Computers, Materials & Continua
DOI:10.32604/cmc.2022.023068
Article

Tech Science Press

Binary Fruit Fly Swarm Algorithms for the Set Covering Problem

Broderick Crawford[1,*], Ricardo Soto[1], Hanns de la Fuente Mella[1], Claudio Elortegui[1], Wenceslao Palma[1], Claudio Torres-Rojas[1], Claudia Vasconcellos-Gaete[2], Marcelo Becerra[1], Javier Peña[1] and Sanjay Misra[3]

[1]Pontificia Universidad Católica de Valparaíso, Valparaíso, 2362807, Chile
[2]LERIA, Université d'Angers, Angers, 49000, France
[3]Department of Computer Science and Communication, Ostfold University College, Halden, Norway
*Corresponding Author: Broderick Crawford. Email: broderick.crawford@pucv.cl
Received: 27 August 2021; Accepted: 21 October 2021

Abstract: Currently, the industry is experiencing an exponential increase in dealing with binary-based combinatorial problems. In this sense, metaheuristics have been a common trend in the field in order to design approaches to solve them successfully. Thus, a well-known strategy consists in the use of algorithms based on discrete swarms transformed to perform in binary environments. Following the No Free Lunch theorem, we are interested in testing the performance of the Fruit Fly Algorithm, this is a bio-inspired metaheuristic for deducing global optimization in continuous spaces, based on the foraging behavior of the fruit fly, which usually has much better sensory perception of smell and vision than any other species. On the other hand, the Set Coverage Problem is a well-known NP-hard problem with many practical applications, including production line balancing, utility installation, and crew scheduling in railroad and mass transit companies. In this paper, we propose different binarization methods for the Fruit Fly Algorithm, using S-shaped and V-shaped transfer functions and various discretization methods to make the algorithm work in a binary search space. We are motivated with this approach, because in this way we can deliver to future researchers interested in this area, a way to be able to work with continuous metaheuristics in binary domains. This new approach was tested on benchmark instances of the Set Coverage Problem and the computational results show that the proposed algorithm is robust enough to produce good results with low computational cost.

Keywords: Set covering problem; fruit fly swarm algorithm; metaheuristics; binarization methods; combinatorial optimization problem

This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1 Introduction

The Set Covering Problem (SCP) is a well-known NP-hard class covering problem, which consists of finding a subset of columns in a zero-one matrix such that it covers all rows of the matrix at minimum cost. It has important practical applications, such as: localization of emergency services [1], scheduling of crews in mass transit companies [2], routing of vehicles [3], reconstruction of sibling relationships [4].

Considering the complex nature of SCP, the huge size of the real data sets and the variety of methods designed to approach similar problems, SCP has been solved by exact methods, metaheuristics and other techniques, such as hyperheuristics [5] or with Machine Learning techniques [6–8]. Solving by exact methods is mainly based on Branch-and-Bound, Branch-and-Cut and Lagrangean heuristics [9].

Resolution by metaheuristics includes Genetic Algorithms [10], Tabu Search [11], Ant Colony Optimization [12], Artificial Bee Colonies [13], Firefly Algorithms [14], Cat Swarm Optimization [15], Cuckoo Search [16], Teaching-learning Based Optimization [17] and Shuffled Frog Jumping Algorithm [18], Binary Black Hole Algorithms [19]. Previous work [20] propose the use of binarization techniques in order to improve the solutions of combinatorial problems like the SCP, but they lack explanation on how the binarization affects the metaheuristics, also the previous work don't propose a pre-processing phase to reduce the computational cost of the instances of the SCP.

In this paper, we present a new approach to solving the SCP based on Wen Tsao-Pan's Fruit Fly Swarm Algorithm (FFSA) [21]. This metaheuristic is based on the foraging behavior of fruit flies, which use the senses of smell and vision to find their food; in terms of the algorithm, these senses are represented by a combination of local (smell) and global (vision) searches to improve the quality of solutions. Since FFSA was developed for continuous spaces and SCP is a binary problem, our work contributes to propose several binarization methods for a continuous algorithm in order to promote a better distribution between exploration/exploitation. In order to achieve this balance, we present eight different transfer functions and five discretization methods, generating a total of 39 variations to the original BFFSA. The results of this work suggests that BFFSA (the binary version of FFSA) is a robust algorithm, capable to produce good results at a low computational cost.

This article is organized as follows: A brief description of the Set Coverage Problem in Section 2, the presentation of Pan's Fruit Fly Algorithm in Section 3, an adaptation of Pan's metaheuristic to work in a binary search space in Section 4. Our proposal with the description of the functions and methods used to allow the algorithm to run in discrete spaces in Section 5. Finally, we present our results, conclusions, and possible future lines of research in Sections 6 and 7.

2 Set Covering Problem

The SCP is a classical covering problem and is defined as a binary matrix A where $a_{i,j} \in \{0,1\}$ is the value of each cell in the matrix and i, j are the size of m-rows and n-columns, respectively:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \cdots & \cdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \tag{1}$$

Defining the column j satisfies a row i if a_{ij} is equal to 1 and this will be the contrary case if this is 0. In addition, it has an associated cost $c \in C$, where $C = \{c_1, c_2, \ldots, c_n\}$ together with

$i \in \{1, 2, \ldots, m\}$ and $j \in \{1, 2, \ldots, n\}$ are the sets of rows and columns, respectively. The problem results in the following objective: to minimize the cost of the subset $S \subseteq J$, with the constraint that all rows $i \in I$ are covered by at least one column $j \in J$. It is taken into consideration that when the column j is in the subset of solution S, this is equal to 1 and 0 otherwise. The SCP can be defined as the following:

$$Minimize\ Z = \sum_{j=1}^{n} c_j x_j \tag{2}$$

Subject to Eqs. (3) and (4):

$$\sum_{j=1}^{n} a_{i,j} x_j \geq 1\ \forall i \in I \tag{3}$$

$$x_j \in \{0, 1\}\ \forall j \in J \tag{4}$$

One of the many practical applications of this problem is the location of fire stations. Lets consider a city divided in a finite number of areas which need to locate and build fire stations. Each one of these areas need to be covered by at least one station, but a single fire station can only bring coverage to its own area and the adjacent ones; also, the problem requires that the number of stations to build needs to be the minimum.

Intentionally, we have selected an instance of SCP with $m = 11$ and $n = 11$ to represent it graphically in Fig. 1 and by Eqs. (5)–(16). When a SCP formulation has a constant cost (a value of 1 in this case), we will refer to it as an *Unicost* SCP instance.

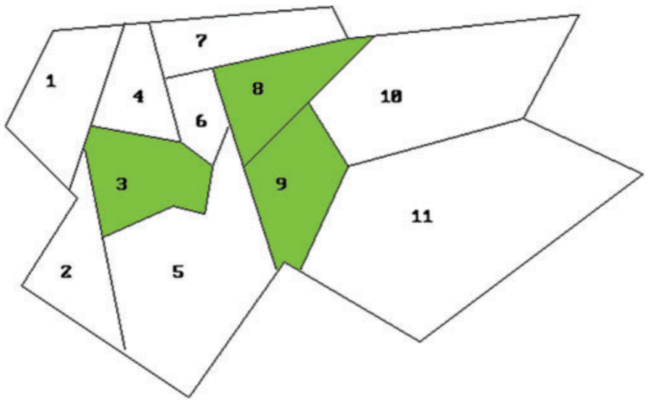

Figure 1: Solution to the practical example of SCP

$$Minimize\ Z = \sum_{j=1}^{11} c_j x_j \tag{5}$$

Subject to:

$$AREA_1 = x_1 + x_2 + x_3 + x_4 \geq 1 \tag{6}$$

$$AREA_2 = x_1 + x_2 + x_3 + x_5 \geq 1 \quad (7)$$

$$AREA_3 = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 1 \quad (8)$$

$$AREA_4 = x_1 + x_3 + x_4 + x_6 + x_7 \geq 1 \quad (9)$$

$$AREA_5 = x_2 + x_3 + x_5 + x_6 + x_8 + x_9 \geq 1 \quad (10)$$

$$AREA_6 = x_3 + x_4 + x_5 + x_6 + x_7 + x_8 \geq 1 \quad (11)$$

$$AREA_7 = x_4 + x_6 + x_7 + x_8 \geq 1 \quad (12)$$

$$AREA_8 = x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} \geq 1 \quad (13)$$

$$AREA_9 = x_5 + x_8 + x_9 + x_{10} + x_{11} \geq 1 \quad (14)$$

$$AREA_{10} = x_8 + x_9 + x_{10} + x_{11} \geq 1 \quad (15)$$

$$AREA_{11} = x_9 + x_{10} + x_{11} \geq 1 \quad (16)$$

As the SCP is a NP-hard class problem, one of the many difficulties that benchmarks arise is their size and the computational time associated. To solve this, many authors propose to do a pre-processing of the problem before apply any exact method or metaheuristic in order to obtain instances that are equivalent to original but smaller in terms of rows and columns. In the next section, we describe the methods used in this research.

2.1 Pre-Processing

To accelerate the problem solving, we introduce a preprocessing phase before run the metaheuristic to reduce the size of instances and improve the performance of the algorithm. In this article, we use two methods that have proven to be more effective: Column Domination [22] and Column Inclusion [23].

Column Domination: It consists into deleting the redundant columns without affecting the final solution. In other words, if the rows belonging to the column j can be covered by another column with a cost lower than c_j, then the column j is *dominated* and it can be removed. This method is detailed in the Algorithm 1.

Algorithm 1: Column Domination

```
1: Order all columns by cost, ascending.
2: **if** Two or more columns have the same cost **then**
3:     Order those columns by the amount of rows $I_j$ covered by column j, descending
4:     Check if rows $I_j$ can be covered by a set of other columns with a cost lower than $c_j$
```

(Continued)

Algorithm 1: Continued

```
5:     if Cost is lower then
6:         The column $j$ is dominated and it can be removed.
7:     end if
8: end if
```

Column Inclusion: If a row is covered by only one column after the above domination, that column must be included in the optimal solution, and there is no need to evaluate its feasibility.

3 Fruit Fly Swarm Algorithm

The FFSA is a bio-inspired metaheuristic [21] based on the foraging behavior of fruit flies or vinegar flies for finding food, considering that their smell *(osphresis)* and vision senses are much better than in any other specie. The foraging behavior processes consider smell the food source, fly to it and then visualize the same food source to determine a better direction.

In Fig. 2, there is a graphical representation of these foraging search processes. Consider S_1, S_2 and S_3 as fruit flies from our population. During the smell-based search, the flies will randomly move across the search space, so their new positions will be (X_1, Y_1), (X_2, Y_2) and (X_3, Y_3) respectively; then, in the next phase, flies will be evaluated in their smell concentration (fitness function) to determine which one is the best in the group; for our example, we are using the reciprocal of distance to the origin $(1/\ Dist_i)$ as fitness function. Finally, and knowing which one is the best fruit fly, the population will move into its direction to get closer to the food source.

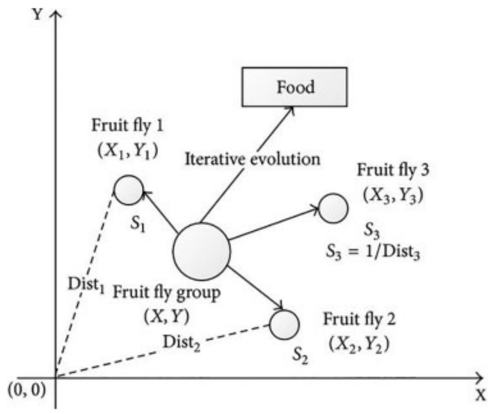

Figure 2: Food searching of a group of fruit flies

The traditional FFSA consists of 4 phases. These are: initialization, smell-based search, population evaluation, and vision-based search. In the initialization phase, parameters are set and the fruit flies (solutions) are created randomly with a very sensitive osphresis and vision organs. During the smell-based search phase, flies use their senses to feel all kinds of smells in the air and fly towards the corresponding locations. Then, the flies are evaluated to find the best concentration of smell. When they are near to food, in the vision-based search phase, they fly toward the food source using their vision organ. The pseudocode of these phases is detailed in Algorithm 2.

Algorithm 2: Fruit Fly Swarm

```
1: Initialization
2: Initialize population size ($N$)
3: Initialize generation max ($gen$)
4: for $i = 1$ to $N$ do
5:     Create randomly $F_i$, the $i$-$th$ fruit fly
6: end for
7: for $t = 1$ to $gen$ do
8:     Smell-based search
9:     Emulate the smell sense by modifying population with random values
10:      $F_i = F_i + random\_value$
11:      Population evaluation
12:      Evaluate solutions fitness using the objective function.
13:      Vision-based search.
14:      $BF = Best\ fruit\ fly$
15:      for $i = 1$ to N do
16:          $F_i = (F_i + BF)/2$
17:      end for
18: end for
```

The FFSA has been successfully used to solve continuous problems such as: the financial distress [21], web auction logistics service [24], power load forecasting [25], design of key control characteristics for automobile parts [26] and distribution of pollution particles [27].

4 Binary-Fruit Fly Swarm Algorithm

In contrast with traditional FFSA, the BFFSA [28] uses a discrete binary string (Fig. 3) to represent a solution and a probability vector to generates the population (Fig. 4); then, the value of each bit of the fruit flies goes from zero to one (and vice versa) to exploit the neighborhood in the smell-based search process and perform a global vision-based search to improve the exploration ability. This new algorithm, detailed later in pseudocode (Algorithm 3), preserves the four phases but adds three search methods: Smell-based, Local-Vision-based and Global-Vision-based. Also, these methods will add new parameters to perform searches; all of them are detailed in Tab. 1.

0	1	0	1	1	...	1	1	0	1

Figure 3: Representation of a fruit fly (solution) in BFFSA

$p^1(t)$	$p^2(t)$	$p^3(t)$	...	$p^{n-1}(t)$	$p^n(t)$

Figure 4: Representation of the probability vector in BFFSA

Algorithm 3: Binary Fruit Fly Swarm Algorithms

```
1: Initialization Phase
2: Initialize parameter values of $N$, $gen$, $S$, $L$ and $b$
3: Initialize probability vector $p(t = 0)$
```

(Continued)

Algorithm 3: Continued

```
4: for $i = 1$ to $N$ do
5:     for $d = 1$ to $n$ do
6:         Create randomly the $F^d$ bit
7:     end for
8: end for
9: for $t = 1$ to $gen$ do
10:     Smell-based Search
11:     for $i = 1$ to $N$ do
12:         for $s = 1$ to $S$ do
13:             Create the $F_{i,s}$ neighbor, flipping $L$ bits around $F_i$
14:         end for
15:     end for
16:     Apply the repair operator
17:     Population Evaluation Phase
18:     Evaluate solution fitness using the objective function
19:     Local-Vision-based Search
20:     for $i = 1$ to $N$ do
21:         Find the best neighbor $F_{i,best}$ for $F_i$
22:         Make the neighborhood fly towards $F_{i,best}$
23:     end for
24:     Global-Vision-based Search
25:     Find the best fruit fly in the population, $F_{best}$
26:     Select randomly two flies $F_1$ and $F_2$
27:     Update probability vector $p(t)$
28:     for $i = 1$ to $N$ do
29:         Create $F_i$ according to $p(t)$
30:     end for
31: end for
```

Table 1: BFFSA parameters

Parameters	Detail
N	Population size.
Gen	Generations (iterations).
S	Neighbors to create during smell-based search.
L	Bits to flip randomly when generating neighbors.
B	Coefficient of vision sensitivity.

