle is replaced with the position of the best solution found in the area.

For this metaheuristic to work properly you need a way to determine what the next and earlier particle position, if we cannot determine this then it is impossible to apply this metaheuristic because you would not know what the next and previous particle to search in that area.

We defined the next and before solutions like binary operations, adding and subtracting one number to the solution vector. For example if we have a Knapsack Problem with 5 elements the solution vector will have 5 spaces and in each space can be 0 or 1, we can see the example of the next and before solution vector in the Figure 3.

Fig. 3. Example of the previous and next solution vector.

The algorithm of the BSO applied to the Knapsack Problem implemented in the present paper is the following.

Require: φ_0 adaptation to environment coefficient, φ_1 neighborhood memory coefficient, φ_2 memory coefficient, n swarm size, sb scout bees, r search radio.
1: Start the swarm particles.
2: Start the velocity vector for each particle in the swarm.
3: **while** stopping criterion not met **do**
4: **for** $i = 1$ to n **do**
5: If the i-particle's fitness is better than the local best then replace the local best with the i-particle.
6: If the i-particle's fitness is better than the global best then replace the global best with the i-particle.
7: Update the velocity vector.
8: Update the particle's position with the velocity vector.
9: **Choose the best** sb **particles**
10: **for all** best sb particle **do**
11: **Search if there are some better particle in the search radio and if exist it remplace the particle with the best particle in the search radio**
12: **end for**
13: **end for**
14: **end while**

Fig. 4. The BSO Algorithm applied to the 0-1 Knapsack Problem.

7. Experiments

To Test the BSO was used the Generator of Knapsack Test Instances [11]; it requires the number of elements and the coefficients range to generate a test instance. Were generate the seven types of test instances described, and was use the same parameters for each metaheuristic to find which of this are better for the Knapsack Problem. Each metaheuristic was run 100 times for obtaining their average and standard deviation, and was use that data for calculating the Confidence Interval at the 97% of confidence.

We use 3 different Initial Population, this because is importance [9] of start with a good solution the metaheu-

ristics, the initial population use were the followings:

- *Greedy:* we use the Greedy Algorithm.
- *Uniformly Distributed:* we use the number generator uniformly distributed.
- *Random:* we use the random number generator of Java.

We determinate the size of the Knapsack obtaining the average size of the instance and considerate it average like the size of the elements and multiply it by 20 for considerate 20 elements.

Table 1. Parameters used in the Generator of Knapsack Test Instances.

Parameters	Values
Elements	50
Coefficients Range	[0,100]

Table 2. Parameters used in the BA.

Parameters	Values
Elements	50
Iterations	100
Elements for Neighborhood	10
Search Radio	3

Table 3. Parameters used in the PSO.

Parameters	Values
Elements	50
Iterations	100
φ_0	1
φ_1	0.5
φ_3	0.8

Table 4. Parameters used in the BSO.

Parameters	Values
Elements	50
Iterations	100
Elements for Neighborhood	10
Search Radio	3
φ_0	1
φ_1	0.5
φ_3	0.8

8. Results

In this work we show the results obtained by testing each Type Instances with the different metaheuristics and the three different kinds of initial population, we show the Average profit, the Best profit and Worst profit, their fitness's Standard Deviation for each metaheuristic and their fitness's Confidence Interval. We also show the metaheuristics behavior through their graphic. In the graphics the red squares represent the results obtained by the Greedy Algorithm, the blue dots represent the Initial Population based on the Greedy, the green triangles represent the Initial Population Uniformly Distributed and the yellow diamonds represent the Initial Population Random.

8.1. Uncorrelated

To test the Uncorrelated instance the Knapsack size was defined at 1037. We can see the results of each metaheuristic with different initial populations in the Table 5 and their behavior in the Figures 5, 6 and 7. The result provide by the Greedy Algorithm was 1751 and as can be seen the BA gives the worst results, always equal or under the Greedy's result, while the PSO and BSO gives better results but the Standard Deviation from the BSO is smaller than the PSO and their Average from the BSO is better.

Table 5. Results obtained from the Uncorrelated Knapsack Problem.

Metaheuristic	Initial Population	Average	Best	Worst	σ	Confidence Interval
PSO	Greedy	1753.79	1758	1751	2.29	[1753.08, 1754.49]
	Uniformly Distributed	1751.39	1758	1725	6.51	[1749.39, 1753.38]
	Random	1750.96	1758	1716	6.3	[1749.02, 1752.89]
BA	Greedy	1751	1751	1751	0	[1751.00, 1751.00]
	Uniformly Distributed	1397.43	1522	1291	51.08	[1381.75, 1413.10]
	Random	1440.25	1550	1296	50.66	[1424.70, 1455.79]
BSO	Greedy	1753.52	1758	1751	2.1	[1752.87, 1754.16]
	Uniformly Distributed	1753.07	1758	1733	4.12	[1751.80, 1754.33]
	Random	1752.65	1758	1733	4.74	[1751.19, 1754.10]

