Finding Solutions of the Set Covering Problem with an Artificial Fish Swarm Algorithm Optimization

Broderick Crawford[1,2,5], Ricardo Soto[1,3,4], Eduardo Olguín[5], Sanjay Misra[6], Sebastián Mansilla Villablanca[1(✉)], Álvaro Gómez Rubio[1], Adrián Jaramillo[1], and Juan Salas[1]

[1] Pontificia Universidad Católica de Valparaíso, 2362807 Valparaíso, Chile
{broderick.crawford,ricardo.soto}@pucv.cl,
{sebastian.mansilla.v,alvaro.gomez.r,adrian.jaramillo.s,
juan.salas.f}@mail.pucv.cl
[2] Universidad Central de Chile, 8370178 Santiago, Chile
[3] Universidad Autónoma de Chile, 7500138 Santiago, Chile
[4] Universidad Científica del Sur, Lima 18, Peru
[5] Facultad de Ingeniería y Tecnología, Universidad San Sebastián, Bellavista 7, 8420524 Santiago, Chile
eduardo.olguin@uss.cl
[6] Covenant University, Ogun 110001, Nigeria
sanjay.misra@covenantuniversity.edu.ng

Abstract. The Set Covering Problem (SCP) is a matrix that is composed of zeros and ones and consists in finding a subset of zeros and ones also, in order to obtain the maximum coverage of necessities with a minimal possible cost. In this world, it is possible to find many practical applications of this problem such as installation of emergency services, communications, bus stops, railways, airline crew scheduling, logical analysis of data or rolling production lines. SCP has been solved before with different nature inspired algorithms like fruit fly optimization algorithm. Therefore, as many other nature inspired metaheuristics which imitate the behavior of population of animals or insects, Artificial Fish Swarm Algorithm (AFSA) is not the exception. Although, it has been tested on knapsack problem before, the objective of this paper is to show the performance and test the binary version of AFSA applied to SCP, with its main steps in order to obtain good solutions. As AFSA imitates a behavior of a population, the main purpose of this algorithm is to make a simulation of the behavior of fish shoal inside water and it uses the population as points in space to represent the position of fish in the shoal.

Keywords: Set Covering Problem · Artificial Fish Swarm Optimization Algorithm · Metaheuristics · Combinatorial optimization

O. Gervasi et al. (Eds.): ICCSA 2016, Part I, LNCS 9786, pp. 166–181, 2016.
DOI: 10.1007/978-3-319-42085-1_13

1 Introduction

Metaheuristics provide "acceptable" solutions in a reasonable time for solving hard and complex problems in science and engineering when it is expensive to find the best solution especially with a computing power limited.

One of the classical problems that Metaheuristics try to solve is Set Covering Problem (SCP) which consists in finding a set of solutions at the lowest possible cost, accomplishing with the constraints of a matrix that has zeros and ones, where each row must be covered of at least one column. In the past, SCP has been solved with different algorithms such as cultural algorithm [5], fruit fly optimization algorithm [13] or teaching-learning-based optimization algorithm [14]. There are many applications of this problem such as optimal selection of ingot sizes [25] or assign fire companies to fire houses [26].

The main goal of this paper is to show the performance of Artificial Fish Swarm Algorithm (AFSA) applied to SCP, previously, it was tested on the knapsack problem [19–21]. This algorithm, simulates the behavior of a fish inside the water which belongs to a shoal and it uses a population of points in space to represent the position of fish in the shoal. In the original version of AFSA, there are five main behaviors such as random, chasing, swarming, searching and leaping. In the following work, it will be showed its simplified version of AFSA in order to solve SCP. The proposed algorithm has the following steps, initialization of its population, generation of trial points, the effect-based crossover, dealing with SCP constraints, selection of a new population, reinitialization of the population, local search and termination conditions. This method will be tested on each one of the 70 SCP benchmarks obtained from OR-Library website. These 70 files are formatted as: number of rows n, number of columns m, the cost of each column $c_j, j \in \{1, \ldots, n\}$, and for each row $i, i \in \{1, ..., m\}$ the number of columns which cover row i followed by a list of the columns which cover rows i. These 70 files were chosen in order to solve SCP in an academic and theoretical way.

The remainder paper is organized as follows: In Sect. 2, it will be explained the set covering problem. In Sect. 3, it is going to be presented the artificial fish swarm algorithm in general terms. In Sect. 4, It will be showed the AFSA method and its simplified version in order to solve the SCP with its main steps for solving this problem and, it is going to be illustrated the proposed algorithm. In Sect. 5, it will be exposed the experimental results. Finally, in Sect. 6, it is going to be presented the conclusions of this paper.