This article proposes and evaluates new instances for BFFSA, created from the combination of the original binary algorithm, eight transfer functions and two discrete methods, in order to improve solutions.

4.1 Initialization

In the BFFSA, each fruit fly is a solution represented by a n-bit binary vector, where n is the number of columns in the instance to solve. Thus, in a fruit fly F_i, the value F^d represents the d^{th} binary decision bit, $d \in [1, n]$. All fruit flies are generated by an n-dimensional probability vector $p(t)$, where t represents generation (or iteration) with $t \in [1, gen]$. Then, the $p^d_{(t)}$ is the probability in the d^{th} dimension of the fruit fly F_i to be 1 during generation t. The pseudocode for this phase is detailed in Algorithm 4.

Algorithm 4: Initial population in BFFSA

```
1: for $i = 1$ to $N$ do
2:     for $d = 1$ to $n$ do
3:         if $rand() < p^d(0)$ then
4:             $F^d = 1$
5:         else
6:             $F^d = 0$
7:         end if
8:     end for
9: end for
```

To generate a uniformly distributed population in the search space, the probability vector must be $p(0) = [0.5, 0.5, \ldots, 0.5]$, so all columns have fifty percent probability of being selected. In the next generation, a new population with N fruit flies will be generated using this probability vector.

4.2 Smell-Based Search

In this phase, we create S neighbors randomly around each fruit fly F_i using the smell-based search. Each one of these neighbors are generated using the following method: first, we select randomly L-bits, and then we flip these L columns values to the opposite binary value. For example, if we have a 9-bit fruit fly and $L = 3$, the smell-based search may produce a neighbor like the one represented in Fig. 5.

0	**1**	0	1	1	**0**	**0**	1	1

$$\Downarrow$$

0	**0**	0	1	1	**1**	**1**	1	1

Figure 5: Creation of a neighbor during smell-based search

At this point, a population with $(N \cdot S)-$ fruit flies is evaluated using the objective function. In case to get unfeasible solutions, we apply a repair operator. This additional phase will be explained later (SubSection 4.5).

4.3 Local-Vision-Based Search

Once all solutions in the neighborhood are feasible, the fruit flies are evaluated with the vision sense (the objective function) to find the best local neighbor and fly towards it. If a better neighbor is found, then the whole neighborhood will fly towards it and this recently discovered "local best" fruit fly will replace the previous solution; otherwise, solution will remain the same.

4.4 Global-Vision-Based Search

This search works on the exploration ability (Eqs. (17) and (18)), considering that previous phases are more focused into the exploitation ability. To update the next fruit flies generation, this phase updates the probability vector with the differential information between the best fruit fly F_{best} and two random fruit flies (F1 and F2) to set a coefficient for the vision sensitivity b to enhance the exploration.

$$\Delta^d(t+1) = F^d_{best} + 0.5(F^d_1 - F^d_2) \tag{17}$$

$$p^d(t+1) = \frac{1}{(1 + e^{-b(\Delta^{\mathrm{d}}(t+1)-0.5)}} \tag{18}$$

As we can see in the Eq. (17), the algorithm have a high probability of exploration in the first steps of the search, because the two random fruit flies tend to be far away one of the other, but always with the guide of the best fruit fly F_{best}. Once the flies are stuck on close positions, they tend to perform more exploitation with the smell-based search and the local-vision-based search.

4.5 Repair Operator

A common issue with metaheuristics is the generation of unfeasible solutions during an iteration. For the SCP, this means that some individuals will not cover all rows and a subset of constraints may be violated. To solve this issue, the algorithm implements a repair operator to make all individuals feasible and eliminate redundancy. The method described in [29] calculates a ratio between the cost of an uncovered column (c_j) and the number of uncovered rows covered by that column; once all rows are covered and the solution is feasible, the operator includes an optimization step to eliminate any redundant column (Algorithm 5).

Algorithm 5: Repair Operator

```
1: $I$ = The set of all rows;
2: $J$ = The set of all columns;
3: $\alpha_i$ = The set of columns that cover row $i, i \in I$;
4: $\beta_j$ = The set of rows covered by column $j,\ j \in J$;
5: $K$ = The set of columns in a solution;
6: $w_i$ = The number of columns that cover row $i,\ i \in I$. For this, $w_i = |S \cap^{\alpha_i}|, \forall i \in I$;
7: $U$ = The set of uncovered rows. For this, $U = \{i|w_i = 0, \forall i \in I\}$;
8: for row $i \in U$ (in increasing order of i) do
9:     Find the first column $j$ in increasing order of $j \in \alpha_i$ that minimizes $\frac{c_j}{|U \cap b_j|}$;
10:    Add $j$ to $K$ and set $w_i = w_i + 1, \forall i \in b_j$;
11:    Set $U = U - b_j$;
12: end for
13: for column $j \in K$ (in decreasing order of $j$) do
14:    if $w_i \geq 2$ then
15:        $K = Kj$;
16:        $w_i = w_i - 1,\ \forall i \in \beta_j$;
17:    end if
18: end for
```

(Continued)

Algorithm 5: Continued

```
19: $K$ is now a feasible solution for the SCP that contains no redundant columns;
18: end for
19: $K$ is now a feasible solution for the SCP that contains no redundant columns;
```

5 Proposed Binarization Methods for the BFFSA

In this article, we propose to modify the original BFFSA with a two-step binarization technique (Fig. 6), which will transform the solution from $\mathfrak{R}$ to an "InterSpace" (in $\mathbb{Z}$) and then to the binary space. Following a procedure similar to the one proposed in [30,31], we will replace the equation for global searching (Eq. (18)) with one of the eight transfer functions (Eqs. (19)–(26)) showed in the Tab. 2. Specifically, our idea is to replace the calculation for the differential information $b(\Delta_i^d - 0.5)$, with one of these eight transfer functions in order to define the probability to move an element of the solution from 1 to 0 (or vice versa), forcing the fruit flies to be in the interval [0, 1]. With this change, we force to have a controlled balance between exploration and exploitation in all the search steps of the algorithm. Thus, we promote the search of new areas meanwhile we search better solution in known promising areas.

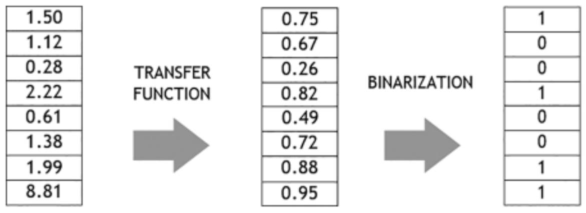

Figure 6: Classic binarization scheme

Table 2: First step-transfer functions

Type	Mathematical formula	Equations
S1 [32,33]	$PS_1(\Delta^d(t+1)) = \frac{1}{1+e^{-2b(\Delta^d(t+1)-0.5)}}$	(19)
S2 [33,34]	$PS_2(\Delta^d(t+1)) = \frac{1}{1+e^{-b(\Delta^d(t+1)-0.5)}}$	(20)
S3 [32,33]	$PS_3(\Delta^d(t+1)) = \frac{1}{1+e^{\frac{-b(\Delta^d(t+1)-0.5)}{2}}}$	(21)
S4 [32,33]	$PS_4(\Delta^d(t+1)) = \frac{1}{1+e^{\frac{-b(\Delta^d(t+1)-0.5)}{3}}}$	(22)
V1 [33,35]	$PV_1(\Delta^d(t+1)) = \left\lvert \operatorname{erf}\left(\frac{\sqrt{\pi}}{2}(b(\Delta^d(t+1)-0.5))\right) \right\rvert$	(23)