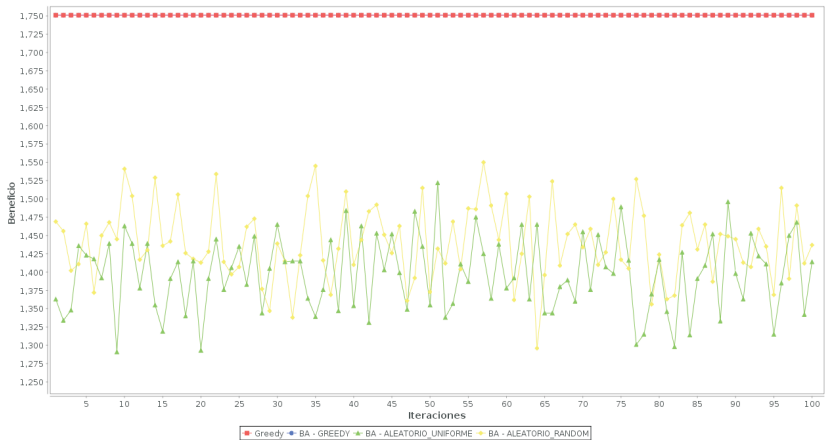

Fig. 5. Graphic behaviors of BA metaheuristics apply to the Uncorrelated problem.

Fig. 6. Graphic behaviors of PSO metaheuristics apply to the Uncorrelated problem.

Fig. 7. Graphic behaviors of BSO metaheuristics apply to the Uncorrelated problem.

8.2. Weakly correlated

To test the Weakly Correlated instance, the Knapsack size was defined at 914. We can see the results of each metaheuristic with different initial populations in the Table 6 and their behavior in the Figures 8, 9 and 10. The result provide by the Greedy Algorithm was 1044 and as we can see the BA gives the worst results, only with the initial population based on the Greedy improve the Greedy's result, the BSO and PSO gives better results but the BSO improve the PSO's results and gives a smaller Standard Deviation.

Table 6. Results obtained from the Weakly Correlated Knapsack Problem.

Metaheuristic	Initial Population	Average	Best	Worst	σ	Confidence Interval
PSO	Greedy	1049.83	1053	1046	2.11	[1049.18, 1050.48]
	Uniformly Distributed	1040.34	1053	1027	5.52	[1038.64, 1042.03]
	Random	1040.03	1053	1023	6.14	[1038.14, 1041.91]
BA	Greedy	1046	1046	1046	0	[1046, 1046]
	Uniformly Distributed	965.14	987	941	9.23	[962.30, 967.97]
	Random	973.1	1006	957	9.46	[970.19,976.00]
BSO	Greedy	1050.87	1053	1046	1.77	[1050.32, 1051.41]
	Uniformly Distributed	1045.18	1053	1038	3.71	[1044.04, 1046.31]
	Random	1045.19	1053	1033	4.14	[1043.91, 1046.46]

Fig. 8. Graphic behaviors of BA metaheuristics apply to the weakly correlated problem.

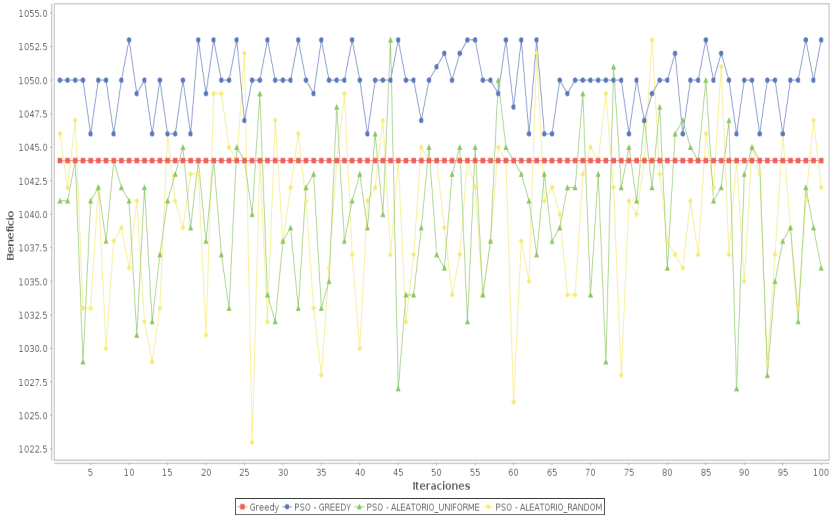

Fig. 9. Graphic behaviors of PSO metaheuristics apply to the weakly correlated problem.

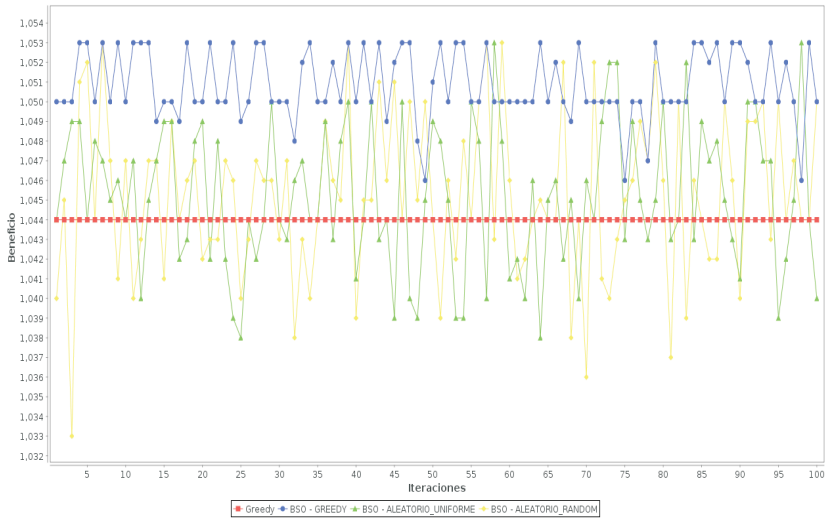

Fig. 10. Graphic behaviors of BSO metaheuristics apply to the weakly correlated problem.