2 Set Covering Problem

This is a classical and well-known NP-hard problem. It is a representation of a sort of combinatorial optimization problem which has many practical applications in the world such as construction of firemen stations in different places or installation of network cell phones in order to obtain the maximum coverage with a minimal possible cost. The SCP can be formulated as follows [1]:

$$\text{minimize} \quad Z = \sum_{j=1}^{n} c_j x_j \tag{1}$$

Subject to:

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1 \quad \forall i \in I \tag{2}$$

$$x_j \in \{0, 1\} \quad \forall j \in J \tag{3}$$

where $A = (a_{ij})$ be a $m \times n$ 0–1 matrix with $I = \{1, \ldots, m\}$ and $J = \{1, \ldots, n\}$ be the row and column sets respectively. Column j can be covered a row i if $a_{ij} = 1$. Where c_j is a nonnegative value that represents the cost of selecting the column j and x_j is a decision variable, that can be 1 if column j is selected or 0 otherwise. The objective is to find a minimum cost subset $S \subseteq J$, such that each row $i \in I$ is covered by at least one column $j \in S$.

2.1 How Has It Been Solved Before?

There are two kinds of methods that have been used to solve the SCP. First, the methods which produce optimal solutions but sometimes need a lot of time and/or high computational cost. Those methods could be constraint programming, branch and bound or integer linear programming. On the other hand, there are the metaheuristics that provide "acceptable or good" solutions in a reasonable time. Even, it is possible to find optimal solutions with many metaheuristics.

2.2 Metaheuristics that Have Solved SCP

In the past, the SCP was successfully solved with many metaheuristics such as artificial bee colony [2–4], cultural algorithm [5], swarm optimization particles [6], ant colony optimization [7], firefly algorithm [8–10], shuffled frog leaping algorithm [11,12], fruit fly optimization algorithm [13], teaching-learning-based optimization algorithm [14], or genetic algorithm [15,16,27]. In others works there are comparisons among different kind of metaheuristics [17], or comparisons among different kinds of nature-inspired metaheuristics [18] in order to solve the SCP.

2.3 Metaheuristics, Bio-Inspired and AFSA

As it mentioned before, one of the main advantage of metaheuristics is to provide good solutions in a "reasonable" time, especially when the computing resources are not infinite. On the other hand, metaheuristics not always provide the optimal results. However, in real life not always is required the best known solution

because this world moves quickly and it is necessary to obtain responses in little time, for instance, in companies when they try to increase benefits or decrease costs. Moreover, metaheuristics can be helped by other techniques, according to [23], such as handling of constraints that problem impose and/or the dimension reduction of the problem.

Bio-inspired metaheuristics are a sort of metaheuristics and they are methods that simulate the behavior of a swarm or group of animals. Also, they try to solve problems of optimization such as maximization or minimization. In this case, AFSA was created with the intention of solving the knapsack problem which is a kind of maximization problem. Hence, the proposed work has the objective of transforming the application of a maximization problem into a minimization problem and observe its results because many others nature inspired metaheuristics have been good results on SCP, as it mentioned above.

3 Artificial Fish Swarm Algorithm

As many other nature inspired metaheuristics which imitate the behavior of population of animals or insects, AFSA is not the exception. According to [19], this algorithm was proposed and applied in order to solve optimization problems and it simulates the behavior of a fish swarm inside the water where a fish represents a point or a fictitious entity of a true fish in a population and the swarm movements are randomly.

3.1 Main Behaviors of AFSA

The fish swarm behavior is summarized as follows [19]:

1. **Random Behavior:** In order to find companion and food, a fish swims randomly inside water.
2. **Chasing Behavior:** If food is discovered by a fish, the others in the neighborhood go quickly after it.
3. **Swarming Behavior:** In order to guarantee the survival of the swarm and avoid dangers from predatory, Fish come together in groups.
4. **Searching Behavior:** Fish goes directly and quickly to a region, when that region is discovered with more food by it. That can be by vision or sense.
5. **Leaping Behavior:** Fish leaps to look for food in other regions, when it stagnates in a region.

These five behaviors are the responsible that the artificial fish swarm tries to search good results. Also, AFSA works with feasible solutions.

3.2 Another Description of AFSA

Additionally to the explanation showed above there is another description of AFSA which is proposed with more details in [20]. The concept of "visual scope" is the main concept utilized in that version of AFSA, and it represents how close is the neighborhood in comparison to a point/fish.

Depending on the position of a point related to the population, there could occur three situations [20]:

(a) The "visual scope" is empty, and the current point with no other points in its neighborhood, moves randomly looking for a better region.
(b) When the "visual scope" is not crowded, the current point can move towards the best point inside the "visual scope", or, if this best point does not improve the objective function value it moves towards the central point of the "visual scope".
(c) When the "visual scope" is crowded, the current point has some difficulty in following any particular point, and searches for a better region by choosing randomly another point (from the "visual scope") and moving towards it.

The condition that decides when the "visual scope" of the current point is not crowded and the central point inside the "visual scope" are explained in [20].

