PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

AN IMPROVED MONKEY ALGORITHM FOR THE SET COVERING PROBLEM

Gabriel Francisco Luis Embry Calderon
Diego Esteban Flores Castillo

Profesor Guía: **Dr. Broderick Crawford Labrín**
Profesor Co-referente: **Dr. Ricardo Soto de Giorgis**

INFORME FINAL PROYECTO DE TITULO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO CIVIL EN INFORMÁTICA

October 2017

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA ESCUELA DE INGENIERÍA
INFORMÁTICA

AN IMPROVED MONKEY ALGORITHM FOR THE SET COVERING PROBLEM

Gabriel Francisco Luis Embry Calderon
Diego Esteban Flores Castillo

Profesor Guía: **Dr. Broderick Crawford Labrín**
Profesor Co-referente: **Dr. Ricardo Soto de Giorgis**

October 2017

Acknowledgements

Gabriel Embry: This project is dedicated to my family and friends, who supported me, economically and emotionally, during this first big step necessary to fulfill my dreams and goals. It is also dedicated to all my teachers, especially to our guide teacher and co-referent, who gave me the necessary knowledge to finish this process.

Diego Flores: This research is dedicated to my family for providing me with unconditional support in all my years of studies, as well as the friendships I have formed inside and outside the university, who have made this process even more pleasing.

Abstract

Abstract. The Set Covering Problem (SCP) is a well-known NP-hard problem of combinatorial analytic. This problem consists in to find solutions covering the needs at lower cost. Those needs can be services to cities, load balancing in production lines or data-banks selections. In this work, we study the resolution of the SCP through the Improved Binary Monkey Algorithm (IBMA), and a new variation of its. This algorithm is based in swarm intelligence inspired from the mountain-climbing behavior of monkeys. We also will be realize comparatives test between the IBMA, the IBMAV and the Binary Cat Swarm Optimization (BCSO).

Keywords: Combinatorial Optimization, Binary Monkey algorithm, Metaheuristic, Set Covering Problem.

Contents

1 Introduction

Over the years, many companies have seen the need to use their resources to meet the needs of a sector, so that, these companies try to always use their capital efficiently as possible, in other words, trying to minimize the costs. Such situations can be represented by the Set Covering Problem (SCP), which is a classic problem in combinatorics, computer science and computational complexity theory. Because the characteristics of this problem, the complexity of this begins to increase with the increasing number of restrictions (needs), so the number of solutions increase and the time needed to check those solutions also does [5], which is why it is not feasible to fix it in manually and have had to investigate new ways to solve it, which led to the use of metaheuristics in order to solve the problem, who are algorithms generally based on neural networks, animal behavior or genetic algorithms.

In this research, it was decided that the Improved Binary Monkey Algorithm [20] (IBMA) is going to be used for solving the SCP. This metaheuristic consists of four steps: Climb Process, Watch-Jump Process, somersault Process and Cooperation Process. These steps aim to find the local optimal solution, find better solutions than the previous step, find a new solution domain and ultimately improve the local solution finding and accelerate the convergence order.

This research aims to understand the SCP and propose a solution based on a metaheuristic. First the Set Covering Problem with its mathematical model will be explained in depth and also an graphic example will be explained. Subsequently it will be discussed the related work, i.e. what kind of research has been done on the SCP, and the metaheuristic chosen. Finally it will explain the IBMA, in which each of its steps will be explained, along with mathematical models that support them, and its pseudo code in order to implement the algorithm.

2 Objectives

2.1 General Objectives

Solve the Set Covering problem using an improved monkey algorithm

2.2 Specific Objectives

- Fully understand the Set Covering Problem
- Fully understand the Improved Monkey Algorithm
- Implement the algorithm and achieve the solution of the SCP
- Perform an experimental evaluation
- Compare the performance between the MA and the IBMA

3 Set covering problem

The allocation Set Covering Problem (SCP) consists in a set of values that have a relationship together, and thanks an objective function, it is possible to maximize or minimize the allocation cost of those values. It is a classic combinatorial problem that belongs to the category NP-Hard [11]. The SCP in specific seeks to find the lowest possible allocation cost. That is, given a set of numbers, called universe (U), and a collection (S) of n sets whose union equals the universe, the Set Covering seeks to identify the smallest sub-collection of S whose union equals the universe, in other words, it seeks to cover all needs (rows) with the lowest cost (columns). That said, we can mention that the best way to represent this problem, is in the form of an assignment matrix (M x N). Where M represent the needs that are to covering and N columns variables to assign. The assignment matrix is based on a series of restrictions that must be cover to be considered a viable solution [8]. Along with the already mentioned in the above paragraph, it should be take into consideration that the Set Covering Problem is a problem of binary domain, i.e. ones (1) and zeros (0).

