Solving the Set Covering Problem Using the Binary Cat Swarm Optimization Metaheuristic

Broderick Crawford, Ricardo Soto, Natalia Berrios, Eduardo Olguín

Abstract—In this paper, we present a binary cat swarm optimization for solving the Set covering problem. The set covering problem is a well-known NP-hard problem with many practical applications, including those involving scheduling, production planning and location problems. Binary cat swarm optimization is a recent swarm metaheuristic technique based on the behavior of discrete cats. Domestic cats show the ability to hunt and are curious about moving objects. The cats have two modes of behavior: seeking mode and tracing mode. We illustrate this approach with 65 instances of the problem from the OR-Library. Moreover, we solve this problem with 40 new binarization techniques and we select the technical with the best results obtained. Finally, we make a comparison between results obtained in previous studies and the new binarization technique, that is, with roulette wheel as transfer function and $V_3$ as discretization technique.

Keywords—Binary cat swarm optimization, set covering problem, metaheuristic, binarization methods.

I. INTRODUCTION

THE Set Covering Problem (SCP) [15], [14], [26] is a classic problem that consists in finding a set of solutions which allow to cover a set of needs at the lowest cost possible. There are many applications of these kind of problems, the main ones are: location of services, files selection in a data bank, simplification of boolean expressions, balancing production lines, among others.

In the field of optimization, many algorithms have been developed to solve the SCP. Examples of these optimization algorithms include: Genetic Algorithm (GA) [25], [1], Ant Colony Optimization (ACO) [3], [30], Particle Swarm Optimization (PSO) [14], [16], Firefly Algorithm [17], [18], Shuffled Frog Leaping [19], and Cultural Algorithms [15] have been also successfully applied to solve the SCP. Our proposal of algorithm uses cat behavior to solve optimization problems, it is called Binary Cat Swarm Optimization (BCSO) [32].

BCSO refers to a serie of heuristic optimization methods and algorithms based on cat behavior in nature. Cats behave in two ways: seeking mode and tracing mode. BCSO is based in CSO [29] algorithm, proposed by Chu and Tsai in 2006 [12]. The difference is that in BCSO the vector position consists of ones and zeros, instead the real numbers of CSO.

This paper is an improvement of previous work [13], this seeks to get better results for each instance of OR-Library. We use a new method of setting parameters, which we choose different parameters for each instances set. Moreover, we tested 40 new techniques binarization [20], then we analyze the results to choose the method with which the best results. To use the binarization technique, we change the technique usually proposed for tracing mode, and we discover if this could help to improve results.

This paper is structured as follows: In Section II, a brief description of what SCP is given. Section III gives: what BCSO is, the explanation and algorithm of behaviors. In Section IV, an explanation of how was BCSO used for solving the SCP is presented. Section V gives an analysis and results table. Finally, conclusions are given in Section VI.

II. SET COVERING PROBLEM

The SCP [9], [6], [28] can be formally defined as follows. Let $A = (a_{ij})$ be an $m$-row, $n$-column, zero-one matrix. We say that a column $j$ can cover a row if $a_{ij} = 1$. Each column $j$ is associated with a nonnegative real cost $c_j$. Let $I = \{1,...,m\}$ and $J = \{1,...,n\}$ be the row set and column set, respectively. The SCP calls for a minimum cost subset $S \subseteq J$, such that each row $i \in I$ is covered by at least one column $j \in S$. A mathematical model for the SCP is

$$v(\text{SCP}) = \min \sum_{j \in J} c_j x_j \quad (1)$$

subject to

$$\sum_{j \in J} a_{ij} x_j \geq 1, \quad \forall i \in I, \quad (2)$$

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

The objective is to minimize the sum of the costs of the selected columns, where $x_j = 1$ if column $j$ is in the solution, 0 otherwise. The constraints ensure that each row $i$ is covered by at least one column.

The SCP has been applied to many real world problems such as crew scheduling [2], location of emergency facilities [35], production planning in industry [34], vehicle routing [4], ship scheduling [22], network attack or defense [7], assembly line balancing [23], traffic assignment in satellite communication systems [31], simplifying boolean expressions [8], the calculation of bounds in integer programs [10], information retrieval [21], political districting [24], stock cutting, crew scheduling problems in airlines [27] and other important real life situations. Because it has wide applicability, we deposit our interest in solving the SCP.
III. Binary Cat Swarm Optimization

Among the known felines, there are about thirty different species, e.g., lion, tiger, leopard, cat, among others. Though many have different living environments, cats share similar behavior patterns.

For wild cats, the hunting skill ensures their food supply and survival of their species. Feral cats are groups with a mission to hunt their food. They are very wild feline colonies, ranging from 2-15 individuals.