3.3 Proposed Algorithm of AFSA Binary Version

Before of applying AFSA on SCP, it is necessary to show the binary version of this algorithm which was proposed by [19], but it was applied on the knapsack problem. That algorithm is the following:

Algorithm 1. Binary version of AFSA

```
1: Set parameter values
2: Set t = 1 and randomly initialize x^{i,t}, i = 1, 2, ..., N
3: Perform decoding and evaluate z. Identify x^{max} and z_{max}
4: if Termination condition is met then
5:     Stop
6: end if
7: for all x^{i,t} do
8:     Calculate"visual scope" and "crowding factor"
9:     Perform fish behavior to create trial point y^{i,t}
10:     Perform decoding to make the trial point feasible
11: end for
12: Perform selection according to step 4 to create new current points
13: Evaluate z and identify x^{max} and z_{max}
14: if t%L = 0 then
15:     Perform leaping
16: end if
17: Set t = t + 1 and go to step 4
```

4 Artificial Fish Swarm Algorithm and Its Simplified Binary Version

In this section, it is going to be showed the main steps of AFSA and its simplified binary version in order to solve SCP. According to the authors [21] AFSA converges to a non-optimal solution in previous versions like [20]. Therefore, there were proposed some modifications of AFSA and they were slightly modified in order to solve SCP.

4.1 Features that Were Modified in AFSA

In [21] the main modifications were: the "visual scope" concept was rejected; The behavior depends on two probability values; Swarming behavior is never utilized; An effect-based crossover is used instead of an uniform crossover in different behaviors to create trial points; A local search with two steps was implemented; Among other modifications that are explained with more details in [21]. Also, it was introduced a repair function for handling the SCP constaraints.

Next, it will be explained the steps of AFSA in order to obtain SCP results.

4.2 Initialization of Population

As many other metaheuristics, it is necessary to initialize the population with objective to find good solutions. Therefore, the best representation of a population is N current points, $\boldsymbol{x}^i$, where $i \in \{1, 2, ..., N\}$ each one represented by a binary string of 0/1 bits of length n and are randomly generated.

4.3 Generation of Trial Population

This metaheuristic works with a trial population at each iteration. So, in order to create trial points in consecutive iterations based on behaviors of random, chasing, and searching is necessary utilize crossover and mutation after the initialization of population. In [21] probabilities of $0 \leq \tau_1 \leq \tau_2 \leq 1$ were introduced and they are the responsible to reach this objective.

Random Behavior: If a fish does not have companion in its neighborhood, then it moves randomly looking for food in another region [21]. This happens when a random number $rand(0, 1)$ is less than or equal to τ_1. The trial point $\boldsymbol{y}^i$ is created randomly setting 0/1 bits of length n [21].

Chasing Behavior: When a fish, or a group of fish in the swarm, discover food, and the others go quickly after it [21]. This happens when $rand(0, 1) \geq \tau_2$ and it is related to the movement towards the best point found so far in the population, $\boldsymbol{x}^{min}$. The trial point $\boldsymbol{y}^i$ is created using an effect-based crossover (see *Algorithm* 2) between $\boldsymbol{x}^i$ and $\boldsymbol{x}^{min}$ [21].

Searching Behavior: When fish discovers a region with more food, by vision or sense, it goes directly and quickly to that region [21]. This behavior is related to the movement towards a point $\boldsymbol{x}^{rand}$ where "rand" is an index randomly chosen from the set $\{i = 1, 2, ..., N\}$. When $\tau_1 < rand(0,1) < \tau_2$ it is implemented. An effect-based crossover between $\boldsymbol{x}^{rand}$ and $\boldsymbol{x}^i$ is utilized to create the trial point $\boldsymbol{y}^i$ [21].

Trial Point Corresponding to the Best Point: In [21], the 3 behaviors explained above are implemented to create $N-1$ trial points; the best point $\boldsymbol{x}^{min}$ uses a 4 flip-bit mutation. It is performed on the point $\boldsymbol{x}^{min}$ to create the corresponding trial point $\boldsymbol{y}^i$. In this operation 4 positions are randomly selected, and the bits of the corresponding positions are changed from 0 to 1 or vice versa [21].

4.4 The Effect-Based Crossover in Simplified Binary Version of AFSA

In order to obtain the trial point in chasing and searching behavior, it necessary to calculate the $effect\ ratio\ ER_{\boldsymbol{u},x^i}$ of u on the current point $\boldsymbol{x}^i$, according to [21]. It can obtain with the following two formulas:

$$ER_{u,x^i} = \frac{q(u)}{q(u) + q(x^i)} \tag{4}$$

$$q(x^i) = \exp[\frac{-(z(x^{min}) - z(x^i))}{(z(x^{min}) - z(x^{max}))}] \tag{5}$$

$u = \boldsymbol{x}^{min}$ is used with chasing behavior, $u = \boldsymbol{x}^{rand}$ is used with searching behavior and $\boldsymbol{x}^{max}$ is the worst point of the population. The effect-based crossover to obtain the trial point $\boldsymbol{y}^i$ is showed in Algorithm 2 according to [21].