8.3. Strongly correlated

To test the Strongly Correlated instance, the Knapsack size was defined at 1026. We can see the results of each metaheuristic with different initial populations in the Table 7 and their behavior on the Figures 11, 12 and 13. The result provide by the Greedy Algorithm was1332 and as we can see the BA gives the worst results, only with the initial population based on the Greedy improve the Greedy's result, the BSO and PSO gives better results but the BSO improve the PSO's results and gives a smaller Standard Deviation.

Table 7. Results obtained from the Strongly Correlated Knapsack Problem.

Metaheuristic	Initial Population	Average	Best	Worst	σ	Confidence Interval
PSO	Greedy	1345.82	1346	1342	0.67	[1345.61, 1346.02]
	Uniformly Distributed	1328.83	1343	1314	6.28	[1326.90, 1330.75]
	Random	1331.78	1336	1317	4.69	[1330.33, 1333.22]
BA	Greedy	1332	1332	1332	0	[1332, 1332]
	Uniformly Distributed	1265.39	1295	1245	10.42	[1262.19, 1268.58]
	Random	1276.78	1296	1260	7.99	[1274.32, 1279.23]
BSO	Greedy	1345.8	1346	1342	0.8	[1345.55, 1346.04]
	Uniformly Distributed	1337.16	1346	1326	3.46	[1336.09, 1338.22]
	Random	1337.24	1346	1331	3.47	[1336.17, 1338.30]

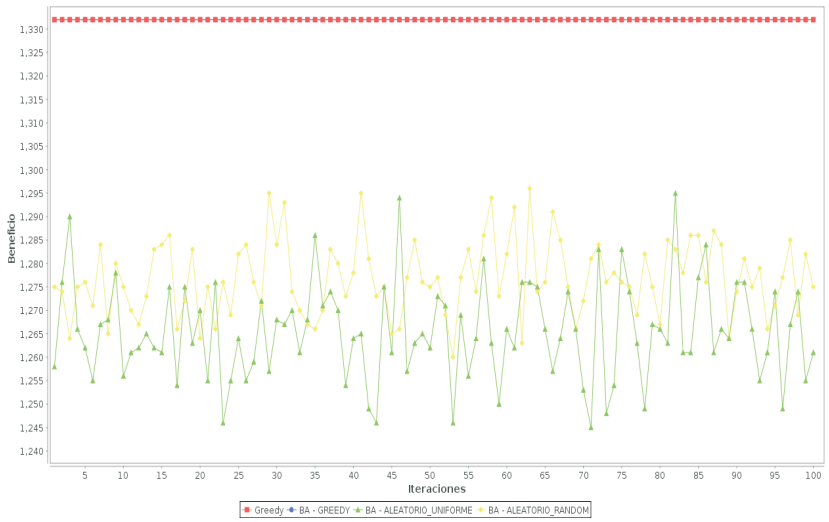

Fig. 11. Graphic behaviors of BA metaheuristics apply to the strongly correlated problem.

Fig. 12. Graphic behaviors of PSO metaheuristics apply to the strongly correlated problem.

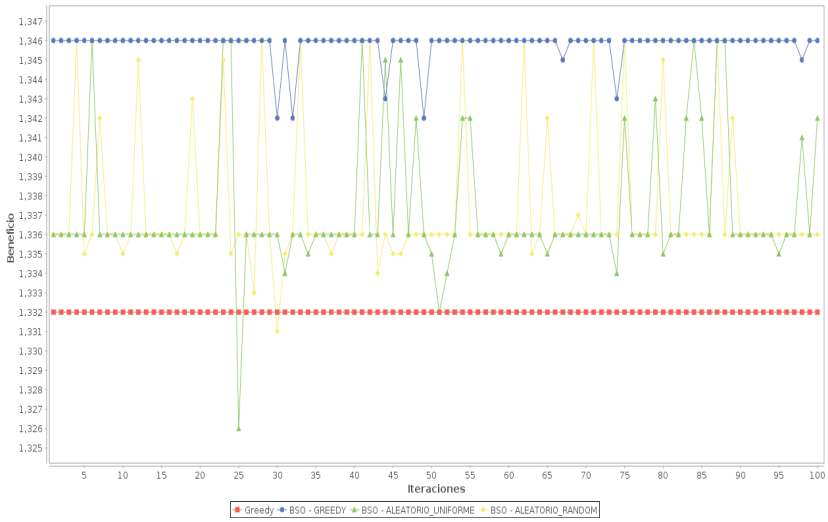

Fig. 13. Graphic behaviors of BSO metaheuristics apply to the strongly correlated problem.

