Natural Computing (2020) 19:825–841
https://doi.org/10.1007/s11047-019-09752-8

A binary monkey search algorithm variation for solving the set covering problem

Broderick Crawford[1] · Ricardo Soto[1] · Rodrigo Olivares[1,2] · Gabriel Embry[1] · Diego Flores[1] · Wenceslao Palma[1] · Carlos Castro[3] · Fernando Paredes[4] · José-Miguel Rubio[5]

Published online: 11 July 2019

Abstract
In complexity theory, there is a widely studied grouping of optimization problems that belongs to the non-deterministic polynomial-time hard set. One of them is the set covering problem, known as one of Karp's 21 $\mathcal{NP}$-complete problems, and it consists of finding a subset of decision variables for satisfying a set of constraints at the minimum feasible cost. However, due to the nature of the problem, this cannot be solved using traditional complete algorithms for hard instances. In this work, we present an improved binary version of the monkey search algorithm for solving the set covering problem. Originally, this approximate method was naturally inspired by the cognitive behavior of monkeys for climbing mountains. We propose a new climbing process with a better exploratory capability and a new cooperation procedure to reduce the number of unfeasible solutions. For testing this approach, we present a detailed computational results section, where we illustrate how this variation of the monkey search algorithm is capable of reaching various global optimums for a well-known instance set from the Beasley's OR-Library and how it outperforms many other heuristics and meta-heuristics addressed in the literature. Moreover, we add a complete statistical analysis to show the effectiveness of the proposed approach with respect to the original version.

Keywords Monkey search algorithm · Set covering problem · Metaheuristics · Parameter setting · Optimization problem

1 Introduction

Over the years, many companies have seen the need to use their resources to meet the needs of a sector; thus, these companies try to always use their capital as efficiently as possible, in other words, trying to minimize the costs. Such situations can be represented by the set covering problem, which is a classic problem in combinatorics, computer science (Crawford et al. 2017), and computational complexity theory. Lots of real-world problems have been modeled by the set covering, such as production planning in industry (Vasko et al. 1987), crew scheduling in airlines (Housos and Elmroth 1997), and facility location problem (Vasko and Wilson 1984), among many others. Due to the characteristics of this problem, the complexity of this begins to increase with an increasing number of constraints (needs). The number of solutions and the time needed to check them increase. The reason is simple: it is not feasible to fix the larger instances of this problem manually using exact algorithms. In this line, studies have had to move in new ways to solve it, leading to the use of metaheuristics in order to find solutions to the problem. These new techniques generally are based on neural networks, animal behavior, human behavior or non-deterministic procedures.

✉ Broderick Crawford
broderick.crawford@pucv.cl

✉ Ricardo Soto
ricardo.soto@pucv.cl

✉ Rodrigo Olivares
rodrigo.olivares@uv.cl

1 Pontificia Universidad Católica de Valparaíso, Valparaiso, Chile

2 Universidad de Valparaíso, Valparaiso, Chile

3 Universidad Técnica Federico Santa María, Valparaiso, Chile

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

5 Universidad Tecnológica de Chile INACAP, Santiago, Chile

However, it is well known that the performance of metaheuristics depends largely on their correct parameter settings. In fact, finding the appropriate values for the parameters of an algorithm is considered to be a non-trivial task. Previous research has addressed this task that can be classified into two major forms of parameter setting: parameter tuning and parameter control. Parameter tuning is known as the process of finding good parameter values before the run of the solver –by using for instance training instances–, and then launching the algorithm using these values, which remain static during solving. On the contrary, parameter control launches the algorithm with initial parameter values which are updated during the run. This applies to some metaheuristics, for instance in evolutionary algorithms the better values of these encoded parameters may lead to better individuals, which in turn have more chances to propagate these good configurations to next generations. In this research, we improved the original version of the binary monkey search algorithm (Zhou et al. 2016b) (we called it IBMSA), and then, we design a new variation of it (IBMSAV), both for solving the set covering problem. The improved version and its variation, both were designed using the concept of online control.

Those algorithms are based on the monkey search algorithm (MA) (Zhao and Tang 2008) proposed in 2007 to solve the global numerical optimization problem. It is a swarm intelligence-based algorithm derived from simulation of the mountain-climbing processes of the monkeys when they look for food. The above algorithms are relatively new; the MA was used to solve the renewable energy problem (Ituarte-Villarreal et al. 2012), and the IBMSA was used to solve the knapsack problem (Zhou et al. 2016b), among others, but neither of them was used to solve the set covering problem. The IBMSA (and the IBMSAV) consists of four steps: Climbing Process, Watch-Jump Process, Somersault Process and Cooperation Process. These steps aim to find the optimal local solution, find better solutions than the previous step, and find a new solution domain. They ultimately improve local solution finding and accelerate the convergence order.

This research aims to understand the set covering problem and propose a new solution based on an improved variation of the binary monkey search algorithm. First, in Sect. 2, we present a literature review of the problem and several techniques to solve it. Then, the set covering problem with its mathematical model is explained in Sect. 3. Later, in Sect. 4, the IBMSA is exposed next to our proposed variation, in which each of its steps will be explained, along with mathematical models that support them, including pseudo-code to implement the algorithm. Finally, the experimental results and discussion at the conclusion of this work are shown in Sects. 5 and 6, respectively.

2 Related works

Metaheuristics are multi-purpose problem solvers devoted to particularly tackle large instances of complex optimization problems. They are commonly able to provide near-optimal solutions in a limited amount of time when no efficient problem-specific algorithm pre-exists. Most metaheuristics are inspired by interesting processes and phenomenon from nature such as the selection and evolution mechanisms of species (Holland 1975), the path seeking skills of ants (Dorigo et al. 1996), the attraction capabilities of fireflies (Yang and He 2013), or the echolocation behavior of bats (Yang 2010). All metaheuristics operate similarly and share various features. The goal is to explore promising regions of the search space in order to rapidly converge to optimal solutions. The exploration is done by multiple agents that employ proper movements, based on some parameter configuration, to reach fruitful search areas. They also use feedback information for modifying their internal operation so as to improve search performance. Unsurprisingly, the successful results given by metaheuristics added to the simplicity of computational requirements have produced a very fast growth of this research field during the last years. Indeed, there exists today more than 70 different metaheuristics and various of them have successfully participated in solving problems from different areas such as: traveling salesman problem (Zhou et al. 2015) and (Xin et al. 2017), vehicle path planning (Zhang et al. 2016), scheduling problem (Zhou et al. 2014), knapsack problem (Zhou et al. 2015, 2016a; Zhou 2016; Basset and Zhou 2018), among others (Burke et al. 2003).