The Set Covering Problem has many application in real life and industry, for example, the allocation of services by municipalities or cities, load balancing production lines [15], selection of files in a database [6], among many others. As shown in the application examples mentioned above, the problem can be applied in different circumstances of decision making. As said, the SCP can be used in different contexts of decision making, moreover, the more information these decision have, it will help to improve the quantitative and qualitative performance of the assets of the entity in charge (i.e. a company) that could be used in a better way, thus improving the performance and quality of service. In order to better understanding of this problem is necessary to explain the mathematical formulation. This will be explained using formulas and mathematical notation that help to expose in a didactic way the complexity and the characteristics of the problem, achieving a greater understanding of the problem. Following, will be exposed the domain; objective function and restrictions of the problem:

Then, will be exposed the domain; objective function and restrictions of the problem:

$$\min Z = \sum_{j=1}^{n} c_j x_j$$

Subject to:

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1 \quad \forall i \in \{1, 2, 3, ..., n\},$$

$$x_j \in \{0, 1\}$$

In order to better understand the problem, let's suppose we have a mine divided in the following way:

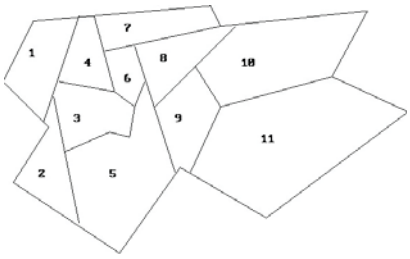

Figure 1: Example

Currently the cost of transporting the mineral is too high, so we must install storage facilities at certain points, which should satisfy all the needs of the mine. Such plants may store the mineral obtained from its quadrant and surrounding area. In other words, it seeks to minimize the cost of transportation of minerals.

Assuming the cost of transportation as one (1) in all quadrants, restrictions for this problem can be expressed mathematically as:

$$Minimize X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 + X_8 + X_9 + X_{10} + X_{11}$$

subject to

$$X_1 + X_2 + X_3 + X_4 \geq 1$$

$$X_1 + X_2 + X_3 + X_5 \geq 1$$

$$X_1 + X_2 + X_3 + X_4 + X_5 + X_6 \geq 1$$

$$X_1 + X_3 + X_4 + X_6 + X_7 \geq 1$$

$$X_2 + X_3 + X_5 + X_6 + X_8 + X_9 \geq 1$$

$$X_3 + X_4 + X_5 + X_6 + X_7 + X_8 \geq 1$$

$$X_4 + X_6 + X_7 + X_8 \geq 1$$

$$X_5 + X_6 + X_7 + X_8 + X_9 + X_{10} \geq 1$$

$$X_5 + X_8 + X_9 + X_{10} + X_{11} \geq 1$$

$$X_8 + X_9 + X_{10} + X_{11} \geq 1$$

$$X_9 + X_{10} + X_{11} \geq 1$$

Given these conditions, one of the best solutions for this typical Set Covering Problem would build a distribution plant in quadrants 3, 7 and 9, thus all the above-defined constraints are satisfied.

4 Previous work

The Set Covering Problem is a problem that has been under investigation for many years by the computer science and complexity theory. It is an interesting problem to investigate due many real world problems [8] can be modeled by the SCP, such as production planning in industry [18], facility location problem [17] and crew scheduling in airlines [12]. During years many algorithms have been developed to solve it. Exact algorithms are mostly based on brand-and-bound and branch-and-cut [1] [9], however, these algorithms are inefficient because are time-consuming and can only solve instances of a very limited size. For this reason many research efforts have been focused on the development of heuristics to find good or near-optimal solution within a reasonable period of time. Classical greedy algorithms are very simple, fast, and easy to code in practice, but they rarely produce high quality solutions for their myopic and deterministic nature [3], due that most of researchers, uses meta-heuristics in order to solve the SCP, such as genetic algorithms inspired by biological evolution and molecular genetic base, simulated annealing algorithm inspired by the metallurgy, animal based meta-heuristics inspired by animal behavior, such as Ant Colony Optimization [4] and the Monkey Algorithm [19], among many others.