8.4. Inverse strongly correlated

To test the Inverse Strongly Correlated instance was define the Knapsack size at 1182. We can see the results of each metaheuristic with different initial populations in the Table 8 and their behavior in the Figures 14, 15 and 16. The result provide by the Greedy Algorithm was 1051 and we can see the BA gives the worst results, only with the initial population based on the Greedy improve the Greedy's result, the BSO and PSO gives better results but the BSO improve the PSO's results and gives a smaller Standard Deviation.

Table 8. Results obtained from the Inverse Strongly Correlated Knapsack Problem.

Metaheuristic	Initial Population	Average	Best	Worst	σ	Confidence Interval
PSO	Greedy	1051.34	1052	1051	0.47	[1051.19, 1051.48]
	Uniformly Distributed	1036.42	1052	1017	5.86	[1034.62, 1038.21]
	Random	1034.65	1049	1021	5.82	[1032.86, 1036.43]
BA	Greedy	1051	1051	1051	0	[1051, 1051]
	Uniformly Distributed	1011.7	1036	998	7.06	[1009.53, 1013.86]
	Random	1003.55	1021	990	5.84	[1001.75, 1005.34]
BSO	Greedy	1051.87	1052	1051	0.33	[1051.76, 1051.97]
	Uniformly Distributed	1044.81	1052	1038	3.99	[1043.58, 1046.03]
	Random	1044.01	1052	1033	3.79	[1042.84, 1045.17]

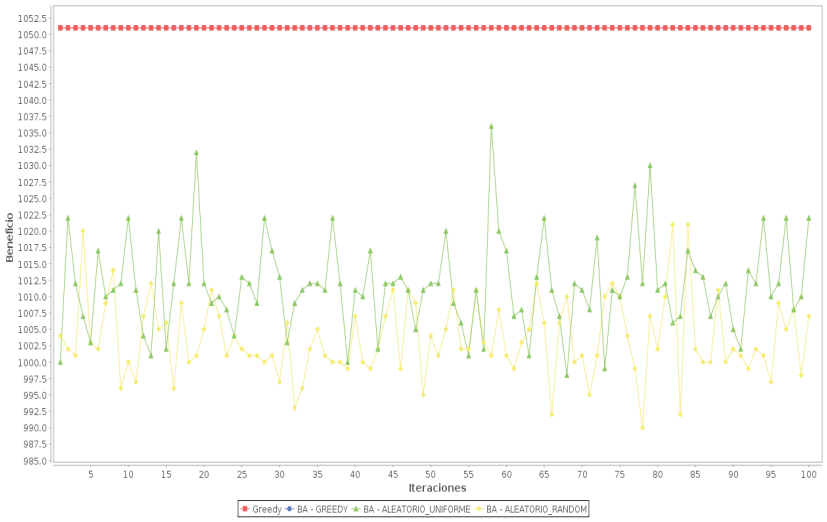

Fig. 14. Graphic behaviors of BA metaheuristics apply to the inverse strongly correlated problem.

Fig. 15. Graphic behaviors of PSO metaheuristics apply to the inverse strongly correlated problem.

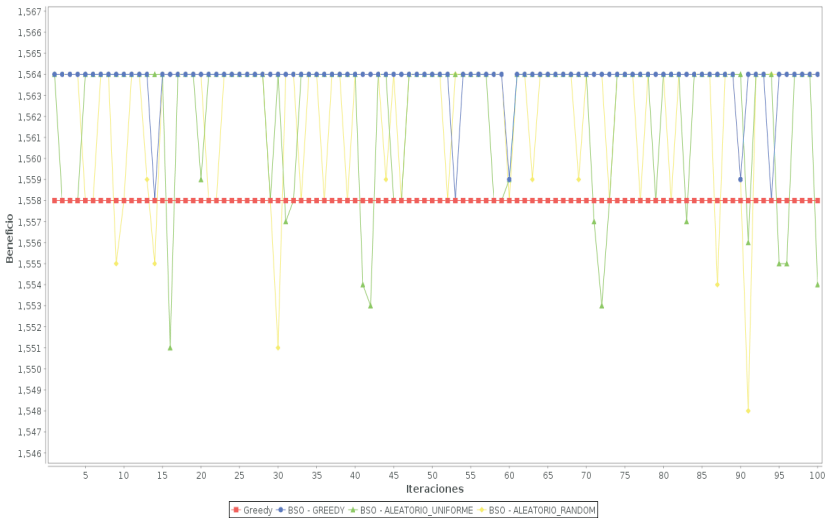

Fig. 16. Graphic behaviors of BSO metaheuristics apply to the inverse strongly correlated problem.

8.5. Almost strongly correlated

To test the Almost Strongly Correlated instance was define the Knapsack size at 124. We can see the results of each metaheuristic with different initial populations in the Table 9 and their behavior in the Figures 17, 18 and 19. The result provide by the Greedy Algorithm was 1524 as we can see the BA gives the worst results, only with the initial population based on the Greedy improve the Greedy's result, the BSO and PSO gives better results but the average from the PSO is better than the BSO while the Standard Deviation from the BSO is smaller than the PSO.

Table 9. Results obtained from the Almost Strongly Correlated Knapsack Problem.

