Nat Comput (2017) 16:213–229
DOI 10.1007/s11047-016-9609-7

Solving the non-unicost set covering problem by using cuckoo search and black hole optimization

Ricardo Soto[1] · Broderick Crawford[1] · Rodrigo Olivares[1,2] · Jorge Barraza[1] · Ignacio Figueroa[1] · Franklin Johnson[3] · Fernando Paredes[4] · Eduardo Olguín[5]

Published online: 11 January 2017

Abstract The set covering problem is a classical optimization benchmark that finds application in several real-world domains, particularly in line balancing production, crew scheduling, and service installation. The problem consists in finding a subset of columns in a zero-one matrix such that they cover all the rows of the matrix at a minimum cost. In this paper, we present two new approaches for efficiently solving this problem, the first one based on cuckoo search and the second one on black hole optimization. Both are relatively modern bio-inspired metaheuristics that have attracted much attention due to their rapid convergence, easy implementation, and encouraging obtained results. We integrate to the core of both metaheuristics an effective pre-processing phase as well as multiple transfer functions and discretization methods. Pre-processing is employed for filtering the values from domains leading to infeasible solutions, while transfers function and discretization methods are used for efficiently handling the binary nature of the problem. We illustrate interesting experimental results where the two proposed approaches are able to obtain various global optimums for a set of well-known set covering problem instances, outperforming also several recently reported techniques.

Keywords Metaheuristic · Bio-inspired · Set covering problem · Cuckoo search · Black hole

1 Introduction

The set covering problem (SCP) is one of the well-known Karp's 21 NP-complete problems. The goal of such a problem is to find a subset of columns in a zero-one matrix such that they cover all the rows of the matrix at a minimum cost. The non-unicost SCP is a slight variant of this problem where the cost of columns are different. Several applications of the non-unicost SCP can be seen in the real world, particularly related to line balancing production (Salveson 1955), crew scheduling (Baker et al. 1979; Bartholdi 1981; Rubin 1973), service installation (Toregas et al. 1971; Walker 1974), databases (Munagala et al. 2004), and Boolean expressions (Breuer 1970). The literature reports various algorithms to solve this problem which range from classical exact methods (Balas 1997; Beasley 1987; Yelbay et al. 2014; Avis 1980; Rushmeier and Nemhauser 1993) to very recent bio-inspired metaheuristics (Crawford et al. 2014; Karaboga 2005).

In this paper, we present two new approaches for efficiently solving different instances of the SCP: cuckoo search and black hole optimization. Both are relatively recent bio-inspired metaheuristics that have attracted much attention during the last years, both being succesfully used in different application domain, such as data clustering, image processing, data mining, and computervision (Kumar et al. 2015; Yang and Deb 2014). Important features that share these metaheuristics are the rapid convergence

This paper is an extended version of Soto et al. (2015a).

✉ Rodrigo Olivares
rodrigo.olivares@uv.cl

[1] Pontificia Universidad Católica de Valparaíso, Valparaíso, Chile

[2] Universidad de Valparaíso, Valparaíso, Chile

[3] Universidad de Playa Ancha, Valparaíso, Chile

[4] Escuela de Ingeniería Industrial, Universidad Diego Portales, Santiago, Chile

[5] Universidad San Sebastián, Santiago, Chile

and the notably easy implementation in order to model and solve the problem at hand. We integrate to the core of both metaheuristics an effective pre-processing phase as well as multiple transfer functions and discretization methods. Pre-processing is used for eliminating those values that do not lead to any feasible solutions. The goal is to alleviate the work of the metaheuristic. Transfers function are used to map the real values generated by the movement operator of both metaheuristics to a [0, 1] real interval. This interval is then used by the discretization method to generate a binary value according to the nature of the SCP. We perform a large set of experiments involving 8 transfer functions and 3 discretization methods in order to select the best performing combination of them. Interesting experimental results are obtained, where the two proposed approaches are able to obtain various global optimums for a set of 65 well-known SCP instances, outperforming also several recently reported techniques.

The remainder of this paper is organized as follows: The related work is introduced in the next section. A brief description of the SCP and pre-processing phases are detailed in Sect. 3. Sect. 4 presents the metaheuristics employed in this work, followed by their corresponding binary representations. Sects. 6 and 7 illustrate the repair of infeasible solutions and the experimental results, respectively. Finally, we conclude and suggest some lines of future research.

2 Related work

The SCP has widely been explored in the optimization and mathematical programming sphere. Preliminary works relied on the proper use of branch-and-bound and branch-and-cut algorithms (Balas 1997) and (Beasley 1987), which are exact methods commonly unable to tackle large instances of the SCP. Greedy algorithms have also been proposed, however their deterministic nature hinders the generation of high quality solutions (Chvatal 1979). This problem may be improved by the incorporation of memory and random components as detailed in Lan and DePuy (2006). As noted in Ceria et al. (1998) and Caprara et al. (1999) Lagrangian relaxation-based heuristics are in general much more effective. A more detailed description of efficient exact methods and heuristics devoted to SCPs can be found in Caprara et al. (2000).

It is well-known that exact methods are in general unable to tackle large instances of NP-complete problems, consequently much research work has been devoted to the study of efficient metaheuristics to solve hard SCP instances in a bounded amount of time. For instance ant colony optimization (Crawford et al. 2011, 2013; Valenzuela et al. 2014), tabu search (Caserta 2007), simulated annealing (Brusco et al. 1999), and genetic algorithms (Yelbay et al. 2014) have extensively been proposed to tackle the classic SCP. More recent metaheuristics for solving this problem can also be found in the literature, such as firefly optimization (Crawford et al. 2014), cat swarm optimization (Crawford et al. 2015b, d), shuffled frog leaping (Crawford et al. 2015a, c), artificial bee colony (Karaboga 2005), electromagnetism-like (Birbil and Fang 2003), among others.

As top-level general search strategies, metaheuristics have also largely been applied to solve SCPs in recent years.

3 Problem description

In this section we provide the formal definition of the SCP (Gass and Fu 2013). Let A be an n-row and m-column incidence matrix, where $a_{i,j}$ denotes the value taken by position (i, j) of A, being $i = 1,\ldots,n$ and $j = 1,\ldots,m$. Let us note that a row i is covered by column j if $a_{ij} = 1$. Every column j hold a non-negative cost c_j, for the non-unicost SCP, c_j vary for each column. The goal is to find a minimun cost such that each row i is covered by at least one column j. The corresponding mathematical is stated in the following:

$$Minimize\ z = \sum_{j=1}^{m} c_j x_j \tag{1}$$

subject to

$$\sum_{j=1}^{m} a_{ij} x_j \geq 1, \quad \forall i = 1,\ldots,n \tag{2}$$

$$x_j \in \{0,1\}, \quad \forall j = 1,\ldots,m \tag{3}$$

The idea is to minimize the summation of the costs of the columns, where $x_j = 1$ if column j belongs to the solution and $x_j = 0$ otherwise. The constraints of the SCP guarantee that every row i is covered by at least one column j.

For clarifying the problem, we present an example of a unicost SCP, therefore the cost vector $c_j = 1$, $\forall j = 1,\ldots,m$.

There are six cities ($city_1$ to $city_6$) in region R. The region must determine where to build fire stations. The region wants to build the minimum number of fire stations and ensure that at least one fire station is near 15 min of each city. The times (in minutes) required to drive between cities are illustrated in Table 1.