Algorithm 2. Effect-based crossover

Require: current point x^i, u and ER_{u,x^i}
for $j = 1$ to n do
  if $rand(0,1) < ER_{u,x^i}$ then
    $y^i_j = u_j$
  else
    $y^i_j = x_j$
  end if
end for
return trial point y^i

4.5 Deal with Constraints of SCP

In order to obtain good results, it is necessary to introduce an appropriate method that helps with SCP constraints. Therefore, it is going to be showed a repair function for handling the SCP constraints in the Algorithm 3. According to [23], Algorithm 3 shows a repair method where all rows not covered are

identified and the columns required are added. Hence, in this way all the constraints will be covered. The search of these columns are based in the relationship showed in the next equation.

$$\frac{cost\ of\ one\ column}{amount\ of\ columns\ not\ covered} \tag{6}$$

Once the columns are added and the solution is feasible, a method is applied to remove redundant columns of the solution. The redundant columns are those that are removed, the solution remains a feasible solution. The algorithm of this repair method is detailed in the Algorithm 3. Where:

(a) I is the set of all rows
(b) J is the set of all columns
(c) J_i is the set of columns that cover the row $i, i \in I$
(d) I_j is the set of rows covered by the column $j, j \in J$
(e) S is the set of columns of the solution
(f) U is the set of columns not covered
(g) w_i is the number of columns that cover the row $i, \forall i \in I$ in S

Algorithm 3. Repair Operator for Dealing with SCP Constraints

```
w_i ←| S ∩ J_i | ∀i ∈ I;
U ← {i | w_i = 0}, ∀i ∈ I;
for i ∈ U do
    find the first column j in J_i that minimize c_j/|U∩I_j| S ← S ∩ j;
    w_i ← w_i + 1, ∀i ∈ I_j;
    U ← U − I_j;
end for
for j ∈ S do
    if w_i ≥ 2, ∀i ∈ I_j then
        S ← S − j;
        w_i ← w_i − 1, ∀i ∈ I_j;
    end if
end for
```

4.6 Selection of New Population

The new population is selected between trial population and current population. Each trial point contends against the current point, therefore, if $z(\boldsymbol{y}^i) \leq z(\boldsymbol{x}^i)$, then the trial point becomes a member of the new population to the next iteration; otherwise, the current point is maintained to the next iteration, according to [21].

$$x^{i,t+1} = \begin{cases} \boldsymbol{y}^{i,t}\ if\ \ z(\boldsymbol{y}^{i,t}) \leq z(\boldsymbol{x}^{i,t}),\ i = 1, 2, ..., N \\ \\ \boldsymbol{x}^{i,t}\ \text{otherwise} \end{cases} \tag{7}$$

4.7 Reinitialization of Current Population

In [21], Every certain iterations, this metaheuristic replaces the population of the last iteration with a new population to the next iteration. Therefore, utilizing the same values, it will be done a randomly reinitialization of 50 % of the population at every R iterations, where R is a positive integer parameter.

4.8 Exploitation or Local Search

Exploitation is related to leaping behavior of AFSA. So, according to the authors [21], in oder to obtain better solutions and improve old versions of AFSA, the concept of exploitation/local search was utilized and its purpose is to find better solutions when the method obtains the same solution as the iterations pass. This is based on a flip-bit mutation which N_{loc} points are selected randomly from the population, where $N_{loc} = \tau_3 N$ with $\tau_3 \in (0, 1)$. This mutation changes the bit values of those points from 0 to 1 and vice versa according to p_m probability. After that, those new points are made feasible by using the repair function of SCP explained in Sect. 4.5. Then they become members of the population, if they improve z_{min} at that moment. Afterwards, the best point of the population is identified and another mutation is operated on N_{ref} positions, with $N_{ref} = \tau_3 n$, those positions are randomly selected from the point [21]. Then it becomes a member of the population, if it improves z_{min} at that moment. This mutation is used L times, where L is s a positive integer parameter.

4.9 Conditions to Finish the Algorithm

As many other metaheuristics, it is necessary to stop this metaheuristic when it is almost impossible to find a better solution and it is unnecessary continuing wasting the computational resource. Therefore, in accordance with [21], AFSA terminates when the known optimal solution is reached or a maximum number of iterations, T_{max}, is exceeded.

$$t > T_{max} \ \ or \ \ z_{min} \leq z_{opt} \tag{8}$$

where z_{min} is the best objective function value attained at iteration t and z_{opt} is the known optimal value available in the literature.