Metaheuristic	Initial Population	Average	Best	Worst	σ	Confidence Interval
PSO	Greedy	1554.62	1555	1552	0.78	[1554.37, 1554.86]
	Uniformly Distributed	1538.19	1555	1521	6.39	[1536.22, 1540.15]
	Random	1541.33	1555	1529	5.54	[1539.62, 1543.03]
BA	Greedy	1552	1552	1552	0	[1552, 1552]
	Uniformly Distributed	1491.92	1508	1474	7.38	[1489.65, 1494.18]
	Random	1501.51	1517	1488	6.73	[1499.44, 1503.57]
BSO	Greedy	1554.97	1555	1554	0.17	[1554.91, 1555.02]
	Uniformly Distributed	1550.06	1555	1542	5.01	[1548.52, 1551.59]
	Random	1550.84	1555	1543	4.83	[1549.35, 1552.32]

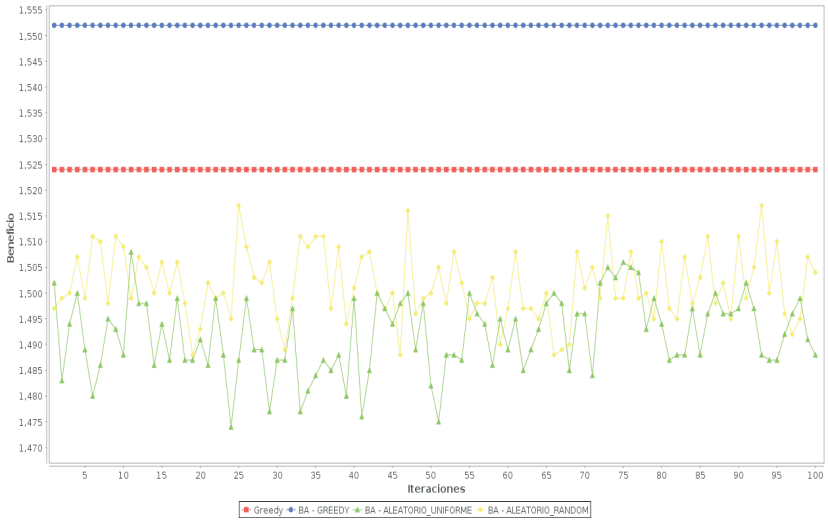

Fig. 17. Graphic behaviors of BA metaheuristics apply to the almost correlated problem.

Fig. 18. Graphic behaviors of PSO metaheuristics apply to the almost correlated problem.

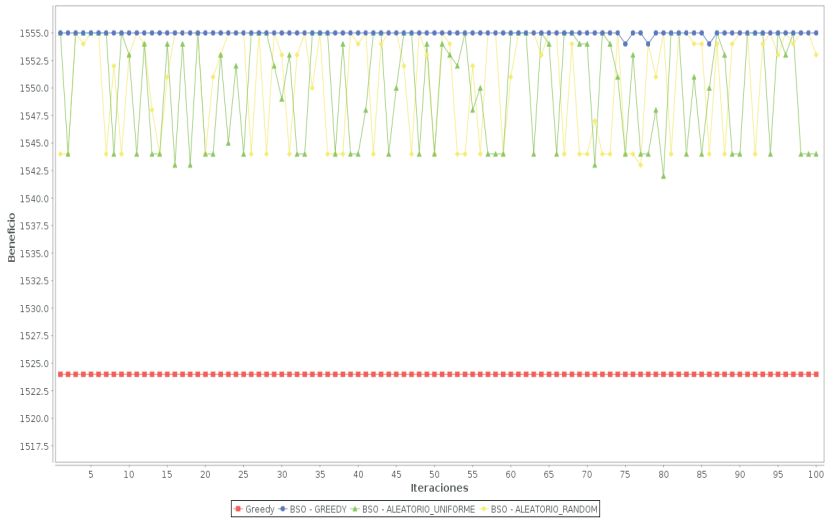

Fig. 19. Graphic behaviors of BSO metaheuristics apply to the almost correlated problem.

8.6. Subset-sum

To test the Subset-num instance was define the Knapsack size at 960. We can see the results of each metaheuristic with different initial populations in the Table 10 and their behavior in the Figures 20, 21 and 22. The result provide by the Greedy Algorithm was 949 and in this instance the PSO and the BSO gives the same results with Standard Deviation zero, better that the Greedy's result, and the BA approaching to this results.

Table 10. Results obtained from the Subset-sum Knapsack Problem.