To formulate the integer-programming model, we consider the following decision variable:

$$x_j = \begin{cases} 1, & \text{if a fire station is built in city}_\text{i} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

Table 1 Times required to drive between cities

From	To					
	$city_1$	$city_2$	$city_3$	$city_4$	$city_5$	$city_6$
$city_1$	0	10	20	30	20	20
$city_2$	10	0	25	35	30	10
$city_3$	20	25	0	15	30	20
$city_4$	30	35	15	0	15	25
$city_5$	20	30	30	15	0	14
$city_6$	20	10	20	25	14	0

Objective function is given by minimum sum of the decision variables activated ($x_j = 1$).

$$Minimize\, z = \sum_{j=1}^{6} x_j \tag{5}$$

Constraints: A fire station within 15 min of each city. This can be seen in the following summary:

$city_1$	$city_1, city_2$	$\Rightarrow$	$x_1 + x_2 \geq 1$
$city_2$	$city_1, city_2, city_6$	$\Rightarrow$	$x_1 + x_2 + x_6 \geq 1$
$city_3$	$city_3, city_4$	$\Rightarrow$	$x_3 + x_4 \geq 1$
$city_4$	$city_3, city_4, city_5$	$\Rightarrow$	$x_3 + x_4 + x_5 \geq 1$
$city_5$	$city_4, city_5, city_6$	$\Rightarrow$	$x_4 + x_5 + x_6 \geq 1$
$city_6$	$city_2, city_5, city_6$	$\Rightarrow$	$x_2 + x_5 + x_6 \geq 1$

Thus, the integer-programming model is as follows:

$$\begin{aligned}
& Minimize\, z = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \\
& \text{subject to}: \\
& x_1 + x_2 \geq 1 \\
& x_1 + x_2 + x_6 \geq 1 \\
& x_3 + x_4 \geq 1 \\
& x_3 + x_4 + x_5 \geq 1 \\
& x_4 + x_5 + x_6 \geq 1 \\
& x_2 + x_5 + x_6 \geq 1 \\
& x_j \in \{0, 1\}, \forall j = 1, \ldots, 6
\end{aligned} \tag{6}$$

Optimal solution is given by $(z, x_1, x_2, x_3, x_4, x_5, x_6) = (2, 0, 1, 0, 1, 0, 0)$ and it represented by the binary vector in Fig. 1: where each value x_j is a component of the solution.

The optimal solution represents the cities in which should be built the fire station (minimum), such that the distance between one city and one fire station is not greater than 15 min.

To transform this example in a non-unicost problem, is necessary to differentiate the installation cost of each fire station (cost vector c_j). In that case, the objective function is given by:

$$Minimize\, z = \sum_{j=1}^{6} c_j x_j \tag{7}$$

where c_j is the cost vector associated to each $x_j, \forall j = 1, \ldots, 6$. If we consider the cost for each fire station is started in Table 2.

The n-tuple $(z, x_1, x_2, x_3, x_4, x_5, x_6) = (120, 1, 0, 1, 0, 0, 1)$ is the vector that describes the new optimal solution and it depicted in Fig. 2.

Finally, for applying correct way the proposed algorithms, we define x_i-binary vector-as the solution i of SCP and x^j as a component part of solution, whose values can be 1, if this component is part of solution or 0 it is not.

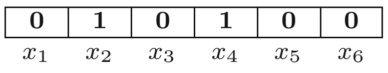


Fig. 1 Representation of binary unicost solution

4 Bio-inspired approaches

The bio-inspired algorithms, have been placed among the best algorithms for solving optimization problems. In this paper, we solve the SCP by using two bio-inspired approaches: cuckoo search and black hole optimization.

4.1 Cuckoo search algorithm

Cuckoo search algorithm (Fouladgar and Lotfi 2015; Yang and Deb 2009) is inspired from the obligate brood parasitism of some cuckoo species by laying their eggs in the nests of other bird species. For simplicity in describing the cuckoo search, here we use the following three idealized rules:

1. Each cuckoo lays one egg at a time and dumps it in a randomly chosen nest.
2. The best nests with high-quality eggs will be carried over to the next generations.
3. The number of available host nests is fixed, and the egg laid by a cuckoo is discovered by the host bird with a probability $p_a \in [0, 1]$. In this case, the host bird can either get rid of the egg or simply abandon the nest and build a completely new nest.

A new solution is generated by using Lévy flight (Yang and Deb 2009) and it is given via Eqs. 8 and 9.

$$\begin{aligned}
& x_i^d(t+1) = x_i^d(t) + \alpha \oplus \text{Levy}(\beta) \\
& \quad \forall i \in \{1, \ldots, n\} \wedge \forall d \in \{1, \ldots, m\}
\end{aligned} \tag{8}$$

where $x_i^d(t)$ is the component d of a solution i at iteration t. Analogously, $x_i^d(t+1)$ is the component d of a solution i at iteration $t+1$. $\alpha \geq 0$ is the step size, which is associated to the range of values that the problem needs (scale value), being determined by upper and lower bounds.

$$\text{Levy} \sim \text{u} = \text{t}^{-\beta}, (1 < \beta < 3) \quad (9)$$

The Lévy flight represents a random trajectory and it is the length of the step obtained by means of a Lévy distribution which has infinite mean and variance. In this case, the consecutive steps of a cuckoo form a process of random walk that follows a distribution of powers law with double tail step length (Fouladgar and Lotfi 2015).

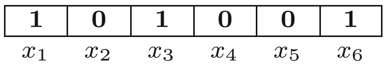
Algorithm 1 Cuckoo Search via Lévy flights.

```
Require: SCP input data, $n$, $p_a$, $\alpha$, $\beta$, $T$
Ensure: The best solution that resolves the SCP
 1: $loadSCPData()$
 2: {Produce the first generation of $n$ nests}
 3: for each nest $n_i, (\forall i = 1, \ldots, n)$ do
 4:    for each dimension $d$, $(\forall d = 1, \ldots, m)$ do
 5:       $x_i^d \leftarrow Random\{0, 1\}$
 6:    end for
 7:    $f_i \leftarrow z(x_i)$
 8: end for
 9: $globalfit \leftarrow +\infty$
10: {Produce $T$-generations of $n$ nests}
11: while $t < T$ do
12:    $\{minfit, minindex\} \leftarrow min(f)$
13:    if $minfit < globalfit$ then
14:       $globalfit \leftarrow minfit$
15:       $\hat{x}^d(t) \leftarrow x_{minindex}^d(t)$
16:    end if
17:    for each nest $n_i, (\forall i = 1, \ldots, n)$ do
18:       for each dimension $d$, $(\forall d = 1, \ldots, m)$ do
19:          if $rand > p_a$ then
20:             {Select the worst nests according to $p_a[0, 1]$
                and and replace them for new solutions}
21:          end if
22:       end for
23:    end for
24:    for each nest $n_i, (\forall i = 1, \ldots, n)$ do
25:       for each dimension $d$, $(\forall d = 1, \ldots, m)$ do
26:          {Generate new solutions through Equations (8)
             and (9)}
27:          $x_i^d(t+1) \leftarrow x_i^d(t) + \alpha \oplus \text{Lévy}(\beta)$
28:          {In previous step, the value generated belongs to
             the real domain and it must be brought to a bi-
             nary domain}
29:          $x_i^d(t+1) \leftarrow T(x_i^d(t+1))$
30:       end for
31:    end for
32: end while
33: Postprocess results and visualization;
```

Algorithm 1 depicts the proposed procedure. At the beginning in Line 1, input data from SCP is processed and loaded. Then, at Lines 3–8, an initial population of *n* nests is randomly generated as a vector of *m* binary values that corresponds to one nest or one potential solution (Line 5).

Table 2 Costs required to build the fire stations in each city

Fire station in the city *i*	To (μ)
$city_1$	35
$city_2$	65
$city_3$	40
$city_4$	65
$city_5$	45
$city_6$	45



1	0	1	0	0	1
x_1	x_2	x_3	x_4	x_5	x_6

Fig. 2 Representation of binary non-unicost solution

For each nest (potential solution), the fitness value is calculated by evaluating the objective function *z* (Line 7).

For generating new solutions, the while loop manages the actions which are self-explanatory, at Lines 11–32. As previously mentioned, the SCP is a minimization problem. This evaluation is handled in comparison presented at Line 13. If the new min value is lower than min global, the min global is changed by the new min value and the best solution is stored in $\hat{x}$.

Next for loop statement (Lines 17–23) allows to select worst solutions according to random a value $rand \in [0, 1]$ greater than p_a.

Finally, the last for loop statement (Lines 24–31) generates new solutions according to Eq. (8), for each dimension *j*. This value belongs to a real domain and it must be brought to a binary domain, thus we used a *T* function that transform the real value to a binary one. In this paper, we test different binarization approaches described in Sect. 5.

4.2 Black hole optimization

Black hole optimization is a population-based method inspired on the black hole phenomenon (Hatamlou 2013; Kumar et al. 2015). A black hole is a region of space that has so much mass concentrated in it that there is no way for a nearby object to escape its gravitational pull. Anything falling into a black hole, including light, cannot escape.