(Continued)

Table 2: Continued

Type	Mathematical formula	Equations
V2 [33,35]	$PV_2(\Delta^d(t+1)) = \lvert\tanh(b((\Delta^d(t+1) - 0.5)))\,\rvert$	(24)
V3 [32,33]	$PV_3(\Delta^d(t+1)) = \left\lvert \frac{\Delta^d(t+1)}{\sqrt{1+(b(\Delta^d(t+1)-0.5))^2}} \right\rvert$	(25)
V4 [32,33]	$PV_4(\Delta^d(t+1)) = \left\lvert \frac{2}{\pi}\arctan\left(\frac{\pi}{2}(b(\Delta^d(t+1) - 0.5))\right)\right\rvert$	(26)

It is important to note that of the *S-shaped* (left-hand in Fig. 7) and *V-shaped* (right-hand in Fig. 7) functions presented here, the original BFFSA uses the transfer function PS_2 with a standard discretization method. In this paper, we test a universe of 40 different instances of the algorithm, where 39 of the 40 are new variations realized by our research.

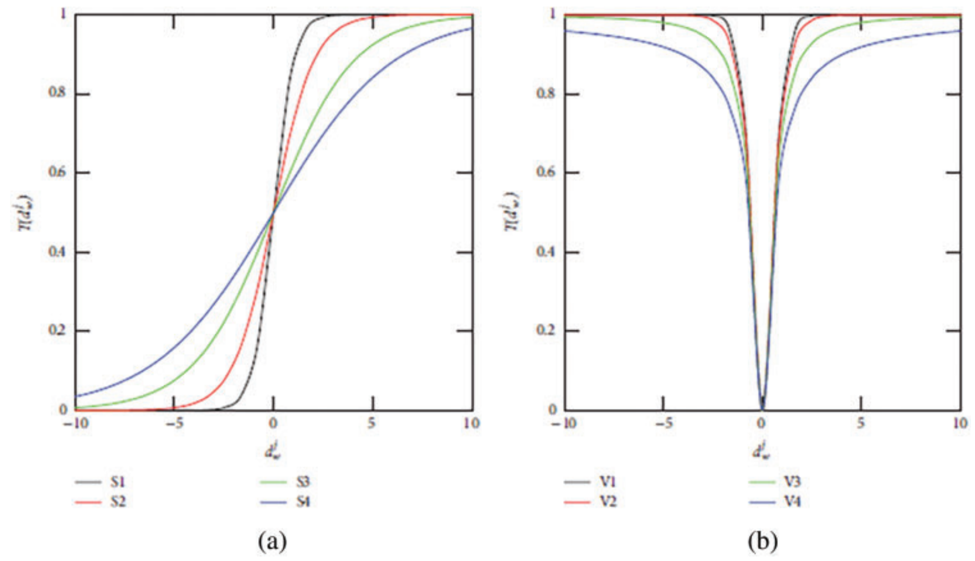

Figure 7: (a) S and (b) V transfer functions

After updating the probability vector with one of these S-shaped or V-shaped transfer functions, an element of a fruit fly will be updated using one of the following discretization methods: *Standard*, *Complement*, *Static Probability*, *Elitist* and *Elitist Roulette*, detailed in Tab. 3 with the Eqs. (27)–(31), respectively. In all of them, F_d represents the d_{th} position of the fruit fly F_i, F_{best} is the best fruit fly in the current generation and α is the static probability.

6 Experiment Results

The modified BFFSA with the transfer functions proposed has been implemented in Java in a Common KVM processor of 2.66 GHz with 4 GB RAM computer, running Microsoft Windows 7. The parameter tuning for the algorithm is detailed in Tab. 4.

Table 3: Second step-techniques of binarization

Type	Binarization	Equations
Standard	$F_i^d(t+1) = \begin{cases} x_w^j, & \textit{if rand} \le p^d(t+1) \\ 0, & \textit{else} \end{cases}$	(27)
Complement	$F_i^d(t+1) = \begin{cases} x_w^j, & \textit{if rand} \le p^d(t+1) \\ 0, & \textit{else} \end{cases}$	(28)
Static probability	$F_i^d(t+1) = \begin{cases} 0, & \textit{if } p^d(t+1) \le a \\ F_i^d(t), & \textit{if } a < p^d(t+1) \le \frac{1}{2}(1+a) \\ 1, & \textit{if } p^d(t+1) \ge \frac{1}{2}(1+a) \end{cases}$	(29)
Elitist	$F_i^d(t+1) = \begin{cases} F_{best}^d, & \textit{if rand} < p^d(t+1) \\ 0, & \textit{else} \end{cases}$	(30)
Elitist roulette	$F_i^d(t+1) = \begin{cases} P[p^d(t+1) = \zeta_j] = \frac{f(\zeta)}{\sum_{\delta \in Q_g} f(\delta)}, & \textit{if rand} \le p^d(t+1) \\ P[p^d(t+1) = 0] = 1, & \textit{else} \end{cases}$	(31)

Table 4: Parameter tuning for BFFSA experiments

Parameters	Detail	Value
N	Population size.	50
Gen	Generations (iterations).	400
S	Neighbors to create during smell-based search.	5
L	Bits to flip randomly when generating neighbors.	3
b	Coefficient of vision sensitivity.	15
α	Static probability	0.2
k	Number of best individuals for Elitist Roulette method	3

All the datasets tested are from Beasley's OR Library 3. In total, we solved 65 data files; instances 4, 5, 6 are from [36], instances A, B, C, D are from [22] and instances NRE, NRF, NRG, NRH are the unknown-solution problem set from [37]. Details of datasets are described in Tab. 5.

For each instance, we report the average values obtained after run 30 times each algorithm.

Table 5: Set covering instances

Instance set	Number of instances	m	n	Cost range	Density (%)	Optimal solution
4	10	200	1000	[1,100]	2	Known
5	10	200	2000	[1,100]	2	Known
6	5	200	1000	[1,100]	5	Known
A	5	300	3000	[1,100]	2	Known
B	5	300	3000	[1,100]	5	Known
C	5	400	4000	[1,100]	2	Known
D	5	400	4000	[1,100]	5	Known
NRE	5	500	5000	[1,100]	10	Unknown
NRF	5	500	5000	[1,100]	20	Unknown
NRG	5	1000	10000	[1,100]	2	Unknown
NRH	5	1000	10000	[1,100]	5	Unknown

6.1 Comparison of Proposed BFFSA with Other Metaheuristics

The Tabs. 6–13 show the detailed results obtained by different instances of our modified BFFSA. In all of them, the results are presented along with the transfer function (TF) and discretization method (DM) used in each case, and compared with other metaheuristics in terms of minimum and maximum number of optimal founded (Z_{MIN}, Z_{MAX}) and the relative percentage deviation (RPD), which represents the deviation of the objective value Z (fitness) from Z_{OPT} (Eq. (32)).