Binary Cat Swarm Optimization [32] is an optimization algorithm that imitates the natural behavior of cats [11], [33]. Cats have curiosity by objects in motion and have a great hunting ability. It might be thought that cats spend most of the time resting, but in fact they are constantly alert and moving slowly. This behavior corresponds to the seeking mode. Furthermore, when cats detect a prey, they spend lots of energy because of their fast movements. This behavior corresponds to the tracing mode. In BCSO these two behaviors are modeled mathematically to solve complex optimization problems. Based on all these behaviors we formulate BCSO.

In BCSO, the first decision is the number of cats needed for each iteration. Each cat, represented by \(c_{k}\), where \(k \in [1, C]\), has its own position consisting of \(M\) dimensions, which are composed by ones and zeros. Besides, they have speed for each dimension, \(d\), a flag for indicating if the cat is on seeking mode or tracing mode and finally a fitness value that is calculated based on the SCP. The BCSO keeps to search the best solution until the end of iterations.

In BCSO the bits of the cat positions are \(x_{j} = 1\) if column \(j\) is in the solution, 0 otherwise (1). Cat position represents the solution of the SCP and the constraint matrix ensure that each row \(i\) is covered by at least one column. Next is described the BCSO general pseudocode where MR is a percentage that determine the number of cats that undertake the seeking mode.

Algorithm 1 BCSO()

1: Create \(C\) cats;
2: Initialize the cat positions randomly with values between 1 and 0;
3: Initialize velocities and flag of every cat;
4: Set the cats into seeking mode according to MR, and the others into tracing mode;
5: Evaluate the cats according to the fitness function;
6: Keep the best cat which has the best fitness value into bestcat variable;
7: Move the cats according to their flags, if \(c_{k}\) is in seeking mode, apply the cat to the seeking mode process, otherwise apply it to the tracing mode process. The process steps are presented above;
8: Re-pick number of cats and set them into tracing mode according to MR, then set the other cats into seeking mode;
9: Check the termination condition, if satisfied, terminate the program, and otherwise repeat since step 5;

A. Seeking Mode

This sub-model is used to model the situation of the cat, which is resting, looking around and seeking the next position to move to. Seeking mode has essential factors: Probability of Mutation Operation (PMO); Counts of Dimensions to Change (CDC), it indicates how many of the dimensions varied; Seeking Memory Pool (SMP), it is used to define the size of seeking memory for each cat. SMP indicates the points explored by the cat, this parameter can be different for different cats.

The following pseudocode describe cat behavior seeking mode. In which \(FS_i\) is the fitness of \(i^{th}\) cat and \(FS_{b} = FS_{max}\) for finding the minimum solution and \(FS_{b} = FS_{min}\) for finding the maximum solution. To solve the SCP we use \(FS_{b} = FS_{max}\).

\[
P_{i} = \frac{FS_{i} - FS_{b}}{FS_{max} - FS_{min}}
\]

(4)

B. Tracing Mode

Tracing mode is the sub-model for modeling the case of the cat in tracing targets. In the tracing mode, cats are moving towards the best target. Once a cat goes into tracing mode, it moves according to its own velocities for each dimension. Every cat has two velocity vector are defined as \(V_{0}^{i}\) and \(V_{1}^{i}\). \(V_{0}^{i}\) is the probability that the bits of the cat change to zero and \(V_{1}^{i}\) is the probability that bits of cat change to one. The velocity vector changes its meaning to the probability of mutation in each dimension of a cat. The tracing mode action is described in the next pseudocode:

Step1: Calculate \(d_{kd}^{0}\) and \(d_{kd}^{1}\) where \(X_{best,d}\) is the \(d^{th}\) dimension of the best cat, \(r_{1}\) has a random values in the interval of \([0,1]\) and \(c_{1}\) is a constant which is defined by the user

\[
\begin{align*}
&\text{if } X_{best,d} = 1 \text{ then } d_{kd}^{1} = r_{1}c_{1} \text{ and } d_{kd}^{0} = -r_{1}c_{1} \\
&\text{if } X_{best,d} = 0 \text{ then } d_{kd}^{1} = -r_{1}c_{1} \text{ and } d_{kd}^{0} = r_{1}c_{1}
\end{align*}
\]

(5)

Step2: Update process of \(V_{0}^{i}\) and \(V_{1}^{i}\) are as follows, where \(w\) is the inertia weight and \(M\) is the column numbers

\[
\begin{align*}
V_{0}^{i} &= wV_{0}^{i} + d_{kd}^{0} \\
V_{1}^{i} &= wV_{1}^{i} + d_{kd}^{1}
\end{align*}
\]

\(d = 1,...,M\)

(6)

Step3: Calculate the velocity of \(c_{k}\), \(V_{kd}^{t}\), according to

\[
V_{kd}^{t} = \begin{cases} V_{1}^{i} & \text{if } X_{kd} = 0 \\
V_{0}^{i} & \text{if } X_{kd} = 1
\end{cases}
\]