Similar to other population-based algorithms, the black hole algorithm starts with an initial population of potential solutions to an optimization problem and an objective function that is calculated for them. At each iteration of the black hole algorithm, the best candidate is selected to be the black hole, which then starts pulling other candidates around it, called stars. If a star gets too close to the black hole, it will be swallowed and it is gone forever. In such a case, a new star (potential solution) is randomly generated and placed in the search space and starts a new search.

The black hole has the ability to absorb the stars that surround it. After initializing the black hole and stars, the black hole begins by absorbing the stars around it and all the stars start moving towards the black hole. The absorption of stars by the black hole is formulated as follows:

$$x_i^d(t+1) = x_i^d(t) + r[x_{BH}^d - x_i^d(t)], \forall i \in \{1,\ldots,N\} \qquad (10)$$

where $x_i^d(t)$ and $x_i^d(t+1)$ are the components of a solution i at iterations t and $(t+1)$, respectively. x_{BH}^d is the component of best solution (black hole) in the search space. r is a random number in the interval [0, 1]. N is the number of stars (potential solutions).

While moving towards the black hole, a star may reach a location with lower cost than the black hole. In such a case, the black hole moves to the location of that star and vice versa. Then the algorithm will continue with the black hole in the new location and then stars start moving towards this new location.

In addition, there is the probability of crossing the event horizon during moving stars towards the black hole. Every star (or potential solution) that crosses the event horizon of the black hole will be swallowed by the black hole. Every time a candidate is sucked in by the black hole, another potential solution (star) is born and distributed randomly in the search space and starts a new search. This is done to keep the number of potential solutions constant. The next iteration takes place after all the stars have been moved.

The radius of the event horizon in the black hole algorithm is calculated using the following equation:

$$R = \frac{f_{BH}}{\sum_{i=1}^{N} f_i} \qquad (11)$$

where f_{BH} is the fitness value of the black hole and f_i is the fitness value of the i-th star. N is the number of stars (potential solutions).

When the distance between a star and the black hole (best candidate) is less than R, that candidate is collapsed and a new star is created and distributed randomly in the search space. Based on the above description the main steps in the black hole algorithm are detailed in Algorithm 2.

Algorithm 2 Black Hole Optimization.

```
Require: SCP input data, $n$, $T$
Ensure: The best solution that resolves the SCP
1: $loadSCPData()$
2: {Produce the first generation of $n$ stars}
3: for each star $s_i, (\forall i = 1, \ldots, n)$ do
4:     for each dimension $d$, $(\forall d = 1, \ldots, m)$ do
5:         $x_i^d \leftarrow Random\{0,1\}$
6:     end for
7:     $f_i \leftarrow z(x_i)$
8: end for
9: $globalfit \leftarrow +\infty$
10: {Produce $T$-generations of $n$ stars}
11: while $t < T$ do
12:     $\{minfit, minindex\} \leftarrow min(f)$
13:     if $minfit < globalfit$ then
14:         $globalfit \leftarrow minfit$
15:         $\hat{x}^d(t) \leftarrow x_{minindex}^d(t)$
16:     end if
17:     for each star $s_i, (\forall i = 1, \ldots, n)$ do
18:         {Calculate the event horizon through Equation (11)}
19:         $R = \frac{f_{BH}}{\sum_{i=1}^{N} f_i}$
20:         if $rand > R$ then
21:             for each dimension $d$, $(\forall d = 1, \ldots, m)$ do
22:                 $x_i^d(t) \leftarrow Random\{0,1\}$
23:             end for
24:         end if
25:     end for
26:     for each star $s_i, (\forall i = 1, \ldots, n)$ do
27:         for each dimension $d$, $(\forall d = 1, \ldots, m)$ do
28:             {Generate new solutions through Equation (10)}
29:             $x_i^d(t+1) = x_i^d(t) + r[x_{BH}^d - x_i^d(t)]$
30:             {In previous step, the value generated belongs to the real domain and it must be brought to a binary domain}
31:             $x_i^d(t+1) \leftarrow T(x_i^d(t+1))$
32:         end for
33:     end for
34: end while
35: Postprocess results and visualization;
```

Similar to the Cuckoo Search Algorithm, the Black Hole optimization begins with the loading and processing phases. Then, at Lines 3–8, an initial population of n stars is randomly generated as a vector of m binary values that corresponds to one star or potential solution (Line 5). For each star, the fitness value is calculated by evaluating the objective function z (Line 7).

Then, while a termination criterion (a maximum number of iterations or a sufficiently good solution was not reached) is met, each fitness of a potential solution is evaluated (Lines 11–34). As previously mentioned, the SCP is a minimization problem. This evaluation is handled in comparison presented at Line 13. If the new min value is less than min global, the min global is changed by the new min value and the best solution is stored in $\hat{x}$ (black hole).

If a star crosses the event horizon of the black hole (calculated via Eq. 11), replace it with a new star in a random location in the search space. This process is described in the for loop statement at Lines 17–25.

Finally, the last for loop statement (Lines 26–33) generates new solutions according to Eq. (10), for each dimension d. As in the Cuckoo Search algorithm, this value belongs to real domain and it must be brought to a binary domain, thus a T function is used again to transform the real value in a binary one.

5 Binary approaches

The set covering is a problem whose domain is limited to binary values, namely, $x_i^d \in \{0,1\}, \forall j \in \{1,\ldots,m\}$. Due to this reason, we use a binary representation for the SCP solution as showed in Fig. 3, where:

$$x_i^d = \begin{cases} 1, & \text{if the dimension d belongs to the solution} \\ 0, & \text{otherwise} \end{cases}$$

In this work, we use the standard forms in both algorithms; for the cuckoo search algorithm (Pereira et al. 2014), each nest holds only one egg and represents a potential solution and for the black hole optimization (Nemati et al. 2013), each star represent a potential solution.

However, the approximate methods used for solving the SCP are designed to solve problems with real domains. This task is resolved by transforming domains, by applying binarization strategies, which are responsible for forcing elements to move in a binary space. The binarization strategy is composed of a transfer function and a discretization method. In this work, we tested 32 different binarization strategies.

We tested eight different functions (see Table 3), separated into two families (Mirjalili and Lewis 2013): $\mathcal{S}$-Shape and $\mathcal{V}$-Shape.

Once a transfer function is applied, the input real number is mapped to a real number belonging to a [0, 1] interval. Then, a discretization method is required to produce a binary value from the real one. For achieving this, we test four different methods:

1. *Standard* If condition is satisfied, standard method return 1, otherwise, return 0.

i-th solution	1	0	1	$\ldots$	0	1
x_i	x_i^1	x_i^2	x_i^3	$\ldots$	$x_i^{(d-1)}$	x_i^d

Fig. 3 Binary solution representation

$$x_i^d(t+1) = \begin{cases} 1, & \text{if rand } \leq \mathrm{T}(\mathrm{x_i^d}(\mathrm{t}+1)) \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

2. *Complement* If condition is satisfied, standard method return the complement value.

$$x_i^d(t+1) = \begin{cases} \overline{x_i^d(t+1)'}, & \text{if rand} \leq \mathrm{T}(\mathrm{x_i^d}(\mathrm{t}+1)) \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

3. *Static probability* A probability is generated and it is evaluated with a transfer function.

$$x_i^d(t+1) = \begin{cases} 0, & \text{if } \mathrm{T}(\mathrm{x_i^d}(\mathrm{t}+1)) \leq \alpha \\ x_i^d(t+1)', & \text{if } \alpha \quad T(x_i^d(t+1)) \leq \frac{1}{2}(1+\alpha) \\ 1, & \text{if } \mathrm{T}(\mathrm{x_i^d}(\mathrm{t}+1)) \geq \frac{1}{2}(1+\alpha) \end{cases} \tag{14}$$

4. *Elitist* Discretization method Elitist Roulette, also known as Monte Carlo, is to select randomly among the best individuals of the population, with a probability proportional to its fitness.

$$x_i^d(t+1) = \begin{cases} x_i^d(t+1)', & \text{if rand} \leq \mathrm{T}(\mathrm{x_i^d}(\mathrm{t}+1)) \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

6 Heuristic feasibility operator

Bio-inspired algorithms may generate infeasible solutions. In the SCP, an infeasible solutions corresponding to those solutions that uncovered rows, clearly violates a subset of constraints. Due to this problem, we employed an additional heuristic operator that achieves the generation of feasible solutions, and additionally eliminates column redundancy.