$$RPD = \frac{100(Z_{MIN} - Z_{OPT})}{Z_{OPT}} \tag{32}$$

Table 6: Computational results for instance set 4

Instance		4.1	4.2	4.3	4.4	4.5	4.6	4.7	4.8	4.9	4.10
Z_{opt}		429	512	516	494	512	560	430	492	641	514
Our approach											
BFFSA	Z_{min}	**429**	**512**	**516**	**494**	**512**	**560**	**430**	**492**	**641**	**514**
	Z_{avg}	**431.57**	**512**	**516**	**495.53**	**514.2**	**560.87**	**430.67**	**494.2**	**646.83**	**514.1**
	RPD	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	TF	S2	S4	S4	S4	S4	S3	S3	S4	V4	S3
	DM	STD	STD	ELT	ELT	STD	STD	STD	STD	ELT	STD
Other Approaches											
BCS	Z_{min}	430	512	517	494	512	560	430	492	641	514
	Z_{avg}	432	516	519	503	516	563	431	495	645	526
	RPD	0.23	0	0.19	0	0	0	0	0	0	0

(Continued)

Table 6: Continued

BBH	Z_{min}	430	512	516	495	514	560	430	493	644	514
	Z_{avg}	430	512	517	501	519	562	432	495	648	517
	RPD	0.23	0	0	0.20	0.39	0	0	0.2	0.46	0
BCSO	Z_{min}	459	570	590	547	545	560	430	493	644	514
	Z_{avg}	480	594	607	578	554	562	432	495	648	517
	RPD	7	11.3	14.3	10.7	6.40	0	0	0.2	0.46	0
BFO	Z_{min}	429	517	519	495	514	563	430	497	655	519
	Z_{avg}	430	517	522	497	515	565	430	499	658	523
	RPD	0	0.97	0.58	0.2	0.39	0.53	0	1.01	2.18	0.97
BSFLA	Z_{min}	430	516	520	501	514	563	431	497	656	518
	Z_{avg}	430	518	520	504	514	563	432	499	656	519
	RPD	0.23	0.78	0.78	1.42	0.39	0.54	0.23	1.02	2.34	0.78
BELA	Z_{min}	447	559	537	527	527	607	448	509	682	571
	Z_{avg}	448	559	539	530	529	608	449	512	682	571
	RPD	4.20	9.18	4.07	6.68	2.93	8.39	4.19	3.46	6.40	11.09
BABC	Z_{min}	430	513	519	495	514	561	431	493	649	571
	Z_{avg}	430	513	521	496	517	565	434	494	651	519
	RPD	0.23	0.20	0.58	0.20	0.39	0.18	0.23	0.20	0.93	0.58

Table 7: Computational results for instance set 5

Instance		5.1	5.2	5.3	5.4	5.5	5.6	5.7	5.8	5.9	5.10
Z_{opt}		253	302	226	242	211	213	293	288	279	265
Our approach											
BFFSA	Z_{min}	**253**	304	**226**	**242**	**211**	**213**	**293**	**288**	**279**	**265**
	Z_{avg}	**255.6**	305.67	**227.73**	**242.03**	**211**	**213.5**	**294.03**	**288.87**	**279.8**	**265.07**
	RPD	**0**	0.66	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	TF	S3	S4	S3	S2	V4	V4	S4	S3	S4	S4
	DM	STD	STD	STD	COMP	COMP	COMP	ELT	STD	STD	STD
Other approaches											
BCS	Z_{min}	253	304	226	242	212	213	293	288	279	265
	Z_{avg}	256	307	227	243	213	215	294	290	280	266
	RPD	0	0.66	0	0	0.47	0	0	0	0	0
BBH	Z_{min}	253	305	228	242	211	213	293	288	279	265
	Z_{avg}	256	307	230	242	211	213	294	289	280	267
	RPD	0	0.99	0.88	0	0	0	0	0	0	0

(Continued)

Table 7: Continued

BCSO	Z_{min}	279	339	247	251	230	232	332	320	295	285
	Z_{avg}	287	340	251	253	230	243	338	330	297	287
	RPD	10.3	12.3	9.3	3.7	9	8.9	13.3	11.1	5.7	7.5
BFO	Z_{min}	257	309	229	242	211	213	298	291	284	268
	Z_{avg}	260	311	233	242	213	213	301	292	284	270
	RPD	1.58	2.31	1.32	0	0	0	1.70	1.04	1.79	1.13
BSFLA	Z_{min}	254	307	228	242	211	213	297	291	281	265
	Z_{avg}	255	307	230	242	213	214	299	293	283	266
	RPD	0.40	1.66	0.88	0	0	0	1.37	1.04	0.72	0
BELA	Z_{min}	280	318	242	251	225	247	316	315	314	280
	Z_{avg}	281	321	242	252	227	248	317	317	315	281
	RPD	10.67	5.30	7.08	3.72	6.64	15.96	7.85	9.38	12.54	5.66
BABC	Z_{min}	254	309	229	242	211	214	298	289	280	267
	Z_{avg}	255	309	233	245	212	214	301	291	281	270
	RPD	0.40	2.32	1.33	0	0	0.47	1.71	0.35	0.36	0.75

Table 8: Computational results for instance set 6 and A

Instance		6.1	6.2	6.3	6.4	6.5	A.1	A.2	A.3	A.4	A.5
Z_{opt}		138	146	145	131	161	253	252	232	234	236
Our approach											
BFFSA	Z_{min}	**138**	**146**	**145**	**131**	**161**	**253**	254	233	**234**	**236**
	Z_{avg}	**140.07**	**148.93**	**146.70**	**131**	**162.30**	**254.80**	258.90	234.80	**234.77**	**236.40**
	RPD	**0**	**0**	**0**	**0**	**0**	**0**	0.79	0.43	**0**	**0**
	TF	S2	S3	S4	S3	S3	S1	S4	S3	S3	V4
	DM	COMP	STD	ELT	STD	STD	COMP	STD	STD	ELT	ELT
Other approaches											
BCS	Z_{min}	140	146	145	131	161	254	256	233	237	236
	Z_{avg}	141	147	146	133	163	254	257	235	239	237
	RPD	0.14	0	0	0	0	0.34	0.16	0.43	0.13	0
BBH	Z_{min}	140	147	145	131	161	253	253	233	234	236
	Z_{avg}	142	150	147	131	164	255	254	234	234	237
	RPD	1.45	0.68	0	0	0	0	0.39	0.43	0	0
BCSO	Z_{min}	151	152	160	138	169	286	274	257	248	244
	Z_{avg}	160	157	164	142	173	287	276	263	251	244
	RPD	9.40	4.10	10.3	5.30	5	13	8.70	10.80	6	3

(Continued)

Table 8: Continued

BFO	Z_{min}	138	147	147	131	164	255	259	238	235	236
	Z_{avg}	140	149	150	131	157	256	261	240	237	237
	RPD	0	0.68	1.37	0	1.86	0.79	2.77	2.58	0.42	0
BSFLA	Z_{min}	140	147	147	131	166	255	260	237	235	236
	Z_{avg}	141	147	148	133	169	258	260	239	238	239
	RPD	10.14	9.59	10.34	6.87	14.29	3.16	10.71	8.62	6.84	2.12
BELA	Z_{min}	152	160	160	140	184	261	279	252	250	241
	Z_{avg}	152	161	163	142	187	264	281	253	252	243
	RPD	10.14	9.59	10.34	6.87	14.29	3.16	10.71	8.62	6.84	2.12
BABC	Z_{min}	142	147	148	131	165	254	257	235	236	236
	Z_{avg}	143	150	149	133	167	254	259	238	237	238
	RPD	2.90	0.68	2.07	0	2.48	0.40	1.98	1.29	0.85	0