Metaheuristic	Initial Population	Average	Best	Worst	σ	Confidence Interval
PSO	Greedy	960	960	960	0	[960, 960]
	Uniformly Distributed	960	960	960	0	[960, 960]
	Random	960	960	960	0	[960, 960]
BA	Greedy	959.82	960	959	0.38	[959.70, 959.93]
	Uniformly Distributed	959.85	960	958	0.38	[959.73, 959.96]
	Random	959.99	960	959	0.1	[959.95, 960.02]
BSO	Greedy	960	960	960	0	[960, 960]
	Uniformly Distributed	960	960	960	0	[960, 960]
	Random	960	960	960	0	[960, 960]

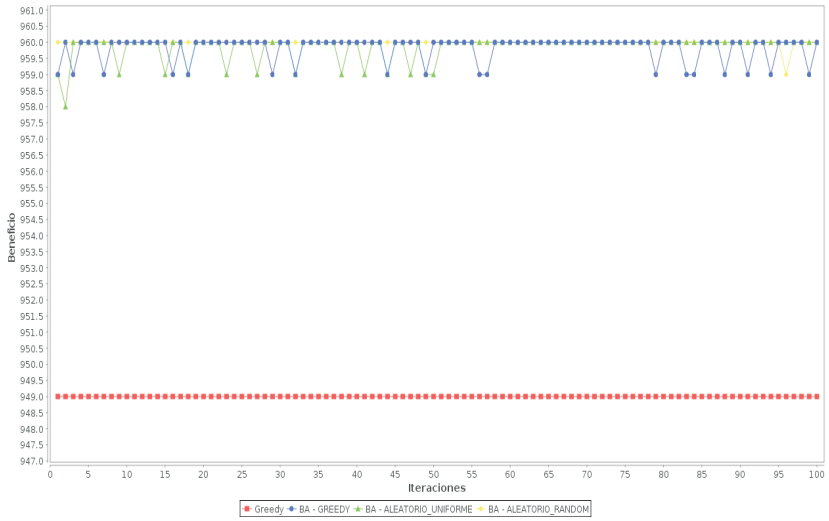

Fig. 20. Graphic behaviors of BA metaheuristics apply to the subset-num problem.

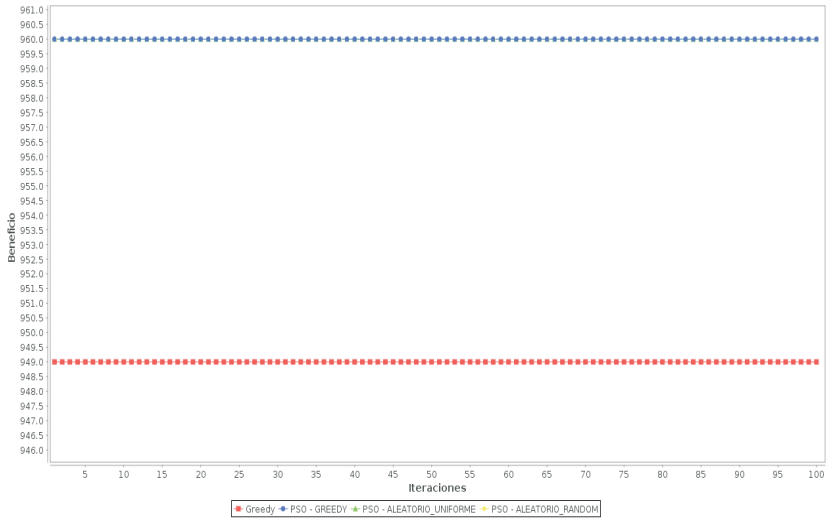

Fig. 21. Graphic behaviors of PSO metaheuristics apply to the subset-num problem.

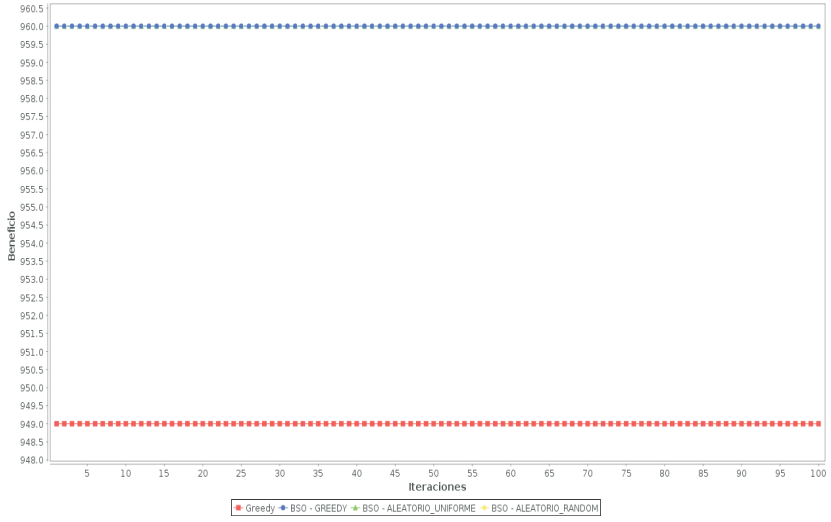

Fig. 22. Graphic behaviors of BSO metaheuristics apply to the subset-num problem.

8.7. Uncorrelated Similar Weight

To test the Uncorrelated Similar Weight instance was define the Knapsack size at 200118. We can see the results of each metaheuristic with different initial populations in the Table 11 and their behavior in the Figures 23, 24 and 25. The result provide by the Greedy Algorithm was 1558 as we can see the BA gives the worst results, only with the initial population based on the Greedy equals the Greedy's result, the BSO and PSO gives better results but the average from the BSO is better than the PSO while the Standard Deviation from the PSO is smaller than the BSO.

Table 11. Results obtained from the Uncorrelated Similar Weight Knapsack Problem.