For making all feasible solutions, we calculate a percentage based on the cost of column j over the sum of all the constraint matrix rows covered by a column j, as shown in Eq. (16).

$$\frac{cost(column_j)}{coveredRows(column_j)} \tag{16}$$

A unfeasible solution is repaired by covering the columns of the solution that had the lower percentage. Then, the heuristic applies a local optimization step for removing the column redundancy. If a column is deleted and the feasibility of the solution is not affected, then the column is redundant.

Algorithm 3 Heuristic Feasibility Operator.

```
1: $I \leftarrow$ The set of all rows;
2: $J \leftarrow$ The set of all columns;
3: $\alpha_i \leftarrow$ The set of columns that cover row $i$, $i \in I$;
4: $\beta_j \leftarrow$ The set of rows covered by column $j$, $j \in J$;
5: $S \leftarrow$ The set of columns in a solution;
6: $w_i \leftarrow$ The number of columns that cover row $i$, $i \in I$. For
   this, $w_i \leftarrow |S \cap \alpha_i|$, $\forall\ i \in I$;
7: $U \leftarrow$ The set of uncovered rows. For this, $U = \{i \mid w_i =$
   $0,\ \forall\ i \in I\}$;
8: for each 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 $S$ and set $w_i \leftarrow w_i + 1$, $\forall\ i \in b_j$;
11:   Set $U \leftarrow U - b_j$;
12: end for
13: for each column $j$ $inS$ (in decreasing order of $j$) do
14:   if $w_i \geq 2$ then
15:      $S \leftarrow S - j$;
16:      $w_i \leftarrow w_i - 1$, $\forall\ i \in \beta_j$;
17:   end if
18: end for
19: $S$ is now a feasible solution for the SCP that contains no
    redundant columns;
```

Algorithm 3 starts with the initialization of variables from instance in Lines 1–5, The recognition of the rows that are not covered are in Lines 6 and 7. Between the statements 8 and 18 is "greedy" heuristic. On the one hand, between the instructions 8 and 12, the columns with lower ratio are added to the solution. On the other hand, between lines 13 and 18, the redundant columns with higher costs are deleted while the solution is feasible.

7 Experimental results

We have performed an experimental evaluation of the proposed approaches on a set of 65 different well-known instances, organized in 11 sets from the Beasley's OR-library.[1] Table 4 describes instance set, number of rows or constraints (m), number of columns or variables (n), range of costs, density (percentage of non-zeroes in the matrix) and if the optimal solution is known or unknown.

In order to reduce the instance size of SCP, we have used a pre-processed instances set. Different pre-processing methods have particularly been proposed for the SCP (Fisher and Kedia 1990). We employ two of them, which have been proved to be the most effective ones:

- *Column Domination* The non-unicost SCP holds different column costs, then once a set of rows I_j is covered by another column j' and $c_{j'} < c_j$, we say that column j is dominated by j', then column j is removed from the solution.
- *Column Inclusion* If a row is covered by only one column after the above domination, means that there is no better column to cover those rows, consequently this column must be included in optimal solution.

The results are evaluated using the relative percentage deviation (*RPD*). *RPD* value quantifies the deviation of the objective value Z from Z_{opt} that in our case is the best known value for each instance (Z_{opt} in Table 4) and it is calculated as follows:

$$RPD = \left(\frac{Z - Z_{opt}}{Z_{opt}}\right) \times 100 \tag{17}$$

The minimum (Min), maximum (Max) and average (Avg) of the solutions obtained were achieved running 30 executions over each one of the 65 SCP test instances. To calculate *RPD* value, we used $Z = Z_{min}$. We test all the combinations of transfer functions and discretization methods over all these instances. Binarization strategies that achieved the best results for the black hole algorithm and cuckoo search are stated in Table 5:

Parameters tuning is known to be a complex task, being itself recognized as an optimization problem. To select these parameters, we performed a sampling test. In this pre-evaluation, both algorithms were executed 10 times, each one for the different population sizes: 1000, 100, 50, and 30 initial solutions and for the different number of generations; keeping all other parameters constant. To this end, we can see that when the population size (n) decreased and the number of generations (T) increased, the fitness value converges more quickly towards a minimum value. Table 6 show the parameter tuning achieved.

The implementation of both algorithms has been done in *Java*^{TM} and experiments have been launched on a 3.0 GHz Quad Core with 8 GB RAM machine running MS Windows 7 Ultimate. We compare our binary cuckoo search (BCS) and binary black hole (BBH) algorithms with the global optimum—or best known—(Z_{opt}) as well as with very recent approaches for solving SCPs based on modern bio-inspired metaheuristics, namely binary cat swarm optimization (BCSO) (Crawford et al. 2015b), binary firefly optimization (BFO) (Crawford et al. 2014), binary shuffled frog leaping algorithm (BSFLA) (Crawford et al. 2015c), binary artificial bee colony (BABC) (Cuesta et al. 2014) and binary electromagnetism-like algorithm (BELA) (Soto et al. 2015c). In order to compare and understand the results, we highlighted in bold the best result for each instance.

Table 7 illustrates the results obtained for instances from groups 4 and 5. Regarding instance set 4, BCSO, BSFLA, BELA and BABC are unable to achieve optimal values and BFO reaches only two. BCS and BBH exhibit the best performance for this data set reaching five and eight global optimums, respectively. Now considering instance set 5,

[1] Available at http://goo.gl/9jTqkX.

Table 3 Transfer functions

$\mathcal{S}$-Shape	$\mathcal{V}$-Shape
$\mathcal{S}_1: T(x_i^d(t+1)) = \frac{1}{1+e^{-2x_i^d(t+1)}}$	$\mathcal{V}_1: T(x_i^d(t+1)) = \left\lvert erf\left(\frac{\sqrt{\pi}}{2}x_i^d(t+1)\right)\right\rvert$
$\mathcal{S}_2: T(x_i^d(t+1)) = \frac{1}{1+e^{-x_i^d(t+1)}}$	$\mathcal{V}_2: T(x_i^d(t+1)) = \left\lvert \tanh(x_i^d(t+1))\right\rvert$
$\mathcal{S}_3: T(x_i^d(t+1)) = \frac{1}{1+e^{\frac{-x_i^d(t+1)}{2}}}$	$\mathcal{V}_3: T(x_i^d(t+1)) = \left\lvert \frac{x_i^d(t+1)}{\sqrt{1+[x_i^d(t+1)]^2}}\right\rvert$
$\mathcal{S}_4: T(x_i^d(t+1)) = \frac{1}{1+e^{\frac{-x_i^d(t+1)}{3}}}$	$\mathcal{V}_4: T(x_i^d(t+1)) = \left\lvert \frac{2}{\pi}arc\tan\left(\frac{\pi}{2}x_i^d(t+1)\right)\right\rvert$

Table 4 SCP instances from the Beasley's OR-Library

Instance set	m	n	Cost range	Density (%)	Optimal solution
4	200	1000	[1,100]	2	Known
5	200	2000	[1,100]	2	Known
6	200	1000	[1,100]	5	Known
A	300	3000	[1,100]	2	Known
B	300	3000	[1,100]	5	Known
C	400	4000	[1,100]	2	Known
D	400	4000	[1,100]	5	Known
E	500	5000	[1,100]	10	Unknown
F	500	5000	[1,100]	20	Unknown
G	1000	10000	[1,100]	2	Unknown
H	1000	10000	[1,100]	5	Unknown

Table 5 Binarization strategies for cuckoo search and black hole optimization

Algorithm	Binarization strategy
Cuckoo search	$\mathcal{S}_2 + Standard$
Black hole	$\mathcal{S}_4 + Elitist$

Table 6 Parameter tuning for black hole and cuckoo search algorithms

Parameter	Algorithms	
	Cuckoo search	Black hole
Population size	$n = 15$ (Nests)	$n = 20$ (Stars)
Number of generations	$T = 4000$	$T = 4000$
Specifics	$p_a = 0.25$;	
	$\alpha = 0.01$;	
	$\beta = 1.5$;	

once again BCSO and BELA are unable to optimally solve any instance. BABC is capable to find two optimal values, while BFO reach three and BSFLA four. Once again BCS and BBH prove to be better methods, reaching both eight optimal values. In terms of solving time (in ms) in 4 and 5 groups, we compare BCS and BBH and we can observe that a big difference can not be seen. In some instances BBH is faster than BCS and in others is the opposite.