Parameter setting (Roeper and Williams 1987) is one of the most studied challenges in the evolutionary computing field (Eiben et al. 1999; Salto and Alba 2011; Qin and Suganthan 2005; Yi et al. 2014; Han et al. 2012; Liang et al. 2001). However, a more reduced work can be observed in the literature when other metaheuristics are involved. For instance, a parameter adaptation study on ant colony optimization is reported in Stutzle et al. (2012), a firefly algorithm for solving the optimal capacitor placement problem is described (Olamaei et al. 2013), in Li and Yin (2012) a self-adaptive artificial bee colony for constrained numerical optimization is presented, and a modified cuckoo search algorithm for solving engineering problems can be encountered in Mahmoudi and Lotfi (2015), Nguyen and Vo (2015). Now, considering more similar works, we find a variation of genetic algorithm that adjusts its population size in Affenzeller et al. (2007). The basic idea is to adapt the actual population size depending on the difficulty of the algorithm in its ultimate goal to generate new child chromosomes that outperform their parents. In Cui et al. (2017), a recent self-adaptive version of a genetic algorithm is proposed, where differential evolution is involved in the process. Similarly, (Li and Yin 2015) presents a version of cuckoo search algorithm that adapts their parameter p_a reflecting the probability whether the nest will be abandoned or updated. In both cases, the adaptive approach depends of quality solution. Finally, in Akay and Karaboga (2012) we can observe a variation of the artificial bee colony that consists in controlling the perturbation frequency.

The set covering problem is a widely studied optimization problem in the mathematical programming literature. In this context, preliminary work proposed to solve it via exact methods, which commonly rely on branch-and-bound (Balas 1997; Beasley 1987) and branch-and-cut algorithms (Fisher and Kedia 1990). However, the main problem is that these algorithms take too long to solve the hard instances. To tackle this concern, much research in recent years has been focused on the development of new heuristics capable of finding optimal solutions in a reasonable amount of time. For instance, classical greedy algorithms are quite fast and easy to implement, although they are unable to generate high-quality solutions due to their deterministic nature (Chvatal 1979). The incorporation of memory and random components in greedy algorithms has led to better solutions (Lan and DePuy 2006). Nevertheless, Lagrangian relaxation-based heuristics are in general much more effective (Ceria et al. 1998; Caprara et al. 1999) than the abovementioned approaches. A detailed description of efficient exact methods and heuristics devoted to set covering problems can be found in Caprara et al. (2000). As top-level general search strategies, metaheuristics have also largely been applied to solve set covering problems in recent years. Various examples in this context include classic approaches, such as genetic algorithms (Yelbay et al. 2014), colony optimization (Crawford et al. 2011; Valenzuela et al. 2014), simulated annealing (Brusco et al. 1999), and a variant of the partial set covering problem using tabu search (Bilal et al. 2014). More recent metaheuristics for solving set covering problems can also be found in the literature, such as cat swarm optimization (Crawford et al. 2015a), firefly algorithm (Crawford et al. 2014a), shuffled frog leaping (Crawford et al. 2015b), electromagnetism-like algorithm (Soto et al. 2015c), artificial bee colony (Crawford et al. 2014b), and interesting comparisons between the behaviors of the cuckoo search algorithm and black hole optimization (Soto et al. 2017), among other works.

3 Problem statement

The set covering problem has many applications in real life and industry, e.g., the allocation of services (ReVelle et al. 2010; Brotcorne et al. 2003), load balancing production lines (Salveson 1995), and the selection of files in a database (Day 1965), among others. Many decision-making contexts can be described using an approach based on this problem, mainly due to its binary nature that can be associated with an assignment or a decision with only two responses. This inference would allow for improving, e.g., the quantitative and qualitative performance of the assets of the entity in charge, letting them be used in a better way.

In previous and recent works, this problem has been described through the use of a set of variables, a set of constraints, and the active relationship that exists between them. The aim is finding a minimum subset of variables for satisfying the set of all the constraints. Formally, in complexity theory, the set covering problem is considered as a classic combinatorial problem that belongs to the $\mathcal{NP}$-Completeness class (Hartmanis 1982), and it was formulated as a binary linear problem, according to the following decision variable:

$$x_j = \begin{cases} 1, & \text{if the variable } j \text{ is part of the solution.} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

It is possible to consider at least two ways to work with this problem: unicost and non-unicost. The unicost variant states that the cost for including a decision variable is equal to 1, for all them. On the other hand, the non-unicost variant considers that the decision variables could have a different inclusion value.

As mentioned, a feasible solution for this problem is given by a subset of decision variables that covers all of the constraints. Under this line of thought, an instance of the set covering problem can be resolved for many solutions, it being necessary to perform evaluating the quality of solutions to know which is the best. This evaluation is carried out through a linear function on the sum of the costs for all variables of the solution. This function is called the objective function and is defined as follows:

$$\texttt{minimize} \sum_{j=1}^{N} c_j x_j \tag{2}$$

where N is the number of decision variables and c_j is the cost of each them. To solve for a unicost instance, $c_j = 1, \forall j \in \{1, \ldots, N\}$.

The specific problem seeks to find feasible solutions at the lowest possible cost. The feasibility of a solution is tested using all constraints. If at least one decision variable that is part of the solution $(x_j = 1)$ covers each one of the constraints, then it is possible to state that the solution is feasible. The formula for this is given by:

$$\sum_{j=1}^{N} a_{ij}x_j \geq 1, \forall i = \{1, \ldots, M\} \tag{3}$$

where N is the total number of decision variables, a_{ij} is the coefficient of the decision variable in the constraint, and M represents the number of constraints to satisfy.

To clarify the set covering problem, we firstly expose the following example.

Four senior developers work in a software company. Each of them uses at least one programming language. The first one knows C, C++, and Python; the second one understands C++ and Java; and the third one comprehends C++, Ruby, and Python. Finally, the fourth one works with C, Java, and Ruby (see Table 1).

The main idea is to put together a team subject to two requirements:

1. The team must be made up of at least one person who knows each language (i.e., C, C++, Java, Python, and Ruby).
2. The team should be as small as possible.

To solve this instance, a binary decision variable is proposed:

$$x_j = \begin{cases} 1, & \text{if the senior developer } j \text{ is part of the team.} \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

To satisfy the first requirement, it is necessary to include senior developers, ensuring that the team knows all the programming languages. To represent the relationship between the senior developers—decision variables—and the programming languages – constraints—we use an assignment zero-one matrix (Feo and Resende 1989). In this matrix, M and N represent the number of programming languages and the number of senior developers, respectively. For instance, according to Row 1, we can determine that there are two senior developers that know the C language.

Table 1 List of senior developers and their knowledge of programming languages

Programming languages	Senior developers (*sd*)			
	sd_1	sd_2	sd_3	sd_4
C	✔			✔
C++	✔	✔	✔	
Java		✔		✔
Python	✔		✔	
Ruby			✔	✔

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \tag{5}$$

To cover the second requirement, we should minimize the number of senior developers that are part of the team. For that, the objective function is used for evaluating the quality of each solution. Finally, the integer programming model is as follows:

$$\begin{aligned} &\texttt{minimize} \sum_{j=1}^{4} c_j x_j \\ &\texttt{subjectto} \\ &x_1 + x_4 \geq 1 \\ &x_1 + x_2 + x_3 \geq 1 \\ &x_2 + x_4 \geq 1 \\ &x_1 + x_3 \geq 1 \\ &x_3 + x_4 \geq 1 \end{aligned} \tag{6}$$

The first three senior developers get to know all programming languages from a set cover of size 3. However, if we consider that the team is formed only by the third and fourth senior developers, we get a set cover of size 2, which is optimal. This solution is represented by the binary vector $\mathbf{X} = \langle 0, 0, 1, 1 \rangle$.

We can transform this example into a non-unicost instance. For that, it is necessary to differentiate the cost of each senior developer. The component c_j is the cost vector associated to each senior developer $x_j, \forall j \in \{1, \ldots, 4\}$. If we consider different costs for each senior developer, it is possible that the solution—team—changes.