Table 9: Computational results for instance set B and C

Instance		B.1	B.2	B.3	B.4	B.5	C.1	C.2	C.3	C.4	C.5
Z_{opt}		69	76	80	79	72	227	219	243	219	215
Our approach											
BFFSA	Z_{min}	**69**	**76**	**80**	**79**	**72**	**227**	**219**	247	**219**	**215**
	Z_{avg}	**70.67**	**76.27**	**80.17**	**80.1**	**72**	**230.77**	**221.57**	254.27	**223.07**	**216.8**
	RPD	**0**	**0**	**0**	**0**	**0**	**0**	**0**	1.65	**0**	**0**
	TF	S2	S3	S1	V3	S1	V3	S4	S3	S3	V4
	DM	COMP	ELT	COMP	COMP	COMP	COMP	ELT	STD	ELT	ELT
Other approaches											
BCS	Z_{min}	69	76	80	79	72	228	221	247	221	216
	Z_{avg}	70	79	80	81	73	230	223	249	223	217
	RPD	0	0	0	0	0	0.44	0.9	1.62	0.9	0.46
BBH	Z_{min}	69	76	80	79	72	229	219	245	219	215
	Z_{avg}	70	77	81	81	73	231	220	246	219	216
	RPD	0	0	0	0	0	0.88	0	0.82	0	0
BCSO	Z_{min}	79	86	85	89	73	242	240	277	250	243
	Z_{avg}	79	89	85	89	73	242	241	278	250	244
	RPD	14.5	13.2	6.3	12.7	1.4	6.6	9.6	14	12.3	13
BFO	Z_{min}	71	78	80	80	72	230	223	253	227	217
	Z_{avg}	72	78	80	81	73	232	225	253	228	219
	RPD	2.89	2.63	0	1.26	0	1.32	1.83	4.12	3.65	0.93

(Continued)

Table 9: Continued

BSFLA	Z_{min}	70	76	80	79	72	229	223	253	227	217
	Z_{avg}	70	77	80	80	73	231	225	253	228	218
	RPD	1.45	0	0	0	0	0.88	1.83	4.12	3.65	0.93
BELA	Z_{min}	86	88	85	84	78	237	237	271	246	224
	Z_{avg}	87	88	87	88	81	238	239	271	248	225
	RPD	24.64	15.79	6.25	6.33	8.33	4.41	8.22	11.52	12.33	4.19
BABC	Z_{min}	70	78	80	80	72	231	222	254	231	216
	Z_{avg}	70	79	80	81	74	233	223	255	233	217
	RPD	1.45	2.63	0	1.27	0	1.76	1.37	4.53	5.48	0.47

Table 10: Computational results for instance set D

Instance		D.1	D.2	D.3	D.4	D.5
Z_{opt}		60	66	72	62	61
Our approach						
BFFSA	Z_{min}	**60**	67	73	**62**	**61**
	Z_{avg}	**60**	67.73	75.7	**63.37**	**62.63**
	RPD	**0**	1.52	1.39	**0**	**0**
	TF	S1	S3	S4	S2	S3
	DM	ERLT	ELT	ELT	COMP	ELT
Other approaches						
BCS	Z_{min}	60	66	73	62	61
	Z_{avg}	60	66	74	62	62
	RPD	0	0	0.14	0	0
BBH	Z_{min}	60	67	73	62	61
	Z_{avg}	60	68	74	62	62
	RPD	0	1.51	138	0	0
BCSO	Z_{min}	65	70	79	64	65
	Z_{avg}	66	70	81	67	66
	RPD	8.3	6.1	9.7	3.2	6.6
BFO	Z_{min}	60	68	75	62	63
	Z_{avg}	61	68	77	62	63
	RPD	0	3.03	4.16	0	3.27
BSFLA	Z_{min}	60	67	75	63	63
	Z_{avg}	62	68	77	63	66
	RPD	0	1.52	4.17	1.61	3.28

(Continued)

Table 10: Continued

BELA	Z_{min}	62	73	79	67	66
	Z_{avg}	62	74	81	69	67
	RPD	3.33	10.61	9.72	8.06	8.20
BABC	Z_{min}	60	68	76	63	63
	Z_{avg}	61	68	77	63	66
	RPD	0	3.03	5.56	1.61	3.28

Table 11: Computational results for instance set NRE and NRF

Instance		NRE.1	NRE.2	NRE.3	NRE.4	NRE.5	NRF.1	NRF.2	NRF.3	NRF.4	NRF.5
Z_{opt}		29	30	27	28	28	14	15	14	14	13
Our approach											
BFFSA	Z_{min}	**29**	**30**	28	29	**28**	**14**	**15**	15	15	14
	Z_{avg}	**29**	**32.13**	28.7	29.63	**28.93**	**15**	**15.9**	16.73	15.03	15.1
	RPD	**0**	**0**	3.7	3.57	**0**	**0**	**0**	7.14	7.14	7.69
	TF	**S3**	**S3**	S4	S4	**V4**	**V4**	**S4**	S4	V1	V3
	DM	**ELT**	**ELT**	ELT	ELT	**ELT**	**ELT**	**ELT**	ELT	ELT	ELT
Other approaches											
BCS	Z_{min}	29	31	28	30	28	14	15	16	15	14
	Z_{avg}	30	32	29	31	29	14	16	16	16	15
	RPD	0	0.32	0.36	0.67	0	0	0	4.28	7.14	7.69
BBH	Z_{min}	29	31	28	29	28	14	15	16	15	14
	Z_{avg}	30	34	32	33	29	15	16	16	16	15
	RPD	0	3.33	3.7	3.57	0	0	0	4.28	7.14	7.69
BCSO	Z_{min}	29	34	31	32	30	17	18	17	17	15
	Z_{avg}	30	34	32	33	30	17	18	17	17	16
	RPD	0	13.3	14.8	14.3	7.1	21.4	20	21.4	21.4	15.4
BFO	Z_{min}	29	32	29	29	29	15	16	16	15	15
	Z_{avg}	31	32	30	31	29	17	16	17	18	19
	RPD	0	6.66	7.4	3.57	3.57	7.14	6.66	14.28	7.14	15.38
BSFLA	Z_{min}	29	31	28	29	28	15	15	16	15	15
	Z_{avg}	29	32	28	30	31	15	15	17	16	17
	RPD	0	3.33	3.7	3.57	0	7.14	0	14.29	7.14	23.08
BELA	Z_{min}	30	35	34	33	30	17	18	17	17	16
	Z_{avg}	31	35	34	34	31	17	18	18	19	17
	RPD	3.45	16.67	25.93	17.86	7.14	21.43	20	21.43	21.43	23.08

(Continued)

Table 11: Continued

BABC	Z_{min}	29	32	29	29	29	14	16	16	15	15
	Z_{avg}	33	32	31	30	32	15	16	17	17	16
	RPD	0	6.67	7.41	3.57	3.57	0	6.67	14.29	7.14	15.38

Table 12: Computational results for instance set NRG

Instance		NRG.1	NRG.2	NRG.3	NRG.4	NRG.5
Z_{opt}		176	154	166	168	168
Our approach						
BFFSA	Z_{min}	178	159	170	170	173
	Z_{avg}	180.3	160.43	171.57	172.2	175
	RPD	1.14	3.25	2.41	1.19	2.98
	TF	S4	V4	S4	V4	S4
	DM	ELT	ELT	ELT	ELT	ELT
Other approaches						
BCS	Z_{min}	176	156	169	170	170
	Z_{avg}	177	157	170	171	171
	RPD	0	0.13	1.8	1.19	0.12
BBH	Z_{min}	179	158	169	170	168
	Z_{avg}	181	160	169	171	169
	RPD	1.7	2.59	1.8	1.19	0
BCSO	Z_{min}	190	165	187	179	181
	Z_{avg}	193	166	188	183	184
	RPD	8	7.1	20.6	6.5	7.7
BFO	Z_{min}	185	161	175	176	177
	Z_{avg}	191	163	177	176	181
	RPD	5.11	4.54	5.42	4.76	5.35
BSFLA	Z_{min}	182	161	173	173	174
	Z_{avg}	183	161	174	177	174
	RPD	3.41	4.55	4.22	2.98	3.57
BELA	Z_{min}	194	176	184	196	198
	Z_{avg}	196	176	185	197	199
	RPD	10.23	14.29	10.84	16.67	17.86
BABC	Z_{min}	183	162	174	175	179
	Z_{avg}	184	163	175	177	181
	RPD	3.98	5.19	4.82	4.17	6.55

For comparison purposes, we consider the values reported in [16] for Binary Cuckoo Search (BCS) and Binary Black Hole (BBH); also, we have taken results for Binary Cat Swarm Optimization (BCSO) [15], Binary Firefly Optimization (BFO) [13], Binary Shuffled Frog Leap Algorithm (BSFLA) [18], Binary Electromagnetism-like Algorithm (BELA) [38] and Binary Artificial Bee Colony (BABC) [39].