Table 8 depicts the results for the instances set 6, A, B and C. Firstly, for set instance 6, BCS and BBH behave similar, three and four optimal values, respectively. In contrast, previous approaches reach less than 10% in average of optimal values. For the group A, BBH continues exhibiting a good performance compared to its competitors, achieving 3 of 5 optimal values. The best results for the proposed approaches can be seen when solving the instances set B, where both algorithms are capable to find the 100% of global optimums. Finally, for the C group, BBH again exhibits an efficient performance reaching 3 of 5 optimal values.

Now, comparing the results in terms of solving time, we can see that for smaller instances -group 6- BCS converges faster than BBH, but for bigger instances of groups A, B and C, BBH finds the best solution more fast than BCS.

Table 9 describes the results for groups D, E, F and G. For the D and E group. BCS and BBH perform similar again, finding in both cases between 60 and 40% of the optimum values. For the F and G groups, three $RPD = 0$

Table 7 Computational results for instance set of groups 4 and 5

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
New 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	**0.00**	0.19	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	Time (*ms*)	1929.4	1922.6	2150.4	2073.8	2149.4	2094.0	1902.4	1950.8	1959.6	2169.3
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	**0.00**	**0.00**	0.2	0.39	**0.00**	**0.00**	0.2	0.46	**0.00**
	Time (*ms*)	1811.2	1926.0	1957.1	2108.1	1992.3	2102.2	2032.2	2202.6	2109.9	2017.1
Previous approaches											
BCSO	Z_{min}	459	570	590	547	545	637	462	546	711	537
	Z_{avg}	480	594	607	578	554	650	467	567	725	552
	RPD	7	11.3	14.3	10.7	6.4	13.8	7.4	11	10.9	4.5
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	517
	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
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
New 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	**0.00**	0.66	**0.00**	**0.00**	0.47	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	Time (*ms*)	2011.8	2084.6	2197.8	2219.5	2025.9	2302.9	2145.5	2135.1	2057.5	2142.7
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	**0.00**	0.99	0.88	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	Time (*ms*)	2001.7	2071.2	2221.8	2191.2	2127.4	2222.1	2332.1	2339.2	2271.2	2057.7
Previous approaches											
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	1.7	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	0.4	1.66	0.88	0	0	0	1.37	1.04	0.72	0

Table 7 continued

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
BELA	Z_{min}	280	318	242	251	225	247	316	315	314	280
	Z_{avg}	281	321	240	252	227	248	317	317	315	282
	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 instances set of groups 6, A, B and C

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
New 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	**0.00**	**0.00**	**0.00**	**0.00**	0.34	0.16	0.43	0.13	**0.00**
	Time (ms)	2817.5	2842.7	2852.1	2904.6	2938.3	2889.3	2713.7	2824.2	2999.1	2923.1
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	**0.00**	**0.00**	**0.00**	**0.00**	0.39	0.43	**0.00**	**0.00**
	Time (ms)	2915.7	2791.6	2751.7	3003.7	3038.3	2967.1	2983.2	3104.9	3111.1	3012.3
Previous approaches											
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	9.4	4.1	10.3	5.3	5	13	8.7	10.8	6	3
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	1.45	0.68	1.38	0	3.11	0.79	3.17	2.16	0.43	0
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

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
New 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	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.44	0.9	1.62	0.9	0.46
	Time (ms)	3255.8	3381.9	3372.4	3391.0	3455.6	3279.7	3392.3	3392.3	3352.3	3491.7
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**

Table 8 continued

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
	RPD	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.88	**0.00**	0.82	**0.00**	**0.00**
	Time	3122.1	3183.6	3452.0	3419.1	3501.0	3397.9	3291.1	3229.2	3323.1	3111.8
Previous approaches											
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	225	217
	Z_{avg}	72	78	80	81	73	232	224	254	227	219
	RPD	2.89	2.63	0	1.26	0	1.32	1.82	4.11	2.73	0.93
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 9 Computational results for instances set of groups D, E, F and G

Instance		D.1	D.2	D.3	D.4	D.5	E.1	E.2	E.3	E.4	E.5
Z_{opt}		60	66	72	62	61	29	30	27	28	28
New approaches											
BCS	Z_{min}	**60**	**66**	73	**62**	**61**	**29**	31	28	30	**28**
	Z_{avg}	**60**	**66**	74	**62**	**62**	**30**	32	29	31	**30**
	RPD	**0.00**	**0.00**	0.14	**0.00**	**0.00**	**0.00**	0.32	0.36	0.67	**0.00**
	Time	4681.7	4704.6	4697.1	5695.5	5746.4	5823.0	4815.0	4916.0	4847.0	4769.0
BBH	Z_{min}	**60**	67	73	**62**	**61**	**29**	31	28	29	**28**
	Z_{avg}	**60**	68	74	**62**	**62**	**30**	31	29	31	**29**
	RPD	**0.00**	1.51	1.38	**0.00**	**0.00**	**0.00**	3.33	3.7	3.57	**0.00**
	Time	6021.0	6094.1	6147.7	6511.6	6712.2	6200.0	6115.1	5991.9	7081.0	7191.1
Previous approaches											
BCSO	Z_{min}	65	70	79	64	65	29	34	31	32	30
	Z_{avg}	66	70	81	67	66	30	34	32	33	30
	RPD	8.3	6.1	9.7	3.2	6.6	0	13.3	14.8	14.3	7.1
BFO	Z_{min}	60	68	75	62	63	29	32	29	29	29
	Z_{avg}	61	68	77	62	63	31	32	30	31	29
	RPD	0	3.03	4.16	0	3.27	0	6.66	7.4	3.57	3.57
BSFLA	Z_{min}	60	67	75	63	63	29	31	28	29	28
	Z_{avg}	62	68	77	65	66	29	32	28	30	31
	RPD	0	1.52	4.17	1.61	3.28	0	3.33	3.7	3.57	0
BELA	Z_{min}	62	73	79	67	66	30	35	34	33	30
	Z_{avg}	62	74	81	69	67	31	35	34	34	31
	RPD	3.33	10.61	9.72	8.06	8.20	3.45	16.67	25.93	17.86	7.14
BABC	Z_{min}	60	68	76	63	63	29	32	29	29	29
	Z_{avg}	61	68	77	65	66	33	32	31	30	32
	RPD	0	3.03	5.56	1.61	3.28	0	6.67	7.41	3.57	3.57

Table 9 continued

Instance		F.1	F.2	F.3	F.4	F.5	G.1	G.2	G.3	G.4	G.5
Z_{opt}		14	15	14	14	13	176	154	166	168	168
New approaches											
BCS	Z_{min}	**14**	**15**	15	15	14	**176**	156	169	170	170
	Z_{avg}	**14**	**17**	16	15	15	**177**	157	170	171	171
	RPD	**0.00**	**0.00**	0.67	0.67	0.71	**0.00**	0.13	0.77	0.12	0.12
	Time	8361.0	9869.0	9823.0	9231.0	8820.0	10959.0	9518.0	9827.0	9339.0	9221.0
BBH	Z_{min}	**14**	**15**	16	15	14	179	158	169	170	**168**
	Z_{avg}	**15**	**16**	16	16	15	181	160	169	171	**169**
	RPD	**0.00**	**0.00**	4.28	7.14	7.69	1.7	2.59	1.8	1.19	**0.00**
	Time	9118.5	10212.1	9293.8	11023.3	12220.2	14494.4	15584.5	14874.0	14919.2	15921.2
Previous approaches											
BCSO	Z_{min}	17	18	17	17	15	190	165	187	179	181
	Z_{avg}	17	18	17	17	16	193	166	188	183	184
	RPD	21.4	20	21.4	21.4	15.4	8	7.1	20.6	6.5	7.7
BFO	Z_{min}	15	16	16	15	15	185	161	175	176	177
	Z_{avg}	17	16	17	18	19	191	163	177	176	181
	RPD	7.14	6.66	14.28	7.14	15.38	5.11	4.54	5.42	4.76	5.35
BSFLA	Z_{min}	15	15	16	15	15	182	161	173	173	174
	Z_{avg}	15	15	17	16	17	183	161	174	177	174
	RPD	7.14	0	14.29	7.14	15.38	3.41	4.55	4.22	2.98	3.57
BELA	Z_{min}	17	18	17	17	16	194	176	184	196	198
	Z_{avg}	17	18	18	19	17	196	176	185	197	199
	RPD	21.43	20	21.43	21.43	23.08	10.23	14.29	10.84	16.67	17.86
BABC	Z_{min}	14	16	16	15	15	183	162	174	175	179
	Z_{avg}	15	16	17	17	16	184	163	175	177	181
	RPD	0	6.67	14.29	7.14	15.38	3.98	5.19	4.82	4.17	6.55