Metaheuristic	Initial Population	Average	Best	Worst	σ	Confidence Interval
PSO	Greedy	1563.79	1564	1558	1.03	[1563.47, 1564.10]
	Uniformly Distributed	1562.18	1564	1548	3.15	[1561.21, 1563.14]
	Random	1561.38	1564	1551	3.63	[1560.26, 1562.49]
BA	Greedy	1558	1558	1558	0	[1558, 1558]
	Uniformly Distributed	1303.6	1406	1243	35.45	[1292.71, 1314.48]
	Random	1312.81	1404	1242	35.8	[1301.82, 1323.79]
BSO	Greedy	1563.72	1564	1558	1.23	[1563.34, 1564.09]
	Uniformly Distributed	1562.22	1564	1551	3.35	[1561.19, 1563.24]
	Random	1562.28	1564	1548	3.26	[1561.27, 1563.28]

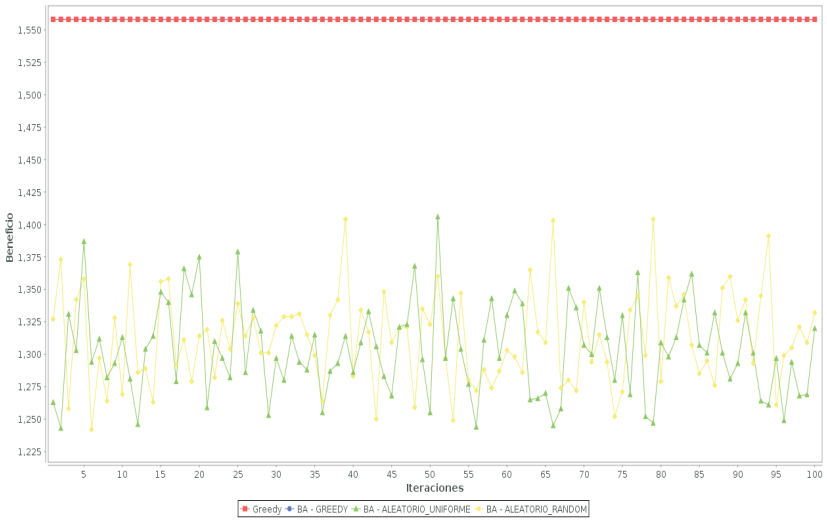

Fig. 23. Graphic behaviors of BA metaheuristics apply to the uncorrelated similar weight problem.

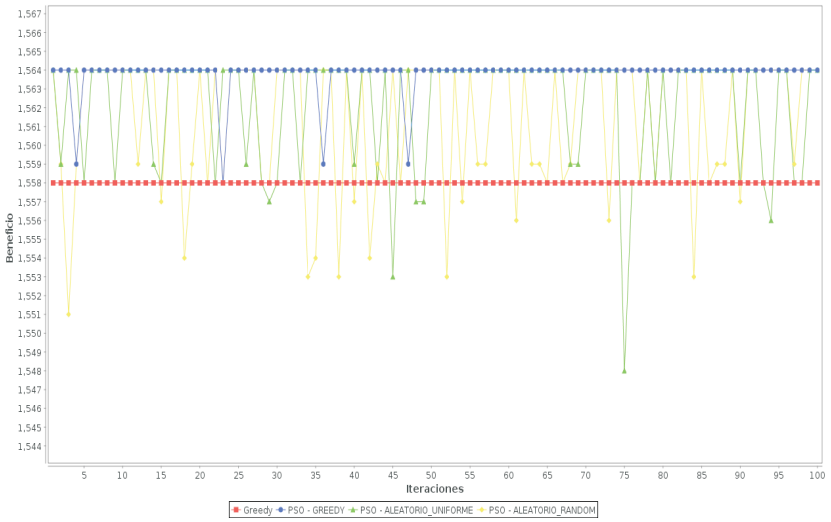

Fig. 24. Graphic behaviors of PSO metaheuristics apply to the uncorrelated similar weight problem.

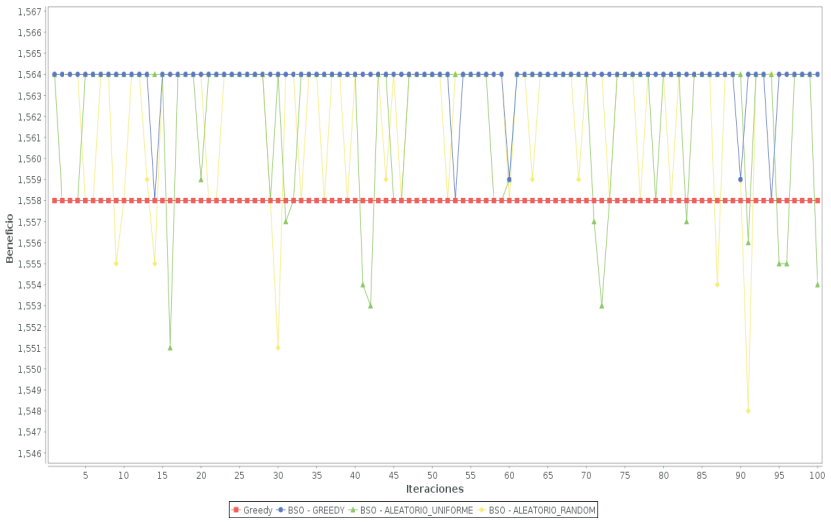

Fig. 25. Graphic behaviors of BSO metaheuristics apply to the uncorrelated similar weight problem.