Table 13: Computational results for instance set NRH

Instance		NRH.1	NRH.2	NRH.3	NRH.4	NRH.5
	Z_{opt}	63	63	59	58	55
Our approach						
BFFSA	Z_{min}	66	66	61	63	**55**
	Z_{avg}	67.47	66	63	63.5	**58.07**
	RPD	4.76	4.76	3.39	3.39	**0**
	TF	S3	S3	S4	S3	S4
	DM	ELT	ELT	ELT	ELT	ELT
Other approaches						
BCS	Z_{min}	64	64	62	59	56
	Z_{avg}	64	64	63	60	57
	RPD	0.16	0.16	10.16	6.77	7.27
BBH	Z_{min}	66	67	65	63	62
	Z_{avg}	67	68	65	64	62
	RPD	4.76	6.34	10.16	8.62	12.72
BCSO	Z_{min}	70	67	68	66	61
	Z_{avg}	71	67	70	67	62
	RPD	11.1	6.3	15.3	13.8	10.9
BFO	Z_{min}	69	66	65	63	59
	Z_{avg}	70	66	67	65	60
	RPD	9.52	4.76	10.16	6.77	7.27
BSFLA	Z_{min}	68	66	62	63	59
	Z_{avg}	69	66	63	64	61
	RPD	7.94	4.76	5.08	8.62	7.27
BELA	Z_{min}	70	71	68	70	69
	Z_{avg}	71	71	70	72	69
	RPD	11.11	12.70	15.25	20.69	25.45
BABC	Z_{min}	70	69	66	64	60
	Z_{avg}	71	72	67	64	61
	RPD	11.11	9.52	11.86	10.34	9.09

Tab. 6 presents the results obtained from instance set 4. in this case our algorithm was better to all others in comparison, as it reached optimal values in all instances; BCSO, BSFLA, BELA and BABC are unable to achieve optimal values and BFO reached only two. The closest methods in comparison were BCS with eight optimal and BBH with five.

Tab. 7 describes the results from instance set 5. Once again, our algorithm got the best results along with BCS and BBH. Algorithms BCSO and BELA are unable to solve optimally any instance, BABC found only two optimal values, BFO reached three and BSFLA got four.

Tab. 8 illustrates the results from instance sets 6 and A. Our algorithm performed well, reaching eight optimal values (the whole set 6 and 3 from set A). BBH was slightly better than BCS this time, BCSO and BELA are unable to optimally solve any instance, BABC is capable to find only two optimal values (one in each set), BFO reached three and BSFLA got four.

Tab. 9 shows the results from instance set B and C. In case of set B, our algorithm had a very good performance, reaching all the optimal values, just like BCS and BBH. For instance set C, situation is similar, as BFFSA reached four out of five optimal values, outperforming all other methods.

Tab. 10 shows the results from instance set D. Here, the BFFSA and BBH (3 optimal values each one) could not reach results of BCS. However, we can still say this is an acceptable result, considering that all other approaches got less than 30% of optimal values.

For the NRE and NRF sets described in Tab. 11, only two $RPD = 0$ per set are reached by the BFFSA algorithm. Other approaches fail in general to find optimum values as the instance set becomes harder. Only BCS and BBH are closer to our results. BSFLA and BABC achieve one optimum for the instances belonging to set NRF, while BBH and BCS reach three.

Finally, for the hardest instance sets NRG and NRH (see Tabs. 12 and 13), we observe that the RPD obtained by the proposed BBFOA is good enough to compete with the approaches like BCS and BBH, as in the three cases, they could only reached one optimal value.

7 Conclusion

This article proposes several variations to BFFSA (39 to be precise), created by adding to the original BFFSA different transfer functions and discrete methods in order to improve the solutions obtained. All of these BBFOA-variations were tested into 65 SCP instances and the values reported correspond to the algorithm with the best performance. From our results, we conclude that variations presented are robust enough to compete with other algorithms as we were able to find many optimal solutions with a little parameter tuning.

We observed that best combinations of transfer functions and discretization methods depend on the instance size. For small instances (4, 5, 6, A, B, C, D) best results were achieved with transfer functions pS3 and pS4 plus the Standard discretization, whereas for huge instances (NRE, NRF, NRG, NRH) the best combinations are the same transfer functions pS3 and pS4, but with the Elitist method. A point to remark is that the use of the Elitist discretization is not exclusive for this algorithm and problem; other articles like [40] report good results with it.

In the future, we are interested in the hybridization of BFFSA with other meta-heuristics or apply an hyper-heuristics version. In the short term, we expect to test our algorithms on other SCP libraries, such like the Unicost (available at OR-Library website) or Italian railways [41] benchmarks. Due to the good results and the simplicity of this algorithm, it could be used to solve other combinatorial problems.

Acknowledgement: Marcelo Becerra-Rozas and Javier Peña are supported by Grant DI Interdisciplinary Undergraduate Research/VRIEA/PUCV/039.421/2021.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] D. Šarac, M. Kopić, K. Mostarac, M. Kujačić and B. Jovanović, "Application of set covering location problem for organizing the public postal network," *PROMET-Traffic&Transportation*, vol. 28, pp. 403–413, 2016.

[2] M. Mesquita and A. Paias, "Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem," *Computers & Operations Research*, vol. 35, pp. 1562–1575, 2008.

[3] V. Cacchiani, V. C. Hemmelmayr and F. Tricoire, "A set-covering based heuristic algorithm for the periodic vehicle routing problem," *Discrete Applied Mathematics*, vol. 163, pp. 53–64, 2014.

[4] W. A. Chaovalitwongse, T. Y. Berger-Wolf, B. Dasgupta and M. V. Ashley, "Set covering approach for reconstruction of sibling relationships," *Optimisation Methods and Software*, vol. 22, pp. 11–24, 2007.

[5] D. Tapia, B. Crawford, R. Soto, F. Cisternas-Caneo, J. Lemus-Romani *et al.,* "A Q-learning hyperheuristic binarization framework to balance exploration and exploitation," in *Applied Informatics: Third International Conference, ICAI 2020*, Ota, Nigeria, 2020.

[6] B. Crawford, R. Soto, J. Lemus-Romani, M. Becerra-Rozas, J. M. Lanza-Gutiérrez *et al.,* "Q-Learnheuristics: Towards data-driven balanced metaheuristics," *Mathematics*, vol. 9, pp. 1–26, 2021.

[7] R. Soto, G. Astorga, J. Lemus-Romani, S. Misra, M. Castillo *et al.,* "Balancing exploration-exploitation in the set covering problem resolution with a self-adaptive intelligent water drops algorithm," *Advances in Science, Technology and Engineering Systems*, vol. 6, pp. 134–145, 2021.