4.10 AFSA-SCP Proposed Algorithm

After showing the main steps of this algorithm, it is going to show the proposed algorithm by mean of two tools. First, the flow chart of this algorithm in Fig. 1. Then it will be showed the pseudocode of the proposed method which is shown in Algorithm 4 and it has the appropriate modifications in order to solve SCP.

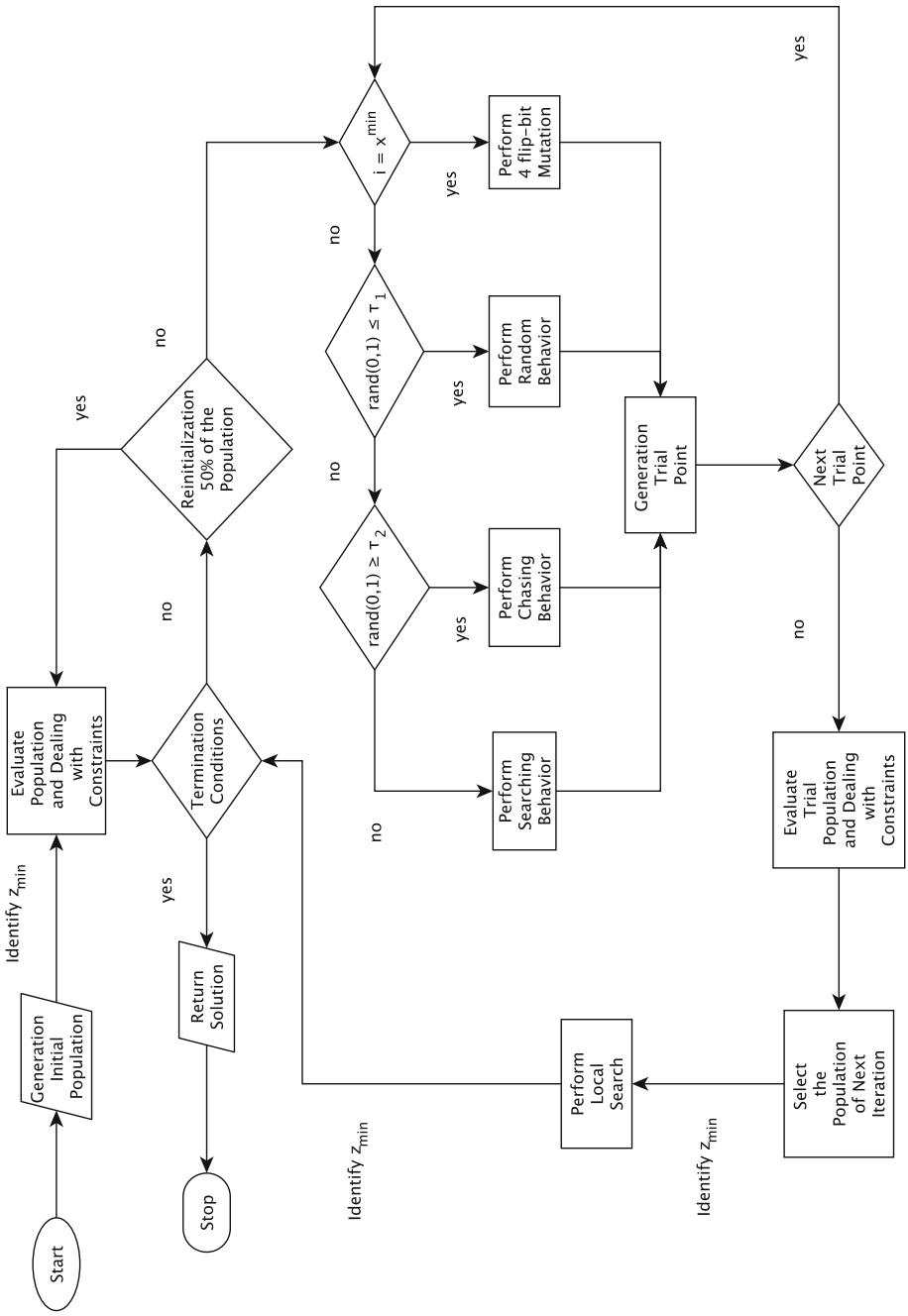

Fig. 1. Flow Chart AFSA

Algorithm 4. AFSA applied to SCP

```
Require: T_max and z_opt and other values of parameters
Set t = 1 Initialize population x^{i,t}, i = 1, 2, ..., N
Execute SCP repair function in order to evaluate the population, identify x^{min} and z_{min}
while t ≤ T_max or z_{min} ≥ z_{opt} do
    if t%R = 0 then
        Reinitialize 50% of the population, keeping x^{min} and z_{min}
        Execute SCP repair function in order to evaluate population, identify x^{min} and
            z_{min}
    end if
    for i = 1 to N do
        if i = x^{min} then
            Execute 4 flip-bit mutation to create trial point y^{i,t}
        else
            if rand(0,1) ≤ τ1 then
                Execute random behavior to create trial point y^{i,t}
            else if rand(0,1) ≥ τ2 then
                Execute chasing behavior to create trial point y^{i,t}
            else
                Execute searching behavior to create trial point y^{i,t}
            end if
        end if
    end for
    Execute SCP repair function in order to evaluate and get y^{i,t}, i = 1, 2, ..., N and
        evaluate them
    Select new population x^{i,t+1}, i = 1, 2, ..., N
    if t%L = 0 then
        Execute exploitation/local search - leaping behavior
        Identify x^{min} and z_{min}
    end if
    Set t = t + 1
end while
return x^{min} and z_{min}
```