(7)

Step4: Calculate the probability of mutation in each dimension, this is defined by parameter \(t_{kd}\), \(t_{kd}\) takes a value in the interval of \([0,1]\)

\[
t_{kd} = \frac{1}{1 + e^{-V_{kd}^{t}}}
\]

(8)

Step5: Based on the value of \(t_{kd}\) the new value of each dimension of cat is update as follows where \(rand\) is an aleatory variable \([0,1]\)
\[ X_{kd} = \begin{cases} X_{best,d} \text{ if } rand < t_{kd} \\ X_{k} \text{ if } t_{kd} < rand \end{cases} \quad d = 1, \ldots, M \quad (9) \]

The maximum velocity vector of \( V'_{kd} \) should be bounded to a value \( V_{max} \). If \( V'_{kd} \) value becomes higher than \( V_{max} \), a new velocity should be assigned to \( V'_{kd} \) or the \( V_{max} \) value.

**C. New Binarization Technique**

In this work, experiments were performed with 40 different binarization techniques [20] using 65 OR-Library instances. A selection of smaller costs obtained is performed for each instance for get the results of all the experiments. Finally, to determine the binarization technique with better results, the technique smallest results obtained by all the instances are selected. Thus we obtained that roulette wheel in the case of the discretization method and \( V_3 \) for transfer function.

The transfer functions define a probability to change an element of solution from 1 to 0, or vice versa. In this case we select the transfer function \( v_3 \) (10).

\[
T(V'_d) = \left| \frac{V'_d}{\sqrt{1 + (V'_d)^2}} \right| \quad (10)
\]

In addition to the Transfer functions, the discretization method was selected, Roulette wheel (11).

\[
Roulette: \quad p_i = \frac{f_i}{\sum_{j=1}^{k} f_j} \quad (11)
\]

**IV. SOLVING THE SET COVERING PROBLEM**

Next is described the Solving SCP pseudocode:

Algorithm 3 Solving SCP()
1: Initialize parameters in cats; 
2: Initialization of cat positions, randomly initialize cat positions with values between 0 and 1; 
3: Initialization of all parameter of BCSO; 
4: Evaluation of the fitness of the population. In this case the fitness function is equal to the objective function of the SCP; 
5: Change of the position of the cat. A cat produces a modification in the position based in one of the behaviors. i.e. seeking mode or tracing mode; 
6: If solution is not feasible then repaired. Each row in must be covered by at least one columns, to choose the missing columns do: the cost of a column/number of not covered row that can cover column \( j \); 
7: Eliminate the redundant columns. A redundant column is one that if removed, the solution remains feasible; 
8: Memorize the best found solution. Increase the number of iterations; 
9: Stop the process and show the result if the completion criteria are met. Completion criteria used in this work are the number specified maximum of iterations. Otherwise, go to step 3;

**A. Parameter Setting**

All the algorithms were configured before performing the experiments. To this end and starting from default values, a parameter of the algorithm is selected to be turned. Then, 10 independent runs are performed for each configuration of the parameter. Next, the configuration which provides the best performance on average is selected. Next, another parameter is selected so long as all of them are fixed. Table I shows the range of values considered and the configurations selected. These values were obtained experimentally.

This procedure was performed for each set of instances, Table I. In all experiments the BCSO was executed with 1000 iterations. Moreover, the results of the eight different transfer functions and five discretization techniques were considered to select the final parameter.

**TABLE I**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Instance Set</th>
<th>Selected</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Cats</td>
<td>C</td>
<td>A and B</td>
<td>[5,10,...,100]</td>
</tr>
<tr>
<td>Mixture Ratio</td>
<td>MR</td>
<td>A and B</td>
<td>[0.10,0.97,...,1.00]</td>
</tr>
<tr>
<td>Seeking Memory Pool</td>
<td>SMP</td>
<td>A and B</td>
<td>[5,10,...,100]</td>
</tr>
<tr>
<td>Probability of Mutation Operation</td>
<td>PMO</td>
<td>A and B</td>
<td>[0.10,0.97,...,1.00]</td>
</tr>
<tr>
<td>Counts of Dimension to Change</td>
<td>CDC</td>
<td>A and B</td>
<td>[0.001,0.01,...,0.9]</td>
</tr>
<tr>
<td>Inertia Weight</td>
<td>w</td>
<td>A and B</td>
<td>[0.1,0.25,...,5]</td>
</tr>
<tr>
<td>Factor c1</td>
<td>c1</td>
<td>A and B</td>
<td>[0.1,0.25,...,5]</td>
</tr>
</tbody>
</table>