[8] D. Tapia, B. Crawford, R. Soto, W. Palma, J. Lemus-Romani *et al.*, "Embedding Q-learning in the selection of metaheuristic operators: The enhanced binary grey wolf optimizer case," in *de 2021 IEEE Int. Conf. on Automation/XXIV Congress of the Chilean Association of Automatic Control (ICA-ACCA)*, Santiago, Chile, 2021.

[9] A. Caprara, P. Toth and M. Fischetti, "Algorithms for the set covering problem," *Annals of Operations Research*, vol. 98, pp. 353–371, 2000.

[10] R. -L. Wang and K. Okazaki, "An improved genetic algorithm with conditional genetic operators and its application to set-covering problem," *Soft Computing*, vol. 11, pp. 687–694, 2007.

[11] M. Caserta, "Tabu search-based metaheuristic algorithm for large-scale set covering problems," in *Metaheuristics*, Springer, Boston, MA: Springer, pp. 43–63, 2007.

[12] Z. -G. Ren, Z. -R. Feng, L. -J. Ke and Z. -J. Zhang, "New ideas for applying ant colony optimization to the set covering problem," *Computers & Industrial Engineering*, vol. 58, pp. 774–784, 2010.

[13] B. Crawford, R. Soto, R. Cuesta and F. Paredes, "Application of the artificial bee colony algorithm for solving the set covering problem," *The Scientific World Journal*, vol. 2014, pp. 1–8, 2014.

[14] B. Crawford, R. Soto, M. O. Suárez, F. Paredes and F. Johnson, "Binary firefly algorithm for the set covering problem," in *de 2014 9th Iberian Conf. on Information Systems and Technologies (CISTI)*, Barcelona, Spain, pp.65–73, 2014.

[15] B. Crawford, R. Soto, N. Berríos, F. Johnson and F. Paredes, "Binary cat swarm optimization for the set covering problem," in *de 2015 10th Iberian Conf. on Information Systems and Technologies (CISTI)*, Águeda, Aveiro, Portugal, 2015.

[16] R. Soto, B. Crawford, R. Olivares, J. Barraza, I. Figueroa *et al.,* "Solving the non-unicost set covering problem by using cuckoo search and black hole optimization," *Natural Computing*, vol. 16, pp. 213–229, 2017.

[17] B. Crawford, R. Soto, F. Aballay, S. Misra, F. Johnson *et al.,* "A teaching-learning-based optimization algorithm for solving set covering problems," in *de Int. Conf. on Computational Science and its Applications*, Banff, Alberta, Canada, 2015.

[18] B. Crawford, R. Soto, C. Peña, W. Palma, F. Johnson *et al.,* "Solving the set covering problem with a shuffled frog leaping algorithm," in *de Asian Conf. on Intelligent Information and Database Systems*, Bali, Indonesia, 2015.

[19] J. García, B. Crawford, R. Soto and P. García, "A multi dynamic binary black hole algorithm applied to set covering problem," in *de nt. Conf. on Harmony Search Algorithm*, Bilbao, Spain, 2017.

[20] B. Crawford, R. Soto, C. Torres-Rojas, C. Peña, M. Riquelme-Leiva *et al.,* "A binary fruit fly optimization algorithm to solve the set covering problem." in *Int. Conf. on Computational Science and its Applications*, Springer, Cham, 2015.

[21] W. -T. Pan, "A new fruit fly optimization algorithm: Taking the financial distress model as an example," *Knowledge-Based Systems*, vol. 26, pp. 69–74, 2012.

[22] J. E. Beasley, "An algorithm for set covering problem," *European Journal of Operational Research*, vol. 31, pp. 85–93, 1987.

[23] M. L. Fisher and P. Kedia, "Optimal solution of set covering/partitioning problems using dual heuristics," *Management Science*, vol. 36, pp. 674–688, 1990.

[24] S. -M. Lin, "Analysis of service satisfaction in web auction logistics service using a combination of fruit fly optimization algorithm and general regression neural network," *Neural Computing and Applications*, vol. 22, pp. 783–791, 2013.

[25] H. -Z. Li, S. Guo, C. -J. Li and J. -Q. Sun, "A hybrid annual power load forecasting model based on generalized regression neural network with fruit fly optimization algorithm," *Knowledge-Based Systems*, vol. 37, pp. 378–387, 2013.

[26] Y. Xing, "Design and optimization of key control characteristics based on improved fruit fly optimization algorithm," *Kybernetes*, vol. 42, no. 3, pp. 466–481, 2013.

[27] Z. He, H. Qi, Y. Yao and L. Ruan, "Inverse estimation of the particle size distribution using the fruit fly optimization algorithm," *Applied Thermal Engineering*, vol. 88, pp. 306–314, 2015.

[28] L. Wang, X. -L. Zheng and S. -Y. Wang, "A novel binary fruit fly optimization algorithm for solving the multidimensional knapsack problem," *Knowledge-Based Systems*, vol. 48, pp. 17–23, 2013.

[29] J. E. Beasley and P. C. Chu, "A genetic algorithm for the set covering problem," *European Journal of Operational Research*, vol. 94, pp. 392–404, 1996.

[30] J. M. Lanza-Gutierrez, B. Crawford, R. Soto, N. Berrios, J. A. Gomez-Pulido *et al.,* "Analyzing the effects of binarization techniques when solving the set covering problem through swarm optimization," *Expert Systems with Applications*, vol. 70, pp. 67–82, 2017.

[31] B. Crawford, R. Soto, G. Astorga, J. García, C. Castro *et al.,* "Putting continuous metaheuristics to work in binary search spaces," *Complexity*, vol. 2017, pp. 1–19, 2017.

[32] M. Olivares-Suarez, W. Palma, F. Paredes, E. Olguín and E. Norero, "A binary coded firefly algorithm that solves the set covering problem," *Science and Technology*, vol. 17, pp. 252–264, 2014.

[33] S. Mirjalili and A. Lewis, "S-shaped versus V-shaped transfer functions for binary particle swarm optimization," *Swarm and Evolutionary Computation*, vol. 9, pp. 1–14, 2013.

[34] Z. W. Geem, J. H. Kim and G. V. Loganathan, "A new heuristic optimization algorithm: Harmony search," *Simulation*, vol. 76, pp. 60–68, 2001.

[35] N. Rajalakshmi, D. P. Subramanian and K. Thamizhavel, "Performance enhancement of radial distributed system with distributed generators by reconfiguration using binary firefly algorithm," *Journal of the Institution of Engineers (India): Series B*, vol. 96, pp. 91–99, 2015.

[36] E. Balas and A. Ho, "Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study," in *de Combinatorial Optimization*, Berlin, Heidelberg: Springer, pp. 37–60, 1980.

[37] J. E. Beasley, "A lagrangian heuristic for set-covering problems," *Naval Research Logistics (NRL)*, vol. 37, pp. 151–164, 1990.

[38] R. Soto, B. Crawford, A. Munoz, F. Johnson and F. Paredes, "Pre-processing, repairing and transfer functions can help binary electromagnetism-like algorithms," in *de Artificial Tntelligence Perspectives and Applications*, Switzerland, Cham: Springer, pp. 89–97, 2015.

[39] R. Cuesta, B. Crawford, R. Soto and F. Paredes, "An artificial bee colony algorithm for the set covering problem," in *de Modern Trends and Techniques in Computer Science*, Switzerland, Cham: Springer, pp. 53–63, 2014.
[40] N. Ghorbani, E. Babaei and F. Sadikoglu, "Bema: Binary exchange market algorithm," *Procedia Computer Science*, vol. 120, pp. 656–663, 2017.
[41] S. Ceria, P. Nobili and A. Sassano, "A Lagrangian-based heuristic for large-scale set covering problems," *Mathematical Programming*, vol. 81, pp. 215–228, 1998.