4 Monkey search algorithm

The monkey search algorithm was proposed in 2007 and was recently improved to solve numerical optimization problems as a new swarm intelligence-based algorithm inspired from the mountain-climbing behavior of monkeys when they look for new food sources (Zhou et al. 2016b). Algorithm 1 describes the primary process to learn how it works.

Algorithm 1 Monkey Search Algorithm

```
begin
  initialPopulation()
  do
    do
      climbingProcess()
      watchJumpProcess()
    while(isMaximum$Nw$Reached())
    cooperationProcess()
    somersaultProcess()
    repairerAndRedundancyReductionProcess()
  while(isStopCriteriaReached())
end
```

The monkey algorithm consists of the following steps: the Climbing Process, the Watch-Jump Process, and the Somersault Process. Assume that there are many mountains in a certain terrain. At the beginning, the monkeys will climb up the nearest mountain from their initial positions in order to find the mountaintops (Climbing Process).

When the monkey gets to the top of its mountain, it will find a higher one within a certain range (called the monkey sight), and it jumps from its current position to the new mountain (Watch-Jump Process) and then repeats the Climbing Process. After repeating the processes mentioned above a certain number of times, each monkey will somersault to a new search domain to find a much higher mountain (Somersault Process).

In 2015, a variation of the MA was proposed to solve the 0–1 Knapsack Problem (Zhao and Tang 2008). The improved binary monkey search algorithm (IBMSA) includes the same three steps that its predecessor has but also includes two new steps: the Greedy Strategy and the Cooperation Process. The greedy algorithm is used to correct the unfeasible solutions and to improve the quality of feasibility; in other words, it is a repair algorithm. The Somersault Process is modified to avoid falling into local search; the Cooperation Process is implemented to speed up the convergence rate. Additionally, we implement a redundancy reduction process, which helps to improve the quality of the solutions.

In this work, we present a new variation of the IBMSA that provides a better exploration method for finding potential high-quality solutions. This algorithm has the same steps as the former, but the Climbing and Cooperation Processes have been modified. The main idea is correcting those solutions with variables outside of the binary domain, applying a transformation function that uses a stochastic procedure inspired by the comparative on sigmoid function versus a uniform distribution. The steps must satisfy our proposal as described in Sects. 4.1–4.9.

4.1 Coding method

The parameter M_{pop} is defined as the population size of monkeys. For a certain monkey i, its position is denoted as a vector $\mathbf{X}_i = \langle x_{i1}, \ldots, x_{iN} \rangle$, and this position will be employed to express a solution of the set covering problem, where $x_{ij} \in \{0, 1\}$ and $j \in \{1, \ldots, N\}$ represent the decision variables.

4.2 Initial population

In the improved binary monkey search algorithm and in the proposed variation, the initial population is randomly generated. The random initialization process for M_{pop} monkeys—potential solutions—and N decision variables is showed in Algorithm 2, where x_{ij} represents the jth decision variable of the ith solution.

4.3 Climbing process

According to the idea of a pseudo-gradient-based simultaneous perturbation stochastic approximation (SPSA) (Spall 1992), this process is a step-by-step procedure to improve the objective function value by choosing the best of two positions that are generated around the current one. For monkey i, $\mathbf{X}_i$ is the solution vector, and the objective function value is given by $f(\mathbf{X}_i)$. The Climbing Process used in the IBMSA is given as follows:

1. Randomly generate two vectors $\mathbf{X}_i^{'} = \langle x_{i1}^{'}, \ldots, x_{iN}^{'} \rangle$ and $\mathbf{X}_i^{''} = \langle x_{i1}^{''}, \ldots, x_{iN}^{''} \rangle$, and each decision variable $\{x_{ij}^{'}, x_{ij}^{''}\}$ is initialized with α or $-\alpha$ value; take it according to an uniform probability. The parameter α is called a *step of the Climbing Process*, and it depends on the situation, but its absolute value is always greater than zero ($|\alpha| > 0$).

Algorithm 2 Initial Population Process

```
for all i from 1 to $M_{pop}$ do
    for all j from 1 to $N$ do
        Randomly generate $r$
        if $r < 0.5$ then
            $x_{ij} \leftarrow 0$
        else
            $x_{ij} \leftarrow 1$
        endif
    endfor
endfor
```

2. Let $x'_{ij} \leftarrow |x_{ij} - x'_{ij}|$, and $x''_{ij} \leftarrow |x_{ij} - x''_{ij}|$, $\forall i \in \{1, \ldots, M_{pop}\} \wedge \forall j \in \{1, \ldots, N\}$. If $\mathbf{X}'_i$ is unfeasible or $\mathbf{X}''_i$ is unfeasible, X_i should be kept unchanged and go back to step (1).
3. Calculate $f(\mathbf{X}'_i)$ and $f(\mathbf{X}''_i)$.
4. If $f(\mathbf{X}'_i) < f(\mathbf{X}''_i)$ and $f(\mathbf{X}'_i) < f(\mathbf{X}_i)$, then $\mathbf{X}_i \leftarrow \mathbf{X}'_i$. If $f(\mathbf{X}''_i) < f(\mathbf{X}'_i)$ and $f(\mathbf{X}''_i) < f(\mathbf{X}_i)$, then $\mathbf{X}_i \leftarrow \mathbf{X}''_i$.
5. Repeat steps (1) to (4) until the maximum allowable number of iterations has been reached. This limit is denoted by Nc.

In the IBMSA variation proposed, step (2) is modified in order to improve the search for new solutions. This approach is inspired by the classic and widely known method for generating binary initial populations. Therefore, instead of keeping X_i unchanged if the generated solution is unfeasible, we "fix" it. Thus, set $x'_{ij} \leftarrow |x_{ij} - x'_{ij}|$ and $x''_{ij} \leftarrow |x_{ij} - x''_{ij}|$, $\forall i \in \{1, \ldots, M_{pop}\} \wedge \forall j \in \{1, \ldots, N\}$. If $\mathbf{X}'_i$ contains variables outside the domain or $\mathbf{X}''_i$ contains variables outside the domain – unfeasible—a random number is generated with uniform distribution $\varphi \sim (0, 1)$. If $\varphi < \frac{1}{1 + e^{-x_{ij}}}$, then $x_{ij} \leftarrow 0$, or else, $x_{ij} \leftarrow 1$.

4.4 Watch jump process

When the monkey reaches the top of the mountain, it will try to find a higher point than its current position that is within its sight. If it discovers one, then it will jump to the higher point, and then it will start the Climbing Process again. For the ith monkey, its position is given by $\mathbf{X}_i = \langle x_{i1}, \ldots, x_{iN} \rangle$, $\forall i \in \{1, \ldots, M_{pop}\}$. The Watch-Jump Process is given as follows:

1. Randomly generate a real vector $\mathbf{Y}$ in the interval $[x_{ij} - b, x_{ij} + b]$, $\forall j \in \{1, \ldots, N\}$ where b is the monkey's sight, and it can be determined depending of the situation, i.e., $\mathbf{Y} = \langle y_1, \ldots, y_N \rangle$.
2. Due to the real value $y_j \in (0, 1)$, randomly generate a real number with uniform distribution $\phi \sim (0, 1)$ and compare it with y_j. If $y_j < \phi$; then, $y_j \leftarrow 0$, or else, $y_j \leftarrow 1$.
3. Calculate $f(\mathbf{Y})$.
4. If $f(\mathbf{Y}) < f(\mathbf{X}_i)$, then $\mathbf{X}_i \leftarrow \mathbf{Y}$.
5. Repeat steps (1) to (4) until the maximum allowable number of iterations has been reached. This limit is denoted by Nw.