9. Conclusions

There are many metaheuristics to solve the Knapsack Problem; in this work we introduce the Bee Swarm Optimization which is a hybrid metaheuristic between the Bee Algorithm and the Particle Swarm Optimization. The experiments were designed with the same parameters for the three metaheuristics to give them the same characteristics in order to be equal between them, and the initial population was generated by a pseudorandom number generator, the generator provide by Java, a uniformly distributed generator and a greedy generator, using the Greedy Algorithm.

After applying the BSO, with the three initial population, to the different instances tests we can see that these metaheuristics give a small Confidence Interval and in most cases its Confidence Interval is better that the PSO's and BA's. In the worst case the Confidence Interval is the same that the PSO's. In the Uncorrelated instances, Subset-sum and Uncorrelated similar weight, the results were similar between PSO and BSO, and in the Weakly correlated instances and Strongly correlated the results were better for BSO than for PSO. Finally in the Inverse Strongly correlated and Almost Strongly Correlated the results were much better for BSO than for PSO.

We can see in all the graphics the metaheuristic behavior, and we can observe that the BA is the worst metaheuristic because it always yields a highly variable result, because of this it's Standard Deviation is so high, only with the Greedy base initial population gives the same result that the provide by the Greedy Algorithm or improve it a little. The PSO yields more consistent results, it's graphs show that this metaheuristic in the most of the cases gives good results.

We can conclude that have a good initial population for the metaheuristics is essential to have good results and we can say that the BSO is an effective metaheuristic to solve the Knapsack Problem, with all the initial population used, because each time that it's runned, it gives a good solution, and this solution is better that the solutions obtained by the other metaheuristics. Overall the results present a low Standard Deviation and thus a short Confidence Interval.

AUTHORS
Marco Aurelio Sotelo-Figueroa*, Rosario Baltazar, Juan Martín Carpio - Instituto Tecnológico de León, Av. Tecnológico S/N, 37290, León, Guanajuato, México.
E-mails: masotelof@gmail.com,
charobalmx@yahoo.com.mx,
jmcarpio61@hotmail.com .
* Corresponding author

References

[1] Forrest J.J.H., Kalagnanam J., Ladanyi L., "A Column-Generation Approach to the Multiple Knapsack Problem with Color Constraints", *INFORMS Journal on Computing*, 2006, pp. 129-134.

[2] Garey Michael R., David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*I, 1979.

[3] Kellerer H., Pferschy U., Pisinger D., *Knapsack Problems*, Stringer, Berlin, 2004.

[4] Kennedy J., Eberhart R.C., "Particle Swarm Optimization". In:, *IEEE International Conference Neural Networks*, vol. 4, 1995, pp. 1942-1948.

[5] Kennedy J., Eberhart R.C., *Swarm Intelligence*, Academic Press, EUA, 2001.

[6] Kiwiel K. C., "Breakpoint searching algorithms for the continuous quadratic knapsack problem", *Math. Program*, Spriger-Verlag, 2008, pp. 473-491.

[7] Maurice C., *Particle Swarm Optimization*, ISTE Ldt, USA, 2006.

[8] McDuff-Spears W., *Using neural networks and genetic algorithms as Heuristics for NP-complete problems*, Thesis of Master of Science in Computer Science, George Mason University, Virginia, 1989.

[9] Parsopoulus K.E., Vrahatis M.N., *Initializing the particle swarm optimizer using nonlinear simplex method*, *Advances in Intelligent Systems*, *Fuzzy Systems*, *Evolutionary Computation*, 2002, pp. 216-221.

[10] Pham D., Ghanbarzadeh A., Koç E., Otris S., Rahim S., Zaidi M., "The bee algorithm - a novel tool for complex optimization problems", *Intelligent Production Machines and Systems*, 2006.

[11] Pisinger D., "Core Problems in Knapsack Algorithms", *Operation Research*, vol. 47, 1999, pp. 570-575.

[12] Pisinger D., "Where Are The Hard Knapsack problems?", *Computers & Operation Research*, vol. 32, 2005, pp. 2271-2282.

[13] Pisinger D., Rasmussen, A. Sandvik R., "Solution of Large Quadratic Knapsack Problems Through Aggressive Reduction, *INFORMS Journal on Computing*, INFORMS, 2007, pp. 280-290.

[14] Shahriar A. Z. M. *et al*., "A multiprocessor based heuristic for multi-dimensional multiple-choice knapsack problem", *J. Supercomput*, 2008, pp. 257-280.

[15] Silvano M., Toth P., *Knapsack Problem, Algorithms and Computer Implementations*, John Wiley and Sons , New York USA , 1990.

[16] Yamada, T., Watanabe K., and Kataoka, S., Algorithms to solve the knapsack constrained maximum spanning tree problem, *International Journal of Computer Mathematics*, Taylor & Francis Ltd, 2005, pp. 23-34.

[17] Zemel E., "An O(n) Algorithm for the linear multiple choice Knapsack problem and related problems", *Information Processing Problems*, North Holland, 1984, pp. 123-128.