are reached by the proposed algorithms. Previous approaches fail in general to find optimum values as the instance set becomes harder. Only BSFLA and BABC achieve one optimum for the instances belonging to sets F and G, while BBH and BCS reach three. Finally, for the most hard instance set (see Table 10), namely H, we observe that the *RPD* obtained by the proposed approaches outperform by far the performance of competitors, remaining very close to optimal values. Now, considering the time required to find a best solution, we can observe that BCS is dramatically more efficient than BBH, needing a less time for all instances of group A.

Table 11 and Fig. 4 summarizes the performance of compared approaches. BBH and BCS are the best performing approaches reaching 35 global optimums, outperforming by far its competitors, BSFLA taking the third place with 14 global optimums, followed by BSFLA, BFO, BCSO, and BELA with no global optimum reached.

7.1 Binary cuckoo search versus binary black hole

In this section we contrast the proposed approaches in order to determine which one performs better. To this end we compare convergence and we detailedly analyze the 30 executions performed for each instance through the *Kolmogorov–Smirnov–Lilliefors* (Lilliefors 1967) and *Wilcoxon's signed rank* (Mann and Whitney 1947) statistical tests.

For instance group 4, 5, 6 and A, the convergence is achieved between 25 and 50 ms, BCS showing a small superiority. For more hard instance sets, such as B, C, and D, the solving time needed slightly increases, ranging from 50 to 100 ms. Finally, for E, F, G and H instance groups, the convergence is achieved between 700 and 1000 ms, showing that although these instances are known to be hard, both approaches are able to find a good solution in a reasonable amount of time. We have also observed that the optimization process applied after the repairing function greatly helps the convergence of BBH and BCS, being as

Table 10 Computational results for intances of group H

Instance		H.1	H.2	H.3	H.4	H.5
Z_{opt}		63	63	59	58	55
New approaches						
BCS	Z_{min}	64	64	62	59	56
	Z_{avg}	64	64	63	60	57
	RPD	0.16	0.16	0.48	0.17	0.18
	Time	13801.0	14097.0	14049.0	16132.0	19384.0
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
	Time	17911.2	19733.4	21840.3	21421.7	22242.5
Previous approaches						
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

well an important step to improve the cost value of each instance. The experimental results also exhibit the robustness of the approaches as the fitness average for each instance remains very near to the optimum global. However, as the results in terms of solving time obtained are not conclusive to assert which approach performs better, we have also performed a statistical analysis to the 30 executions of each instance.

We firtly employ the *Kolmogorov-Smirnov-Lilliefors* test to determine the independence of samples and then we statistically compare them through the *Wilcoxon's signed rank*. For both tests, we consider a hypothesis evaluation, which is analyzed assuming a significance level of 0.05, i.e., smaller values that 0.05 determine that the corresponding hypothesis cannot be assumed. Both test were conducted using *GNU Octave*[2].

The first test allows us to analyze the independence of samples by determining whether the fitness values obtained from the 30 executions come from a normal distribution or

[2] Available at https://goo.gl/oeGvms.

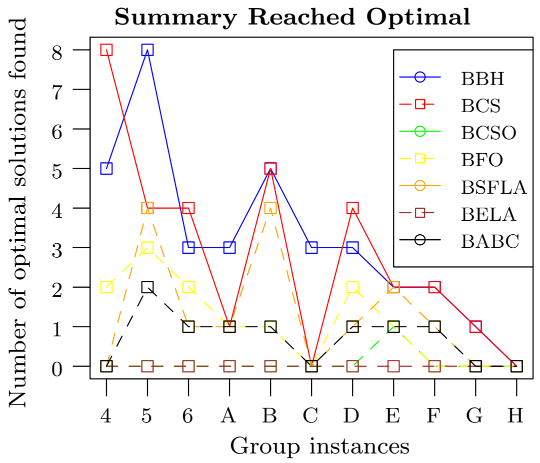


Fig. 4 Summary of reached optimal ($RPD = 0$)

they are independent. To proceed we consider two hypothesis: H_0 and H_1. H_0 states that fitness values follow a normal distribution and H_1 otherwise. The conducted test has yielded *p*values lower than 0.05, therefore we cannot assume H_0.

Then, as samples are independent and do not follow a normal distribution, it is not feasible to use the central limit theorem to approximate the distribution of the sample mean as Gaussian. Therefore, we assume the use of a non-parametric test for evaluating the heterogeneity of samples. Under this assumption, we can use the *Wilcoxon's signed rank* test for matched pairs. This is a paired test that compare the medians of two distributions. To proceed, we consider again two hypothesis. $H_0 : \tilde{X}_{BBH} \leq \tilde{X}_{BCS}$, where $\tilde{X}$ corresponds to the arithmetic median of fitness values achieved; and $H_1 : \tilde{X}_{BCS} < \tilde{X}_{BBH}$.

Tables 12, 13, 14 and 15 compare the algorithms for all tested instances via the *Wilcoxon's signed rank* test. As the significance level is also established to 0.05, smaller values that 0.05 defines that H_0 cannot be assumed. Bold font is used for a winner value of the metaheuristic stated in the column of the table, e.g. for instance 4.1, BCS is better than BHH as its value is lower than 0.05, then H_0 cannot be assumed.

According to results, p values lower than 0.05 are equivalent for both metaheuristics (24 for each one) illustrating again a very similar performance for all tested instances. However we may note that BHH performs clearly better than BCS for the E group, which is the hardest one (Table 16).

8 Conclusions and future work

In this paper, we have presented new BBH and BCS algorithms for solving SCPs. Both metaheuristics are quite simple to implement and can efficiently be adapted to binary

Table 11 Summary of reached optimal ($RPD = 0$)

Group instances	Total	BCS	BBH	BCSO	BFO	BSFLA	BELA	BABC
4.x	10	8	5	0	2	0	0	0
5.x	10	8	8	0	3	4	0	2
6.x	5	4	3	0	2	1	0	1
A.x	5	1	3	0	1	1	0	1
B.x	5	5	5	0	1	4	0	1
C.x	5	0.00	3	0	0	0	0	0
D.x	5	4	3	0	2	1	0	1
E.x	5	2	2	1	1	2	0	1
F.x	5	2	2	0	0	1	0	1
G.x	5	1	1	0	0	0	0	0
H.x	5	0.00	0.00	0	0	0	0	0
Total %	65	35 53.84	35 53.84	1 1.53%	12 18.46%	14 21.53%	0 0%	8 12.3%

Table 12 Exact p values obtained on instance of groups 4 and 5

4-Group				5-Group			
H_0		BBH	BCS	H_0		BBH	BCS
4.1	BBH	–	**2.1E^{-6}**	5.1	BBH	–	> 0.05
	BCS	> 0.05	–		BCS	> 0.05	–
4.2	BBH	–	**5E^{-12}**	5.2	BBH	–	> 0.05
	BCS	> 0.05	–		BCS	> 0.05	–
4.3	BBH	–	**3.2E^{-8}**	5.3	BBH	–	> 0.05
	BCS	> 0.05	–		BCS	**2.5E^{-14}**	–
4.4	BBH	–	**0.04**	5.4	BBH	–	**1.1E^{-7}**
	BCS	> 0.05	–		BCS	> 0.05	–
4.5	BBH	–	> 0.05	5.5	BBH	–	**2.1E^{-18}**
	BCS	**6.2E^{-5}**	–		BCS	> 0.05	–
4.6	BBH	–	> 0.05	5.6	BBH	–	**5E^{-12}**
	BCS	> 0.05	–		BCS	> 0.05	–
4.7	BBH	–	> 0.05	5.7	BBH	–	> 0.05
	BCS	**0.01**	–		BCS	> 0.05	–
4.8	BBH	–	> 0.05	5.8	BBH	–	**0.04**
	BCS	> 0.05	–		BCS	> 0.05	–
4.9	BBH	–	> 0.05	5.9	BBH	–	> 0.05
	BCS	**4.7E^{-5}**	–		BCS	> 0.05	–
4.10	BBH	–	> 0.05	5.10	BBH	–	> 0.05
	BCS	**0.02**	–		BCS	**0.04**	–

domains by using transfer functions and discretization methods. We have integrated to the core of both metaheuristics an effective pre-processing phase as well as multiple transfer functions and discretization methods. Preprocessing serves as a filtering phase to discard values leading to unfeasible solutions, while transfers function and discretization methods are used for efficiently handling the binary nature of the problem. We have tested 65 non-unicost instances from the Beasley's OR-Library where several global optimum values were reached by both approaches, remaining very close to optimal values when was not possible to reach them, particularly for the hardest instances. We have also compared both approaches by using statistical tests yielding very similar results in terms of performance.