**V. RESULTS**

The BCSO performance was evaluated experimentally using 65 SCP test instances from the OR-Library of Beasley [5]. The Table II and III shows the results of the 65 instances. The \( Z_{opt} \) column reports the optimal value or the best known solution for each instance. The \( Z_{best} \) and \( Z_{avg} \) columns report the lowest cost and the average of the best solutions obtained in 30 runs respectively. The quality of a solution is evaluated in terms of the percentage deviation relative (RPD) of the solution reached \( Z_b \) and \( Z_{opt} \) (which can be either the optimal or the best known objective value). RPD was evaluated using \( Z_b = Z_{best} \).
\[ RPD = \left( \frac{Z_b - Z_{opt}}{Z_{opt}} \right) \times 100 \] (12)

About the solutions obtained we reach 6 optimum, all in Table II. The others results are very close to optimum values. If the best binarization technique was chosen for each set of instances is very probable that better results are obtained.

Comparing the \( RPD_{Best} \) average of each instances set with the obtained results in previous work [13], where transfer function and discretization technique are not used, it can be seen that in most of cases the results were improved. In Table IV the best difference is in instance set with 8,91 RPD, where Average of the New RPD is 0,82. Other instances set have a difference between about 3,0 and 7,0 RPD. The bad result is in the instance set NRF, with -0,12 of difference, worst results were obtained. The most important thing is that in most cases the results were improved. This shows that using the transfer function, discretization technique and use the new setting parameters achieves better results.

<table>
<thead>
<tr>
<th>Instance</th>
<th>( Z_{opt} )</th>
<th>( Z_{Best} )</th>
<th>( Z_{Avg} )</th>
<th>( RPD_{Best} )</th>
<th>( RPD_{Avg} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCP 51</td>
<td>429</td>
<td>432</td>
<td>414,6</td>
<td>0,7</td>
<td>2,9</td>
</tr>
<tr>
<td>SCP 52</td>
<td>512</td>
<td>516</td>
<td>532,2</td>
<td>0,8</td>
<td>3,9</td>
</tr>
<tr>
<td>SCP 53</td>
<td>516</td>
<td>520</td>
<td>554,9</td>
<td>0,8</td>
<td>7,5</td>
</tr>
<tr>
<td>SCP 54</td>
<td>492</td>
<td>497</td>
<td>514,5</td>
<td>0,6</td>
<td>4,1</td>
</tr>
<tr>
<td>SCP 55</td>
<td>512</td>
<td>515</td>
<td>527</td>
<td>0,6</td>
<td>2,9</td>
</tr>
<tr>
<td>SCP 56</td>
<td>560</td>
<td>560</td>
<td>568,7</td>
<td>0,0</td>
<td>1,6</td>
</tr>
<tr>
<td>SCP 47</td>
<td>430</td>
<td>434</td>
<td>437,5</td>
<td>0,9</td>
<td>1,7</td>
</tr>
<tr>
<td>SCP 48</td>
<td>492</td>
<td>494</td>
<td>518</td>
<td>0,4</td>
<td>5,3</td>
</tr>
<tr>
<td>SCP 49</td>
<td>641</td>
<td>658</td>
<td>678,4</td>
<td>2,7</td>
<td>5,7</td>
</tr>
<tr>
<td>SCP 410</td>
<td>514</td>
<td>518</td>
<td>526,6</td>
<td>0,8</td>
<td>2,4</td>
</tr>
</tbody>
</table>

VI. CONCLUSIONS

In this paper we use a binary version of cat swarm optimization, to solve SCP using its column based representation (binary solutions). In binary discrete optimization problems the position vector is binary. This causes significant change in BCSO with respect to CSO with real numbers. In fact in BCSO the seeking mode the slight change in the position takes place by introducing the mutation operation. The interpretation of velocity vector in tracing mode also changes to probability of change in each dimension of position of the cats. The proposed BCSO is implemented and tested using 65 SCP test instances from the OR-Library of Beasley.

As can be seen from the results, metaheuristic performs well in most cases observed according to old RPD works [13]. This paper has shown that the BCSO is a valid alternative to solve the SCP. The algorithm performs well regardless of the scale of the problem. Moreover, it could also better solutions using different parameter setting for each set of instances.

We can see the premature convergence problem, a typical problem in metaheuristics, which occurs when the cats quickly attain to dominate the population, constraining it to converge to a local optimum. For future works the objective will be make them highly immune to be trapped in local optima and thus less vulnerable to premature convergence problem. Thus,
we could propose an algorithm that shows improved results in terms of both computational time and quality of solution.

ACKNOWLEDGMENTS

The author Broderick Crawford is supported by grant CONICYT/FONDECYT/REGULAR/1140897 and Ricardo Soto is supported by grant CONICYT/FONDECYT/INICIACION/11130459

REFERENCES