5 Experimental Results

After many experiments, it is going to be showed the obtained results after performing AFSA to solve SCP. At the final of this section it is possible to find the Tables 1 and 2, which shows the results of SCP with more details.

In other related works, algorithms were run 30 times for each instance, also it is an accepted number in the literature. Therefore, this algorithm proposed was run with that number of times. Moreover, this algorithm needs almost 20 h to analyze the last 10 files, sets from NRG to NRH, because each one has a matrix with a big dimension, great deal of rows and columns. That should not be considered a problem in academic area but that could be considered a problem if it is applied in a real problem due to the big quantity of hours.

This algorithm tested the 70 data files from the OR-Library, 25 of them are the instance sets $4, 5, 6$ was originally from Balas and Ho [22], the others 25, the sets A, B, C, D, E from Beasley [23] and 20 of these data files are the test problem sets E, F, G, H from Beasley [24]. These 70 files are formatted as: number of rows n, number of columns m, the cost of each column $c_j, j \in \{1, \ldots, n\}$, and for each row $i, i \in \{1, ..., m\}$ the number of columns which cover row i followed by a list of the columns which cover rows i.

Table 1. Experimental results of SCP benchmark sets (4, 5, 6, A, B, C, D, E, NRE, NRF, NRG and NRH)

Number	*NumberofFiles*	*Instance*	ARPD
1	10	4	0,84
2	10	5	0,83
3	5	6	0,78
4	5	A	1,83
5	5	B	2,92
6	5	C	2,29
7	5	D	3,65
8	5	E	0,0
9	5	NRE	4,93
10	5	NRF	7,16
11	5	NRG	5,82
12	5	NRH	6,72

This algorithm was implemented in Java programming language, using Eclipse IDE, with the following hardware: Intel core i5 dual core 2.60 GHz processor, 8 GB RAM and it was run under OSX Yosemite.

Finally, the program was executed only with feasible solutions, with a population of $N = 20$ fish, probability $\tau_1 = 0.1$, probability $\tau_2 = 0.9$, probability $\tau_3 = 0.1$, probability $p_m = 0.1$, $L = 50$, reinitialization of population $R = 10$ and each trial was run 1000 iterations, and 30 times each one. After obtained all results, it was obtained the averages values from these 30 times for each one of the files.

Table 1 shows an overview of all sets. It contains the instance number, set number and $ARPD$ which is an average of the deviation of the objective value (best known solution). With these results, it is possible to show that the best results which have an $ARPD$ minor to 1 %, are the groups $4, 5, 6$ and E, and due to that reason they were selected because they have better results in comparison with other benchmark groups. Hence, Table 2 shows the best results obtained where the first column is the number of experiment of each instance, the second column $Instance$ indicates each benchmark evaluated, and Z_{opt} shows the best known solution value of each instance. The next columns Z_{min}, Z_{max}, Z_{avg} represents the minimum, maximum among minimums, and average of minimums solutions obtained. The last column reports the relative percentage deviation RPD which represents the deviation of the objective value or best known solution f_{opt} from f_{min} which is the minimum value obtained for each instance. RPD was calculated as follows:

$$\text{RPD} = \frac{100(f_{min} - f_{opt})}{f_{opt}} \tag{9}$$

Table 2. Experimental results of SCP benchmark sets (4, 5, 6 and E)

Number	*Instance*	Z_{opt}	Z_{min}	Z_{max}	Z_{avg}	RPD
1	4.1	429	430	445	437,4	0,23
2	4.2	512	515	546	530,83	0,59
3	4.3	516	519	543	528,27	0,58
4	4.4	494	495	532	514,83	0,20
5	4.5	512	514	536	521,73	0,39
6	4.6	560	565	597	580,9	0,89
7	4.7	430	432	447	437,37	0,47
8	4.8	492	492	514	501,73	0,0
9	4.9	641	658	688	669,8	2,65
10	4.10	514	525	559	539,6	2,14
11	5.1	253	254	271	263,03	0,40
12	5.2	302	310	318	314,27	2,65
13	5.3	226	228	244	232,77	0,88
14	5.4	242	242	247	244,77	0,0
15	5.5	211	212	215	212,6	0,47
16	5.6	213	214	242	227,77	0,47
17	5.7	293	299	315	307,9	2,05
18	5.8	288	291	313	298,97	1,04
19	5.9	279	279	296	285,73	0,0
20	5.10	265	266	276	272,07	0,38
21	6.1	138	138	153	146,37	0,0
22	6.2	146	149	156	151,97	2,05
23	6.3	145	145	161	149,63	0,0
24	6.4	131	131	137	134,17	0,0
25	6.5	161	164	181	172,67	1,86
26	E.1	5	5	6	5,87	0,0
27	E.2	5	5	6	5,5	0,0
28	E.3	5	5	6	5,2	0,0
29	E.4	5	5	6	5,7	0,0
30	E.5	5	5	6	5,57	0,0