Table 13 Exact *p* values obtained on instance of groups 6, A and B

6-Group				A-Group				B-Group			
H_0		BBH	BCS	H_0		BBH	BCS	H_0		BBH	BCS
6.1	BBH	–	> 0.05	**A.1**	BBH	–	> 0.05	B.1	BBH	–	**0.01**
	BCS	**9.3E^{-4}**	–		BCS	**1.1E^{-5}**	–		BCS	> 0.05	–
6.2	BBH	–	> 0.05	**A.2**	BBH	–	**2.1E^{-18}**	B.2	BBH	–	**3.2E^{-4}**
	BCS	**6.3E^{-7}**	–		BCS	> 0.05	–		BCS	> 0.05	–
6.3	BBH	–	> 0.05	**A.3**	BBH	–	**9E^{-5}**	B.3	BBH	–	> 0.05
	BCS	**1.5E^{-3}**	–		BCS	> 0.05	–		BCS	**0.02**	–
6.4	BBH	–	**8.1E^{-13}**	**A.4**	BBH	–	**2.1E^{-18}**	B.4	BBH	–	> 0.05
	BCS	> 0.05	–		BCS	> 0.05	–		BCS	> 0.05	–
6.5	BBH	–	> 0.05	**A.5**	BBH	–	> 0.05	B.5	BBH	–	> 0.05
	BCS	> 0.05	–		BCS	> 0.05	–		BCS	> 0.05	–

Table 14 Exact *p* values obtained on instance of groups C, D and E

C-Group				D-Group				E-Group			
H_0		BBH	BCS	H_0		BBH	BCS	H_0		BBH	BCS
C.1	BBH	–	> 0.05	D.1	BBH	–	> 0.05	**E.1**	BBH	–	> 0.05
	BCS	**4.8E^{-14}**	–		BCS	**6.5E^{-5}**	–		BCS	> 0.05	–
C.2	BBH	–	**2.2E^{-12}**	D.2	BBH	–	> 0.05	**E.2**	BBH	–	**1.8E^{-5}**
	BCS	> 0.05	–		BCS	**1.6E^{-16}**	–		BCS	> 0.05	–
C.3	BBH	–	**4.1E^{-14}**	D.3	BBH	–	> 0.05	**E.3**	BBH	–	> 0.05
	BCS	> 0.05	–		BCS	> 0.05	–		BCS	> 0.05	–
C.4	BBH	–	**2.1E^{-18}**	D.4	BBH	–	> 0.05	**E.4**	BBH	–	**0.04**
	BCS	> 0.05	–		BCS	> 0.05	–		BCS	> 0.05	–
C.5	BBH	–	**4.9E^{-3}**	D.5	BBH	–	> 0.05	**E.5**	BBH	–	**0.02**
	BCS	> 0.05	–		BCS	**0.03**	–		BCS	> 0.05	–

Table 15 Exact *p* values obtained on instance of groups F, G and H

F-Group				G-Group				H-Group			
H_0		BBH	BCS	H_0		BBH	BCS	H_0		BBH	BCS
F.1	BBH	–	**1.1E^{-7}**	**G.1**	BBH	–	> 0.05	H.1	BBH	–	> 0.05
	BCS	> 0.05	–		BCS	**2.7E^{-15}**	–		BCS	**2.1E^{-18}**	–
F.2	BBH	–	> 0.05	**G.2**	BBH	–	> 0.05	H.2	BBH	–	> 0.05
	BCS	> 0.05	–		BCS	**5.3E^{-14}**	–		BCS	**2.1E^{-18}**	–
F.3	BBH	–	> 0.05	**G.3**	BBH	–	**5.9E^{-10}**	H.3	BBH	–	> 0.05
	BCS	**6.7E^{-5}**	–		BCS	> 0.05	–		BCS	**2.9E^{-15}**	–
F.4	BBH	–	> 0.05	**G.4**	BBH	–	> 0.05	H.4	BBH	–	> 0.05
	BCS	**9.2E^{-9}**	–		BCS	> 0.05	–		BCS	**2.1E^{-18}**	–
F.5	BBH	–	> 0.05	**G.5**	BBH	–	**4.2E^{-14}**	H.5	BBH	–	> 0.05
	BCS	> 0.05	–		BCS	> 0.05	–		BCS	**4.2E^{-14}**	–

As future work, we plan to experiment with additional modern metaheuristics and to provide a larger comparison of recent bio-inspired techniques to solve SCPs. The integration of autonomous search to the presented approach would be another direction of research to follow as well, for instance to dynamically select the best binarization strategy during solving according to performance indicators as analogously studied in (Soto et al. 2013, 2015b, d).

Table 16 Summary of results obtained from the statistical tests

	4.1	4.2	4.3	4.4	4.5	4.6	4.7	4.8	4.9	4.10
BCS	✔	✔	✔	✔	-	-	-	-	-	-
BBH	–	–	–	–	✔	–	✔	–	✔	✔
	5.1	5.2	5.4	5.4	5.5	5.6	5.7	5.8	5.9	5.10
BCS	–	–	–	✔	✔	✔	–	✔	–	–
BBH	–	–	✔	–	–	–	–	–	–	✔
	6.1	6.2	6.3	6.4	6.5	A.1	A.2	A.3	A.4	A.5
BCS	–	–	–	✔	–	–	✔	✔	✔	–
BBH	✔	✔	✔	–	–	✔	–	–	–	–
	B.1	B.2	B.3	B.4	B.5	C.1	C.2	C.3	C.4	C.5
BCS	✔	✔	–	–	–	–	✔	✔	✔	✔
BBH	–	–	✔	–	–	✔	–	–	–	–
	D.1	D.2	D.3	D.4	D.5	E.1	E.2	E.3	E.4	E.5
BCS	–	–	–	–	–	–	✔	–	✔	✔
BBH	✔	✔	–	–	✔	–	–	–	–	–
	F.1	F.2	F.3	F.4	F.5	G.1	G.2	G.3	G.4	G.5
BCS	✔	–	–	–	–	–	–	✔	–	✔
BBH	–	–	✔	✔	–	✔	✔	–	–	–
	H1		H.2		H.3		H.4		H.5	
BCS	–		–		–		–		–	
BBH	✔		✔		✔		✔		✔	

Acknowledgements Ricardo Soto is supported by Grant CONICYT / FONDECYT/REGULAR/1160455, Broderick Crawford is supported by Grant CONICYT/FONDECYT/REGULAR/1140897, Fernando Paredes is supported by Grant CONICYT/FONDECYT/REGULAR/1130455 and Rodrigo Olivares is supported by Postgraduate Grant Pontificia Universidad Católica de Valparaíso (INF-PUCV 2015).