4.5 Greedy strategy: repair process

In solving the set covering problem, some monkeys may have an abnormal encode (that does not meet the constraints). For this reason, the local search strategy-greedy algorithm is implemented and run. The main idea is repairing the unfeasible solutions to improve feasibility. Assuming the monkey $\mathbf{X}_i$ is not a feasible solution, then $\mathbf{X}_i = \langle x_{i1}, \ldots, x_{iN} \rangle$. The greedy algorithm is given as follows:

1. Search for an uncovered row according to the constraint zero-one matrix.
2. Create a vector with possible candidates to cover the previously mentioned row.
3. For each candidate, calculate its weight.

$$W = \frac{\text{Number of rows covered by the candidate}}{\text{Cost of the candidate}} \tag{7}$$

4. Select the candidate with the highest W, and put it in the monkey $\mathbf{X}_i$.
5. Repeat steps (1) to (4) until the monkey $\mathbf{X}_i$ becomes feasible.

4.6 Redundancy reduction process

Once the monkeys have gone through the Repair Process, the Redundancy Reduction Process is initiated (Lanza-Gutierrez et al. 2017) to reduce the overall cost of the monkey. For each monkey i, $\mathbf{X}_i = \langle x_{i1}, \ldots, x_{iN} \rangle$, the redundancy reduction process is given as follows:

1. Set a new vector $\mathbf{Y} = \langle y_1, \ldots, y_N \rangle$.
2. Initialize $y_k \leftarrow 0, \forall k \in \{N, \ldots, 1\}$.
3. Then, check the feasibility of $\mathbf{Y}$. If $\mathbf{Y}$ is still feasible, it means that the column is redundant; thus, let $x_{ik} \leftarrow 0$ and decrease k in 1. Otherwise ($\mathbf{Y}$ is not feasible), $y_k \leftarrow 1$, and decrease k in 1.
4. Repeat step (1) to (3) for each monkey until there is no redundancy.

4.7 Cooperation process

After the Climb and Watch-Jump Process, each monkey will arrive at the highest mountain in his neighborhood. However, the highest mountain will differ across all the monkeys. The purpose of the cooperation process is to improve the solution by cooperating with the monkey that has the higher mountain (best position), i.e., they will move along the direction of the best monkey. This process can speed up the convergence rate. The optimal position is described by $\mathbf{X}^* = \langle x_1^*, \ldots, x_N^* \rangle$. For the monkey i, $\mathbf{X}_i = \langle x_{i1}, \ldots, x_{iN} \rangle$, $\forall i \in \{1, \ldots, M_{pop}\}$. The cooperation process is given as follows:

1. Randomly generate a real number with a uniform distribution $\varphi \sim (0, 1)$.
2. If $\varphi < \dfrac{1}{1 + e^{-y_j}}$, then $y_j \leftarrow x_j$, or else, $y_j \leftarrow x_j^*$, $\forall j \in \{1, \ldots, N\}$.
3. Update the monkey's position, $\mathbf{X}_i \leftarrow \mathbf{Y}$.

   In the IBMSA variation proposed in this work, new steps to the Cooperation Process are added. These new steps are based on the scout bee phase in an artificial bee colony (ABC) algorithm (Karaboga and Basturk 2007), which uses a number of trials for improving a solution, where the number is specified by the parameter "limit" L. For the IBMSA variation, this improvement allows for checking if $f(\mathbf{Y})$ have a lower cost than $f(\mathbf{X})$ before replacing the values. Otherwise, the process is repeated if, after a determined number of iterations, the algorithm cannot find a better $f(\mathbf{Y})$, and $\mathbf{X}_i$ is maintained.
4. Calculate $f(\mathbf{Y})$ and $f(\mathbf{X}_i)$. If $f(\mathbf{Y}) < f(\mathbf{X}_i)$, then the monkey $\mathbf{X}_i$ is updated with $\mathbf{Y}$; otherwise, do nothing.
5. Repeat steps (1) to (4) until $\mathbf{X}_i$ is updated or the limit L is reached.

4.8 Somersault process

Monkeys will reach their highest mountaintops around their initial positions after repetitions of the Climb, Watch-Jump and Cooperation Process. To find a higher mountain, each monkey will somersault to a new search domain. The new position is not arbitrary, and it is limited to a certain region by the pivot and the somersault interval. This process can efficiently prevent monkeys falling into a local search; however, after many iterations, the somersault process may lose efficacy, resulting in monkeys falling into the optimal local domain, decreasing the population diversity. In the original MA, the monkeys will somersault along the direction pointing to the barycenter of the current positions for all monkeys. Here, we randomly choose a position of a monkey as the pivot replacing the one proposed by the original algorithm. For each monkey i, $\mathbf{X}_i = \langle x_{i1}, \ldots, x_{iN} \rangle$, $\forall i \in \{1, \ldots, M_{pop}\}$, the Somersault Process is given as follows:

1. Randomly generate a real number θ from the interval (c, d) (this interval governs the maximum distance that monkeys can somersault, and is called the somersault interval).
2. Randomly generate an integer k, with $k \in \{1, \ldots, M_{pop}\}$. The kth monkey $(\mathbf{X}_k)$ will be the somersault pivot.
3. Calculate $y_j \leftarrow x_{kj} + \theta(x_{kj} - x_{ij})$.
4. Randomly generate a real number with a uniform distribution $\phi \sim (0, 1)$ and compare it with y_j. If $y_j < \phi$; then, $y_j \leftarrow 0$, or else, $y_j \leftarrow 1$. Update the positions of monkeys $\mathbf{X}_i \leftarrow \mathbf{Y}$.

After repetitions of the Somersault Process, monkeys may reach the same domain, making the Somersault Process lose efficacy. In this case, we set a parameter called "limit" (equal to in ABC) to handle the case where all monkeys run into the optimal local solution. If a predetermined number of trials does not improve the optimal global solution, the monkeys are abandoned and then reinitialized.

4.9 Termination condition

Following the six steps of this algorithm, all monkeys will be ready for their next action. The condition for terminating the IBMSA and our variation is when the maximum number of iterations is reached.

5 Experimental results

Runs of both algorithms, IBMSA and IBMSAV, were done for the purpose to compare them. They were implemented in the Java programming language, and they were carried out on a Windows 8.1 operating system with a Intel Core i3 2.50 GHz processor with 16 GB of RAM. For both algorithms, the parameter initial values were given in a sampling phase where we solve a instance of the set covering problem. Then, the best setting is as follows: $Iterations = 5000$, $M_{pop} = 20$, $L = 50$, $\alpha = 1$, $b = 1$, $c = -1$, $d = 1$, and $Nc = Nw = Iterations \times 1\%$.

The instances resolved were taken from OR-library (Beasley 2018). Table 2 describes the group of instance sets, number of rows or constraints (M), number of columns or variables (N), the range of costs, density (percentage of non-zeroes in the matrix) and if the optimal solution is known or unknown.