6 Conclusions

AFSA optimization has a good performance with some instances. It has been observed that sets $4, 5, 6$ and E obtained each one an $ARPD$ minor to $1\,\%$, and sets A, B, C, D obtained an $ARPD$ minor to $4\,\%$. Then with the last 20 instances the results of $ARPD$ start to decrease. Due to that reason sets $4, 5, 6$ and E were

selected because they have better results in comparison with other benchmark groups and their results were showed in Table 2. Moreover, in comparison to other metaheuristics such as Artificial Bee Colony Algorithm [4], Binary Firefly Algorithm [8] or Genetic Algorithm [27], the results of AFSA are similar with sets $4, 5, 6$ and E and worse than the other groups. For this reason, this technique requires more study.

It was observed that in the first 200 iterations, this algorithm converges quickly to very good solutions or optimal solutions in some cases. Between 200 and 1000 iterations, sometimes, the algorithm can obtain a slightly better results if it does not obtain the optimal result before but in other cases it maintain the same result reached at first 200 iterations. Therefore, It could be possible that it is necessary more than 1000 iterations and/or find a better configuration of parameters to obtain the optimal results or closest to the optimal in all instances. Thus, reduce the variability in its results.

Another characteristic that was observed is that AFSA requires a lot of processing time with some benchmarks, especially the last 10 files, sets from NRG to NRH need almost 20 h. Therefore, It could be possible that it is necessary to introduce a the technique of the dimension reduction of the problem in order to reduce the processing time.

Although, this paper showed that this algorithm is a good way to solve SCP with feasible solutions, in a binary domain. An interesting future work is to apply other versions of AFSA on SCP or making a comparison among the different versions of AFSA.

References

1. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co, New York (1990)
2. Crawford, B., Soto, R., Aguilar, R.C., Paredes, F.: A new artificial bee colony algorithm for set covering problems. Electr. Eng. Inf. Technol. **63**, 31 (2014)
3. Crawford, B., Soto, R., Aguilar, R.C., Paredes, F.: Application of the Artificial Bee Colony Algorithm for Solving the Set Covering Problem. Sci. World J. 2014 (2014)
4. Cuesta, R., Crawford, B., Soto, R., Paredes, F.: An artificial bee colony algorithm for the set covering problem. In: Silhavy, R., Senkerik, R., Oplatkova, Z.K., Silhavy, P., Prokopova, Z. (eds.) CSOC 2014. AISC, vol. 285, pp. 53–63. Springer, Switzerland (2014)
5. Crawford, B., Soto, R., Monfroy, E.: Cultural algorithms for the set covering problem. In: Tan, Y., Shi, Y., Mo, H. (eds.) ICSI 2013, Part II. LNCS, vol. 7929, pp. 27–34. Springer, Heidelberg (2013)
6. Crawford, B., Soto, R., Monfroy, E., Palma, W., Castro, C., Paredes, F.: Parameter tuning of a choice-function based hyperheuristic using Particle Swarm Optimization. Expert Syst. Appl. **40**(5), 1690–1695 (2013)
7. Crawford, B., Soto, R., Monfroy, E., Paredes, F., Palma, W.: A hybrid Ant algorithm for the set covering problem (2014)
8. Crawford, B., Soto, R., Olivares-Suárez, M., Paredes, F.: A binary firefly algorithm for the set covering problem. Modern Trends Tech. Comput. Sci. **285**, 65–73 (2014)