References

Avis D (1980) A note on some computationally difficult set covering problems. Math Program 18(1):138–145

Baker E, Bodin L, Finnegan W, Ponder R (1979) Efficient heuristic solutions to an airline crew scheduling problem. AIIE Trans 11(2):79–85

Balas E (1997) A dynamic subgradient-based branch-and-bound procedure for set covering. Locat Sci 5(3):203–203

Bartholdi J (1981) A guaranteed-accuracy round-off algorithm for cyclic scheduling and set covering. Oper Res 29(3):501–510

Beasley J (1987) An algorithm for set covering problem. Eur J Oper Res 31(1):85–93

Birbil Şİ, Fang S-C (2003) An electromagnetism-like mechanism for global optimization. J Global Optim 25(3):263–282

Breuer M (1970) Simplification of the covering problem with application to boolean expressions. J ACM 17(1):166–181

Brusco M, Jacobs L, Thompson G (1999) A morphing procedure to supplement a simulated annealing heuristic for cost and coverage correlated set covering problems. Ann Oper Res 86:611–627

Caprara A, Fischetti M, Toth P (1999) A heuristic method for the set covering problem. Oper Res 47(5):730–743

Caprara A, Fischetti M, Toth P (2000) Algorithms for the set covering problem. Ann OR 98(1–4):353–371

Caserta M (2007) Tabu search-based metaheuristic algorithm for large-scale set covering problems, In: Metaheuristics: progress in complex systems optimization. Springer, Boston, pp. 43–63. ISBN 978-0-387-71921-4

Ceria S, Nobili P, Sassano A (1998) A lagrangian-based heuristic for large-scale set covering problems. Math Prog 81:215–228

Chvatal V (1979) A greedy heuristic for the set-covering problem. Math Oper Res 4(3):233–235

Crawford B, Soto R, Monfroy E, Paredes F, Palma W (2011) A hybrid ant algorithm for the set covering problem. Int J Phys Sci 6(19):4667–4673

Crawford B, Soto R, Monfroy E (2013) Cultural algorithms for the set covering problem, In: Advances in swarm intelligence: 4th international conference, ICSI 2013, Harbin, China, June 12–15, 2013, Proceedings, Part II. Springer, Berlin, Heidelberg, pp 27–34

Crawford B, Soto R, Olivares-Suárez M, Paredes F (2014) A binary firefly algorithm for the set covering problem. In: 3rd computer science on-line conference 2014 (CSOC 2014). Advances in intelligent systems and computing, vol. 285. Springer, Cham, pp 65–73

Crawford B, Soto R, Peña C, Riquelme-Leiva M, Torres-Rojas C, Johnson F, Paredes F (2015a) Binarization methods for shuffled frog leaping algorithms that solve set covering problems, In: Proceedings of the 4th computer science on-line conference 2015 (CSOC2015), vol 3: software engineering in intelligent systems. Advances in intelligent systems and computing, vol. 349. Springer, Cham, pp 317–326

Crawford B, Soto R, Berros N, Johnson F, Paredes F, Castro C, Norero E (2015b) A binary cat swarm optimization algorithm for the non-unicost set covering problem. Math Probl Eng 2015:1–8

Crawford B, Soto R, Peña C, Palma W, Johnson F, Paredes F (2015c) Solving the set covering problem with a shuffled frog leaping algorithm. In: 7th Asian conference, ACIIDS 2015, Bali, Indonesia, March 23-25, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9012. Springer, Cham, pp 41–50

Crawford B, Soto R, Berros N, Johnson F, Paredes F (2015d) Solving the set covering problem with binary cat swarm optimization. In: Advances in swarm and computational intelligence. Lecture notes in computer science, vol. 9140. Springer, Cham, pp 41–48

Cuesta R, Crawford B, Soto R, Paredes F (2014) An artificial bee colony algorithm for the set covering problem, In: 3rd Computer science on-line conference 2014 (CSOC 2014). Advances in intelligent systems and computing, vol. 285. Springer, Cham, pp 53–63

Fisher M, Kedia P (1990) Optimal solution of set covering/partitioning problems using dual heuristics. Manage Sci 36(6):674–688

Fouladgar N, Lotfi S (2015) A novel swarm intelligence algorithm based on cuckoo search algorithm (NSICS). In: 11th International conference, ICIC 2015, Fuzhou, China, August 20–23, 2015, Proceedings, part I. Lecture notes in computer Science, vol. 9225. Springer, Cham, pp 587–596

Gass S, Fu M (2013) Set-covering problem, In: Encyclopedia of operations research and management science. Springer, Cham, pp 1393–1393

Hatamlou A (2013) Black hole: a new heuristic optimization approach for data clustering. Inf Sci 222:175–184

Karaboga D (2005) An idea based on honey bee swarm for numerical optimization, Technical Report 06, Computer Engineering Department, Erciyes University, Kayseri, Turkey

Kumar S, Datta D, Singh S (2015) Black hole algorithm and its applications, In: Computational intelligence applications in modeling and control. Studies in computational intelligence, vol. 575. Springer, Cham, pp 147–170

Lan G, DePuy G (2006) On the effectiveness of incorporating randomness and memory into a multi-start metaheuristic with application to the set covering problem. Comput Ind Eng 51(3):362–374

Lilliefors H (1967) On the kolmogorov–smirnov test for normality with mean and variance unknown. J Am Stat Assoc 62(318):399–402

Mann H, Whitney D (1947) On a test of whether one of two random variables is stochastically larger than the other. Ann Math Stat 18(1):50–60

Mirjalili S, Lewis A (2013) S-shaped versus v-shaped transfer functions for binary particle swarm optimization. Swarm Evol Comput 9:1–14

Munagala K, Babu S, Motwani R, Widom J (2004) The pipelined set cover problem. In: Database theory-ICDT 2005. Springer, Berlin, Heidelberg, pp 83–98

Nemati M, Momeni H, Bazrkar N (2013) Article: binary black holes algorithm. Int J Comput Appl 79(6):36–42

Pereira L, Rodrigues D, Almeida T, Ramos C, Souza A, Yang X-S, Papa JaP (2014) A binary cuckoo search and its application for feature selection. In: Cuckoo search and firefly algorithm. Studies in computational intelligence, vol 516. Springer, Cham, pp 141–154

Rubin J (1973) A technique for the solution of massive set covering problems, with application to airline crew scheduling. Transp Sci 7(1):34–48

Rushmeier R, Nemhauser G (1993) Experiments with parallel branch-and-bound algorithms for the set covering problem. Oper Res Lett 13(5):277–285

Salveson M (1955) The assembly line balancing problem. J Ind Eng 6:18–25

Soto R, Crawford B, Misra S, Palma W, Monfroy E, Castro C, Paredes F (2013) Choice functions for autonomous search in constraint programming: GA vs PSO. Tech Gaz 20(4):621–629

Soto R, Crawford B, Olivares R, Barraza J, Johnson F, Paredes F (2015a) A binary cuckoo search algorithm for solving the set covering problem. In: Bioinspired computation in artificial systems-international work-conference on the interplay between natural and artificial computation, IWINAC 2015, Elche, Spain, June 1–5, 2015, Proceedings, Part II, pp 88–97

Soto R, Crawford B, Palma W, Galleguillos K, Castro C, Monfroy E, Johnson F, Paredes F (2015b) Boosting autonomous search for CSPs via skylines. Inf Sci 308:38–48

Soto R, Crawford B, Palma W, Monfroy E, Olivares R, Castro C, Paredes F (2015) Top-*k* based adaptive enumeration in constraint programming. Math Probl Eng 2015:1–12

Soto R, Crawford B, Muñoz A, Johnson F, Paredes F (2015c) Pre-processing, repairing and transfer functions can help binary electromagnetism-like algorithms. In: Artificial intelligence perspectives and applications. Advances in intelligent systems and computing, vol. 347. Springer, Cham, pp 89–97

Toregas C, Swain R, ReVelle C, Bergman L (1971) The location of emergency service facilities. Oper Res 19(6):1363–1373

Valenzuela C, Crawford B, Soto R, Monfroy E, Paredes F (2014) A 2-level metaheuristic for the set covering problem. Int J Comput Commun Control 7(2):377

Walker W (1974) Using the set-covering problem to assign fire companies to fire houses. Oper Res 22(2):275–277

Yang X-S, Deb S (2014) Cuckoo search: recent advances and applications. Neural Comput Appl 24(1):169–174

Yang X-S, Deb S (2009) Search Cuckoo, via Levy flights. In: Nature biologically inspired computing, NaBIC 2009. World Congress on 2009, pp 210–214

Yelbay B, Birbil Şİ, Bülbül K (2014) The set covering problem revisited: an empirical study of the value of dual information. JIMO 11(2):575–594