The results are evaluated using the relative percentage deviation (RPD). The RPD value quantifies the deviation of the objective function value Z_{min} from Z_{opt} that in our experiment is the minimal best-known value for each instance, and it is calculated as follows:

$$RPD = \left(\frac{Z_{min} - Z_{opt}}{Z_{opt}}\right)100 \tag{8}$$

If the RPD value reaches zero percent, we state that the algorithm has found the optimal value.

5.1 IBMSAV versus other optimization techniques

To evaluate the performance of our approach, we perform a comparison with different approximation techniques: binary cat swarm optimization (BCSO) (Crawford et al. 2015a), binary firefly optimization (BFO) (Crawford et al. 2014a), binary shuffled frog leaping algorithm (BSFLA) (Crawford et al. 2015b), binary artificial bee colony (BABC) (Crawford et al. 2014b) and binary electromagnetism-like algorithm (BELA) (Soto et al. 2015c).

Tables 3, 4, 5, and 6 illustrate that the proposed variation is able to obtain competitive results in contrast to those modern optimization techniques.

Evaluating group 4, we can see that our approach is able to find all optimum values, while BFO finds only two optimum values. For other metaheuristics, these instances were hard to solve.

If we only consider group 5, it is possible to observe that BFO, BSFLA, and BABC achieve three, three, and two optimal values, respectively. Now, analyzing BCSO and BELA, we see they show a poor performance, obtaining zero optimal values. However, for this group, our approach can find all the best-known values again.

Comparing the results for groups 6, A and B, we can state that the proposed variation works better than other bio-inspired optimization algorithms, as these others solve only some instances, in contrast to IBMSAV, which resolves 100% of cases.

For group C and D, the IBMSAV shows outstanding behavior, obtaining 4 out of 5 and 5 out of 5 optimum values, respectively. In contrast, approximate methods do not resolve more than two instances each.

Finally, for the largest instances—from group NRE to group NRH—the variation of the algorithm obtains better values than their competitors.

5.2 Monkey search algorithms comparison

In this section, we compare the improved binary monkey search algorithm (IBMSA) with the proposed variation (IBMSAV), analyzing their performance and efficiency. Tables 7 and 8 illustrate the results obtained for all

Table 2 Instances of set covering problem taken from the Beasley's OR-Library

Instance group	Instance data				
	M	N	Cost range	Density %	Best known Z_{opt}
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
NRE	500	5000	[1, 100]	10	Unknown (except **NRE.1**)
NRF	500	5000	[1, 100]	20	Unknown (except **NRF.1**)
NRG	1000	10,000	[1, 100]	2	Unknown (except **NRG.1**)
NRH	1000	10,000	[1, 100]	5	Unknown

Table 3 Results for instance set of groups 4 and 5

	Z_{opt}	Instance 4									
		1 429	2 512	3 516	4 494	5 512	6 560	7 430	8 492	9 641	**10** 514
IBMSAV	Z_{min}	**429**	**512**	**516**	**494**	**512**	**560**	**430**	**492**	**641**	**514**
	Z_{avg}	430	514	**516**	502	515	561	**430**	497	645	516
	RPD	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
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	0.00	0.97	0.58	0.2	0.39	0.53	0.00	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

	Z_{opt}	Instance 5									
		1 253	2 302	3 226	4 242	5 211	6 213	7 293	8 288	9 279	**10** 265
IBMSAV	Z_{min}	**253**	**302**	**226**	**242**	**211**	**213**	**293**	**288**	**279**	**265**
	Z_{avg}	**253**	307	229	**242**	213	**213**	**293**	**288**	281	267
	RPD	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
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	0.00	0.00	0.00	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	0.00	0.00	0.00	1.37	1.04	0.72	0.00
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	0.00	0.00	0.47	1.71	0.35	0.36	0.75

Table 4 Results for instance groups 6, A, B, and C

	Z_{opt}	Instance 6					Instance A				
		1	2	3	4	5	1	2	3	4	5
		138	**146**	**145**	**131**	**161**	**253**	**252**	**232**	**234**	**236**
IBMSAV	Z_{min}	**138**	**146**	**145**	**131**	**161**	**253**	**252**	**232**	**234**	**236**
	Z_{avg}	**138**	**146**	148	132	163	**253**	253	234	239	241
	RPD	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
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	0.00	0.68	1.37	0.00	1.86	0.79	2.77	2.58	0.42	0.00
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	0.00	3.11	0.79	3.17	2.16	0.43	0.00
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	0.00	2.48	0.40	1.98	1.29	0.85	0.00

	Z_{opt}	Instance B					Instance C				
		1	2	3	4	5	1	2	3	4	5
		69	76	80	79	72	227	219	243	219	215
IBMSAV	Z_{min}	**69**	**76**	**80**	**79**	**72**	**227**	**219**	244	**219**	**215**
	Z_{avg}	73	79	84	83	**72**	229	**219**	248	221	217
	RPD	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.004	**0.00**	**0.00**
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	0.00	1.26	0.00	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	0.00	0.00	0.00	0.00	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	0.00	1.27	0.00	1.76	1.37	4.53	5.48	0.47

Table 5 Results for instance groups D, NRE, NRF, and NRG

	Z_{opt}	Instance									
		Instance D					Instance NRE				
		1 60	2 66	3 72	4 62	5 61	1 29	2 30	3 27	4 28	5 28
IBMSAV	Z_{min}	**60**	**66**	**72**	**62**	**61**	**29**	**30**	28	**28**	**28**
	Z_{avg}	63	69	77	**62**	**61**	**29**	31	28	**28**	**28**
	RPD	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.037	**0.00**	**0.00**
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	0.00	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	0.00	3.03	4.16	0.00	3.27	0.00	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	0.00	1.52	4.17	1.61	3.28	0.00	3.33	3.7	3.57	0.00
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	0.00	3.03	5.56	1.61	3.28	0.00	6.67	7.41	3.57	3.57

	Z_{opt}	Instance									
		Instance NRF					Instance NRG				
		1 14	2 15	3 14	4 14	5 13	1 176	2 154	3 166	4 168	5 168
IBMSAV	Z_{min}	**14**	**15**	**14**	**14**	**13**	**176**	156	**166**	171	169
	Z_{avg}	15	**15**	16	17	14	177	156	169	171	169
	RPD	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.012	**0.00**	0.029	0.005
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	0.00	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	0.00	6.67	14.29	7.14	15.38	3.98	5.19	4.82	4.17	6.55

Table 6 Results for instances of group NRH

	Z_{opt}	Instance NRH				
		1 63	2 63	3 59	4 58	5 55
IBMSAV	Z_{min}	65	67	64	63	62
	Z_{avg}	65	67	65	63	62
	RPD	0.032	0.063	0.085	0.086	0.127
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

instances of the set covering problem. We use bold font when our approach (IBMSAV) finds the optimal value, and we underline the better value of each algorithm.

It is possible to see that the proposed approach is more effective than IBMSA. IBMSAV resolves correctly 55 of 65 instances (84.62%), while IBMSA does not overcome 45% of cases. Now, in 33 instances, IBMSAV has a better yield than IBMSA. In only one test does IBMSA perform better (NRG.4), but the difference is not significant.