9. Crawford, B., Soto, R., Riquelme-Leiva, M., Peña, C., Torres-Rojas, C., Johnson, F., Paredes, F.: Modified binary firefly algorithms with different transfer functions for solving set covering problems. In: Silhavy, R., Senkerik, R., Oplatkova, Z.K., Prokopova, Z., Silhavy, P. (eds.) CSOC 2015. AISC, vol. 349, pp. 307–315. Springer, Switzerland (2015)
10. Crawford, B., Soto, R., Olivares-Suárez, M., Paredes, F.: A new approach using a binary firefly algorithm for the set covering problem. WIT Trans. Inf. Commun. Technol. **63**, 51–56 (2014)
11. Crawford, B., Soto, R., Peña, C., Palma, W., Johnson, F., Paredes, F.: Solving the set covering problem with a shuffled frog leaping algorithm. In: Nguyen, N.T., Trawiński, B., Kosala, R. (eds.) ACIIDS 2015. LNCS, vol. 9012, pp. 41–50. Springer, Heidelberg (2015)
12. Crawford, B., Soto, R., Peña, C., Riquelme-Leiva, M., Torres-Rojas, C., Johnson, F., Paredes, F.: Binarization methods for shuffled frog leaping algorithms that solve set covering problems. Software Engineering in Intelligent Systems, pp. 317–326 (2015)
13. Crawford, B., Soto, R., Torres-Rojas, C., Peña, C., Riquelme-Leiva, M., Misra, S., Johnson, F., Paredes, F.: A binary fruit fly optimization algorithm to solve the set covering problem. In: Gervasi, O., Murgante, B., Misra, S., Gavrilova, M.L., Rocha, A.M.A.C., Torre, C., Taniar, D., Apduhan, B.O. (eds.) ICCSA 2015. LNCS, vol. 9158, pp. 411–420. Springer, Heidelberg (2015)
14. Crawford, B., Soto, R., Aballay, F., Misra, S., Johnson, F., Paredes, F.: A teaching-learning-based optimization algorithm for solving set covering problems. In: Gervasi, O., Murgante, B., Misra, S., Gavrilova, M.L., Rocha, A.M.A.C., Torre, C., Taniar, D., Apduhan, B.O. (eds.) ICCSA 2015. LNCS, vol. 9158, pp. 421–430. Springer, Heidelberg (2015)
15. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, 3rd edn. Springer, Heidelberg (1996)
16. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer Science & Business Media, Heidelberg (2013)
17. Soto, R., Crawford, B., Galleguillos, C., Barraza, J., Lizama, S., Muñoz, A., Vilches, J., Misra, S., Paredes, F.: Comparing cuckoo search, bee colony, firefly optimization, and electromagnetism-like algorithms for solving the set covering problem. In: Gervasi, O., Murgante, B., Misra, S., Gavrilova, M.L., Rocha, A.M.A.C., Torre, C., Taniar, D., Apduhan, B.O. (eds.) ICCSA 2015. LNCS, vol. 9155, pp. 187–202. Springer, Heidelberg (2015)
18. Crawford, B., Soto, R., Peña, C., Riquelme-Leiva, M., Torres-Rojas, C., Misra, S., Johnson, F., Paredes, F.: A comparison of three recent nature-inspired metaheuristics for the set covering problem. In: Gervasi, O., Murgante, B., Misra, S., Gavrilova, M.L., Rocha, A.M.A.C., Torre, C., Taniar, D., Apduhan, B.O. (eds.) ICCSA 2015. LNCS, vol. 9158, pp. 431–443. Springer, Heidelberg (2015)
19. Azad, M.A.K., Rocha, A.M.A.C., Fernandes, E.M.G.P.: Solving multidimensional 0–1 knapsack problem with an artificial fish swarm algorithm. In: Murgante, B., Gervasi, O., Misra, S., Nedjah, N., Rocha, A.M.A.C., Taniar, D., Apduhan, B.O. (eds.) ICCSA 2012, Part III. LNCS, vol. 7335, pp. 72–86. Springer, Heidelberg (2012)
20. Azad, M.A.K., Rocha, A.M.A., Fernandes, E.M.: Improved binary artificial fish swarm algorithm for the 0–1 multidimensional knapsack problems. Swarm Evol. Comput. **14**, 66–75 (2014)

21. Azad, M.A.K., Rocha, A.M.A., Fernandes, E.M.: Solving large 0–1 multidimensional knapsack problems by a new simplified binary artificial fish swarm algorithm. J. Math. Model. Algorithms Oper. Res. **14**(3), 1–18 (2015)
22. Balas, E., Ho, A.: Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study. In: Padberg, M.W. (ed.) Combinatorial Optimization, vol. 12, pp. 37–60. Springer, Heidelberg (1980)
23. Beasley, J.E.: An algorithm for set covering problem. Eur. J. Oper. Res. **31**(1), 85–93 (1987)
24. Beasley, J.E.: A lagrangian heuristic for set-covering problems. Naval Res. Logistics (NRL) **37**(1), 151–164 (1990)
25. Vasko, F.J., Wolf, F.E., Stott, K.L.: Optimal selection of ingot sizes via set covering. Oper. Res. **35**(3), 346–353 (1987)
26. Walker, W.: Using the set-covering problem to assign fire companies to fire houses. Oper. Res. **22**(2), 275–277 (1974)
27. Beasley, J.E., Chu, P.C.: A genetic algorithm for the set covering problem. Eur. J. Oper. Res. **94**(2), 392–404 (1996)