If we consider the times required for reaching the solutions, we may observe that the times are very similar for the two algorithms. However, there is a small difference in favor of the IBMSA, with respect to IBMSAV, but this can be explained because extra time is required to recalculate the Climbing and Cooperation Processes.

Now, in order to show a significant difference between the improved binary monkey search algorithm and its variation, we perform a contrast statistical test for each instance through the *Kolmogorov–Smirnov–Lilliefors* test to determine the independence of samples (Lilliefors 1967) and the *Wilcoxon's signed rank* test (Mann and Donald 1947) to compare the results statistically.

For both tests, we consider a hypothesis evaluation, which is analyzed assuming a *p_value* of 0.05, i.e., values smaller than 0.05 mean that the corresponding hypothesis cannot be assumed. Both tests were conducted using *GNU Octave* (Eaton 2018).

The first test allows for us to analyze the independence of samples by determining whether the Z_{min} obtained from the 30 executions of each instance come from a normal distribution or whether they are independent. To proceed, we propose the following hypotheses:

- H_0: states that Z_{min} follows a normal distribution.
- H_1: states the opposite.

The conducted test has yielded *p_value* lower than 0.05; therefore, H_0 cannot be assumed.

Then, as samples are independent and it cannot be assumed that they 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. For that, we use the *Wilcoxon's signed rank* test. This is a paired test that compare the medians of two distributions. To proceed, we propose the following new hypotheses:

- H_0: $\tilde{Z}_{min}$ achieved by IBMSA is better than $\tilde{Z}_{min}$ achieved by IBMSAV.
- H_1: states the opposite.

Table 9 compares the improved optimization algorithm versus the improved variation approach for all tested instances via the *Wilcoxon's signed rank* test. As the significance level is also established to be 0.05, smaller values than 0.05 mean that H_0 cannot be assumed. A bold font is used to indicate a better value of the metaheuristic stated in the column of the table. As an example, for instance 4.1, the improved variation is better than the improved version as its value is lower than 0.05; thus, H_0 cannot be assumed. In the other cases, there is not enough information.

According to the results, for *p_values* lower than 0.05, for the improved optimization algorithm, there are 10 such results; for the improved variation approach, there are 36 results. The rest of the tests do not provide significant information because *p_values* are greater than 0.05 but less than 0.95. These results illustrate that the performance of the improved variation is better than the improved binary monkey search algorithm.

6 Conclusions

The set covering problem is a well-known NP-hard problem of combinatorial analytics. This problem consists in finding solutions covering the needs at a lower cost. Those needs can be services to cities, load balancing in production lines or data-bank selections. In this work, we study the solution of this problem through an algorithm based on swarm intelligence inspired from the mountain-climbing behavior of monkeys, called the improved binary monkey

Table 7 Instances of set covering problem taken from the Beasley's OR-Library (Groups 4, 5, 6, A, B, and C)

Instance	Z_{opt}	Monkey search algorithm					
		IBMSA			IBMSAV		
		Z_{min}	*RPD*	Times (ms)	Z_{min}	*RPD*	Times (ms)
4.1	429	430	0.002	1009.2	**429**	0.00	1167.2
4.2	512	512	0.00	1011.2	**512**	0.00	1021.3
4.3	516	516	0.00	1701.5	**516**	0.00	1822.4
4.4	494	495	0.002	1837.4	**494**	0.00	1982.3
4.5	512	514	0.004	2014.3	**512**	0.00	2412.5
4.6	560	560	0.00	2314.4	**560**	0.00	2402.4
4.7	430	430	0.00	2424.4	**430**	0.00	2443.3
4.8	492	494	0.004	2921.0	**492**	0.00	3332.5
4.9	641	644	0.005	2813.2	**641**	0.00	3015.5
4.10	514	515	0.002	2983.4	**514**	0.00	3523.3
5.1	253	253	0.00	2943.9	**253**	0.00	3914.6
5.2	302	305	0.01	3112.3	**302**	0.00	3344.3
5.3	226	228	0.00	2935.3	**226**	0.00	3013.4
5.4	242	243	0.004	2993.4	**242**	0.00	4002.3
5.5	211	211	0.00	2650.3	**211**	0.00	3224.3
5.6	213	213	0.00	3352.4	**213**	0.00	3932.4
5.7	293	293	0.00	2934.5	**293**	0.00	4034.6
5.8	288	290	0.007	3043.4	**288**	0.00	3440.1
5.9	279	281	0.007	4224.6	**279**	0.00	3922.1
5.10	265	265	0.00	3881.4	**265**	0.00	4162.4
6.1	138	140	0.014	4434.7	**138**	0.00	4941.3
6.2	146	147	0.007	4110.2	**146**	0.00	3974.2
6.3	145	145	0.00	3712.2	**145**	0.00	4672.3
6.4	131	131	0.00	5542.3	**131**	0.00	5601.3
6.5	161	161	0.00	4872.3	**161**	0.00	5721.3
A.1	253	253	0.00	3324.4	**253**	0.00	4454.1
A.2	252	253	0.004	4879.6	**252**	0.00	4212.4
A.3	232	233	0.004	4314.5	**232**	0.00	5739.1
A.4	234	234	0.00	4808.5	**234**	0.00	5403.4
A.5	236	236	0.00	5013.3	**236**	0.00	4973.5
B.1	69	69	0.00	4944.3	**69**	0.00	5924.3
B.2	76	76	0.00	4302.3	**76**	0.00	4487.1
B.3	80	80	0.00	4435.3	**80**	0.00	4469.3
B.4	79	79	0.00	4962.1	**79**	0.00	5429.4
B.5	72	72	0.00	4933.3	**72**	0.00	5151.3
C.1	227	229	0.009	5641.1	**227**	0.00	5719.6
C.2	219	219	0.00	4991.4	**219**	0.00	4974.5
C.3	243	245	0.008	5090.6	244	0.004	4913.5
C.4	219	219	0.00	4983.7	**219**	0.00	4339.4
C.5	215	216	0.004	4776.6	**215**	0.00	4981.2
D.1	60	60	0.00	4621.9	**60**	0.00	4833.6
D.2	66	67	0.015	5034.2	**66**	0.00	5626.7
D.3	72	73	0.014	5809.8	**72**	0.00	5935.6
D.4	62	62	0.00	5083.7	**62**	0.00	5813.5
D.5	61	61	0.00	6112.5	**61**	0.00	6134.3

Table 8 Instances of set covering problem taken from the Beasley's OR-Library (Groups D, NRE, NRF, NRG, and NRH)

Instance	Z_{opt}	Monkey search algorithm					
		IBMSA			IBMSAV		
		Z_{min}	*RPD*	Times (ms)	Z_{min}	*RPD*	Times (ms)
NRE.1	29	30	0.034	6011.4	**29**	0.00	6782.4
NRE.2	30	31	0.033	6577.4	**30**	0.00	7222.4
NRE.3	27	28	0.037	6888.5	28	0.037	7184.2
NRE.4	28	29	0.036	6891.1	**28**	0.00	6775.4
NRE.5	28	28	0.00	6801.7	**28**	0.00	7615.1
NRF.1	14	14	0.00	6889.4	**14**	0.00	6801.4
NRF.2	15	15	0.00	7112.7	**15**	0.00	6876.5
NRF.3	14	17	0.214	6828.2	**14**	0.00	9266.3
NRF.4	14	16	0.143	7463.6	**14**	0.00	10053.4
NRF.5	13	14	0.077	7323.2	**13**	0.00	8247.9
NRG.1	176	182	0.034	8125.8	**176**	0.00	8541.6
NRG.2	154	160	0.039	7838.9	156	0.012	9092.8
NRG.3	166	170	0.024	8301.7	**166**	0.00	10353.8
NRG.4	168	170	0.012	8766.9	171	0.029	11012.9
NRG.5	168	170	0.012	8843.6	169	0.006	12801.4
NRH.1	63	66	0.048	9556.5	65	0.032	10223.5
NRH.2	63	67	0.063	8576.9	67	0.063	12776.5
NRH.3	59	64	0.085	12861.2	64	0.085	14395.4
NRH.4	58	64	0.103	10978.2	63	0.086	12098.4
NRH.5	55	63	0.145	9833.2	62	0.127	13791.3

search algorithm (IBMSA), and a new variation in the Climbing and Cooperation Processes (IBMSAV) providing more exploratory capabilities.

We have tested 65 non-unicost instances from Beasley's OR-Library where several global optimum values, which were not reached using the basic improved binary optimization algorithm, were achieved via the improved variation approach. Both approaches have been evaluated by nonparametric statistical tests, and the results are conclusive. Moreover, the IBMSAV was shown to be slightly better than other optimization techniques; thus, we strongly believe that this variation approach can improve with a specific configuration for each problem.

As future work, we plan to experiment on self-adaptive approaches in recent bio-inspired algorithms and to provide a more extensive comparison of techniques for solving the set covering problem. The integration of online parameter control can lead the research toward new study lines, such as dynamically selecting the best binarization strategy during solving according to performance indicators as analogously studied in Soto et al. (2013, 2015a, b). Finally, seeing the positive results of integrating the online control technique, we can think about improving internal resolution processes (local search, for example) by applying machine and deep learning (Calvet et al. 2017; Fink 2007; Iba 2018; Memeti et al. 2018).

Table 9 Statistical test for all instances of the set covering problem

	Instance									
	4.1	4.2	4.3	4.4	4.5	4.6	4.7	4.8	4.9	4.10
IBMSA	–	–	–	–	–	0.04	–	–	–	0.05
IBMSAV	0.01	0.02	–	0.02	0.01	–	0.02	–	0.04	–

	Instance									
	5.1	5.2	5.3	5.4	5.5	5.6	5.7	5.8	5.9	5.10
IBMSA	–	–	–	–	–	0.01	0.05	–	0.03	–
IBMSAV	0.01	0.03	0.01	0.01	–	–	–	0.05	–	0.05

	Instance									
	6.1	6.2	6.3	6.4	6.5	A.1	A.2	A.3	A.4	A.5
IBMSA	–	–	–	–	–	–	–	–	0.01	–
IBMSAV	0.04	0.05	0.01	–	0.01	–	0.02	0.03	–	0.05

	Instance									
	B.1	B.2	B.3	B.4	B.5	C.1	C.2	C.3	C.4	C.5
IBMSA	–	0.01	–	–	–	–	0.01	–	–	–
IBMSAV	0.02	–	0.02	0.05	–	0.05	–	–	0.03	0.02

	Instance									
	D.1	D.2	D.3	D.4	D.5	NRE.1	NRE.2	NRE.3	NRE.4	NRE.5
IBMSA	–	–	–	–	–	–	0.01	–	–	–
IBMSAV	0.05	0.05	–	–	0.01	0.03	–	0.02	0.04	0.03

	Instance									
	NRF.1	NRF.2	NRF.3	NRF.4	NRF.5	NRG.1	NRG.2	NRG.3	NRG.4	NRG.5
IBMSA	–	–	–	–	–	–	–	–	–	–
IBMSAV	0.01	0.01	–	0.01	–	0.05	–	0.01	0.01	0.03

	Instance				
	NRH.1	NRH.2	NRH.3	NRH.4	NRH.5
IBMSA	–	–	–	0.01	–
IBMSAV	0.03	0.01	–	–	0.05

Acknowledgements Broderick Crawford is supported by Grant CONICYT/FONDECYT/REGULAR/1171243. Ricardo Soto is supported by Grant CONICYT/FONDECYT/REGULAR/1190129. Rodrigo Olivares is supported by CONICYT/FONDEF/IDeA/ID16I10449, STIC-AMSUD/17STIC- 03, FONDECYT/MEC/MEC80170097 and Postgraduate Grant Pontificia Universidad Católica de Valparaíso (INF-PUCV 2015-2019).

References

Affenzeller M, Wagner S, Winkler S (2007) Self-adaptive population size adjustment for genetic algorithms. Computer aided systems theory EUROCAST 2007. Springer, Berlin, pp 820–828

Akay B, Karaboga D (2012) A modified artificial bee colony algorithm for real-parameter optimization. Inf Sci 192:120

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

Basset MA, Zhou Y (2018) An elite opposition-flower pollination algorithm for a 0–1 knapsack problem. Int J Bio Inspir Comput 11(1):46. https://doi.org/10.1504/ijbic.2018.090080

Beasley J (2018) Or-library. http://people.brunel.ac.uk/~mastjjb/jeb/orlib/scpinfo.html. Accessed 14 Feb 2018

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

Bilal N, Galinier P, Guibault F (2014) An iterated-tabu-search heuristic for a variant of the partial set covering problem. J Heuristics 20(2):143

Brotcorne L, Laporte G, Semet F (2003) Ambulance location and relocation models. Eur J Oper Res 147(3):451

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

Burke E, Kendall G, Newall J, Hart E, Ross P, Schulenburg S (2003) Handbook of metaheuristics, vol 57. International series in operations research and management science. Springer, Berlin, pp 457–474

Calvet L, de Armas J, Masip D, Juan AA (2017) Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs. Open Math 15(1):261–80

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

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

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

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

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

Crawford B, Soto R, Olivares-Suárez M, Paredes F (2014a) Advances in intelligent systems and computing. 3rd Computer science online conference 2014 (CSOC 2014), vol 285. Springer, Berlin, pp 65–73

Crawford B, Soto R, Palma W, Johnson F, Paredes F, Olguín E (2014b) Advances in swarm intelligence. Lecture notes in computer science, vol 8794. Springer, Berlin, pp 189–196

Crawford B, Soto R, Astorga G, García J, Castro C, Paredes F (2017) Putting continuous metaheuristics to work in binary search spaces. Complexity 2017:1

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

Crawford B, Soto R, Peña C, Palma W, Johnson F, Paredes F (2015b) Intelligent information and database systems. In: 7th Asian conference, ACIIDS 2015, Bali, Indonesia, March 23–25, 2015, Proceedings, Part II. Lecture notes in computer science, vol 9012. Springer, Berlin, pp 41–50

Cui L, Li G, Zhu Z, Wen Z, Lu N, Lu J (2017) A novel differential evolution algorithm with a self-adaptation parameter control method by differential evolution. Soft Comput. https://doi.org/10.1007/s00500-017-2685-5

Day RH (1965) Letter to the editor-on optimal extracting from a multiple file data storage system: an application of integer programming. Oper Res 13(3):482

Dorigo M, Maniezzo V, Colorni A (1996) The ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern 26(1):1

Eaton JW (2018) Gnu octave. https://www.gnu.org/software/octave/ (2002). Accessed 14 Feb 2018

Eiben A, Hinterding R, Michalewicz Z (1999) Parameter control in evolutionary algorithms. IEEE Trans Evol Comput 3(2):124

Feo TA, Resende MG (1989) A probabilistic heuristic for a computationally difficult set covering problem. Oper Res Lett 8(2):67

Fink M (2007) Proceedings of the Eleventh international conference on artificial intelligence and statistics, proceedings of machine learning research (PMLR, San Juan, Puerto Rico, 2007), vol 2, pp 115–122

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

Han MF, Liao SH, Chang JY, Lin CT (2012) Dynamic group-based differential evolution using a self-adaptive strategy for global optimization problems. Appl Intell 39(1):41. https://doi.org/10.1007/s10489-012-0393-5

Hartmanis J (1982) Computers and intractability: a guide to the theory of NP-completeness. SIAM Rev 24(1):90

Holland JH (1975) Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor

Housos E, Elmroth T (1997) Automatic optimization of subproblems in scheduling airline crews. Interfaces 27(5):68

Iba H (2018) Evolutionary approach to machine learning and deep neural networks. Springer, Singapore, pp 27–75. https://doi.org/10.1007/978-981-13-0200-8_2

Ituarte-Villarreal CM, Lopez N, Espiritu JF (2012) Using the monkey algorithm for hybrid power systems optimization. Proc Comput Sci 12:344

Karaboga D, Basturk B (2007) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Global Optim 39(3):459

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

Lanza-Gutierrez J, Crawford B, Soto R, Berrios N, Gomez-Pulido J, Paredes F (2017) Analyzing the effects of binarization techniques when solving the set covering problem through swarm optimization. Expert Syst Appl 70:67

Li X, Yin M (2012) Self-adaptive constrained artificial bee colony for constrained numerical optimization. Neural Comput Appl 24(3–4):723

Li X, Yin M (2015) Modified cuckoo search algorithm with self adaptive parameter method. Inf Sci 298:80

Liang KH, Yao X, Newton CS (2001) Adapting self-adaptive parameters in evolutionary algorithms. Appl Intell 15(3):171. https://doi.org/10.1023/a:1011286929823

Lilliefors H (1967) On the Kolmogorov-Smirnov test for normality with mean and variance unknown. J Am Stat Assoc 62(318):399

Mahmoudi S, Lotfi S (2015) Modified cuckoo optimization algorithm (MCOA) to solve graph coloring problem. Appl Soft Comput 33:48

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

Memeti S, Pllana S, Binotto A, Kołodziej J, Brandic I (2018) Proceedings of the international conference on learning and optimization algorithms: theory and applications - LOPAL 18. ACM Press. doi 10(1145/3230905):3230906

Nguyen TT, Vo DN (2015) Modified cuckoo search algorithm for short-term hydrothermal scheduling. Int J Electr Power Energy Syst 65:271

Olamaei J, Moradi M, Kaboodi T (2013) 18th Electric power distribution conference, pp 1–6

Qin A, Suganthan P (2005) Self-adaptive differential evolution algorithm for numerical optimization. In: 2005 IEEE congress on evolutionary computation (IEEE, 2005), pp 1785–1791. https://doi.org/10.1109/cec.2005.1554904

ReVelle C, Toregas C, Falkson L (2010) Applications of the location set covering problem. Geogr Anal 8(1):65

Roeper T, Williams E (1987) Parameter setting. In: Hyams N (ed) The theory of parameters and syntactic development. Springer, Netherlands, pp 191–215

Salto C, Alba E (2011) Designing heterogeneous distributed GAs by efficiently self-adapting the migration period. Appl Intell 36(4):800. https://doi.org/10.1007/s10489-011-0297-9

Salveson ME (1995) The assembly line balancing problem. J Ind Eng 6(3):18

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

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

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

Soto R, Crawford B, Muñoz A, Johnson F, Paredes F (2015c) Advances in intelligent systems and computing. Artificial Intelligence Perspectives and Applications, vol 347. Springer, Berlin, pp 89–97

Soto R, Crawford B, Olivares R, Barraza J, Figueroa I, Johnson F, Paredes F, Olguín E (2017) Solving the non-unicost set covering problem by using cuckoo search and black hole optimization. Nat Comput 16(2):213

Spall J (1992) Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. IEEE Trans Autom Control 37(3):332

Stutzle T, Lopez-Ibanez M, Pellegrini P, Maur M, Montes de Oca M, Birattari M, Dorigo M (2012) What is autonomous search?. Parameter adaptation in ant colony optimization. Springer, Berlin, pp 191–215

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

Vasko FJ, Wilson GR (1984) Using a facility location algorithm to solve large set covering problems. Oper Res Lett 3(2):85

Vasko FJ, Wolf FE, Stott KL (1987) Optimal selection of ingot sizes via set covering. Oper Res 35(3):346

Xin C, Zhou Y, Zhonghua T, Qifang L (2017) A hybrid algorithm combining glowworm swarm optimization and complete 2-opt algorithm for spherical travelling salesman problems. Appl Soft Comput 58:104. https://doi.org/10.1016/j.asoc.2017.04.057

Yang XS (2010) Nature Inspired Cooperative Strategies for optimization (NICSO), vol 284. Studies in computational intelligence. Springer, Berlin, pp 65–74

Yang XS, He X (2013) Firefly algorithm: recent advances and applications. Int J Swarm Intell 1(1):36. https://doi.org/10.1504/ijsi.2013.055801

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

Yi W, Gao L, Li X, Zhou Y (2014) A new differential evolution algorithm with a hybrid mutation operator and self-adapting control parameters for global optimization problems. Appl Intell 42(4):642. https://doi.org/10.1007/s10489-014-0620-3

Zhang S, Zhou Y, Li Z, Pan W (2016) Grey wolf optimizer for unmanned combat aerial vehicle path planning. Adv Eng Softw 99:121. https://doi.org/10.1016/j.advengsoft.2016.05.015

Zhao R, Tang W (2008) Monkey algorithm for global numerical optimization. J Uncertain Syst 2(3):165

Zhou Y (2016) Hybrid symbiotic organisms search algorithm for solving 0–1 knapsack problem. Int J Bio Inspir Comput 1(1):1. https://doi.org/10.1504/ijbic.2016.10004304

Zhou Y, Chen H, Zhou G (2014) Invasive weed optimization algorithm for optimization no-idle flow shop scheduling problem. Neurocomputing 137:285. https://doi.org/10.1016/j.neucom.2013.05.063

Zhou Y, Luo Q, Chen H, He A, Wu J (2015a) A discrete invasive weed optimization algorithm for solving traveling salesman problem. Neurocomputing 151:1227. https://doi.org/10.1016/j.neucom.2014.01.078

Zhou Y, Li L, Ma M (2015b) A complex-valued encoding bat algorithm for solving 0–1 knapsack problem. Neural Process Lett 44(2):407. https://doi.org/10.1007/s11063-015-9465-y

Zhou Y, Bao Z, Luo Q, Zhang S (2016a) A complex-valued encoding wind driven optimization for the 0–1 knapsack problem. Appl Intell 46(3):684. https://doi.org/10.1007/s10489-016-0855-2

Zhou Y, Chen X, Zhou G (2016b) An improved monkey algorithm for a 0–1 knapsack problem. Appl Soft Comput 38:817