The Monkey Algorithm(MA) [19] was proposed to solve global numerical optimization problem with continuous variables in 2007. It is a swarm intelligence based algorithm derived from simulation of the mountain-climbing processes of the monkeys when they look for food. The MA consist in four process, initialization, climb process, watch-jump process and somersault process. Later in 2016 this algorithm was improved by implementing the cooperation process in order to speed up the convergence rate of the algorithm and the greedy strategy repair process, used to correct the infeasible solutions and improve the quality of the feasibility, this new algorithm was called improved monkey algorithm(IMA) [21].

The MA is relatively new, it has been used to solve the Renewable Energy Integration Problem [13], who is solved as a Multi-Objective Optimization Problem where the objectives take in account are the minimization of the total life-cycle cost of the system, involving the capital, replacement, operation, maintenance and the minimization of greenhouse gas emissions. In the other hand, the IMA have only been used for the knapsack problem [21]. The knapsack problem can be explained as follows, we have a backpack and objects that carry it with different weights, the total of these items exceeds the maximum that can carry the backpack so you need to maximize the total value without exceeding the limit.

5 Pre-Processing

To accelerate the speed with which the algorithm finds solutions, we introduce to Pre-Processing phase before the meta-heuristic, which will reduce the size of the instances and improve the performance, in other words, it will generate a equivalent smaller problem. In this work, we will use two methods that have proven to be effective: Column Domination [2] and Column Inclusion [10]. Column Domination consists of deleting the redundant columns without affecting the final solution. In other words, if the rows belonging to the column j are covered by another column with a lower cost than C_j, then the column is "dominated" and it can be removed. The pseudo-code of Column Domination is given as follows:

```
1:    Order all columns by cost, ascending.
2:    if Two or more columns have the same cost then.
3:        Order those columns by the amount of rows Ij
        covered by column j, descending.
4:        Check if rows Ij can be covered by a set of
        other columns with a cost lower than Cj.
5:        if Cost is lower
6:            The column j is dominated and it can be
              removed.
7:        endif
8:    endif
```

After performing the above algorithm, we perform the Column Inclusion method. It consists in include in the final solutions the columns that cover a row that is only covered for that column. That is, if a row is covered by only one column after the above domination, that column must be included in the optimal solution, and there is no need to evaluate its feasibility.

6 Description of the Improved Monkey Algorithm

The Monkey Algorithm (MA) was proposed in 2007 to solve numerical optimization problems as a new swarm intelligence based algorithm inspired from the mountain-climbing behavior of monkey when they look for new food sources [21]. This algorithm, consist in the following steps: the Climb 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 (Climb 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 his current position to the new mountain (Watch-Jump Process), and then repeat the Climb 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, the Improved Binary Monkey Algorithm (IBMA) [19], it include the same three steps that it have its predecessor, but also includes two new steps: the Greedy Strategy and the Cooperation Process. The greedy algorithm is used to correct the infeasible solutions and to improve the quality of feasibility, in other words, is a reparation algorithm. The Somersault Process is modified to avoid falling into local search; the Cooperation Process is implemented to speed up the convergence rate. Also we implement a redundancy reduction process which helps to improve the quality of the solutions.

In this work, we will present a new variation of the IBMA. This new algorithm have the same steps as the former, but the Climb Process and the Cooperation process have been modified in order to improve the solutions.

6.1 Coding Method

The parameter M is defined as the population size of monkeys. For a certain monkey i, its position is denoted as a vector $X_i = (x_{i1}, x_{i2}, \ldots, x_{in})$, and this position will be employed to express a solution of the set covering problem, where $x_{ij} \in \{0, 1\}$ and $j \in 1, 2, \ldots, n$, represent the number of decision variables $x_{ij} = 1$ indicates the item $j$is included in the solution, and $x_{ij} = 0$ indicates it is not.

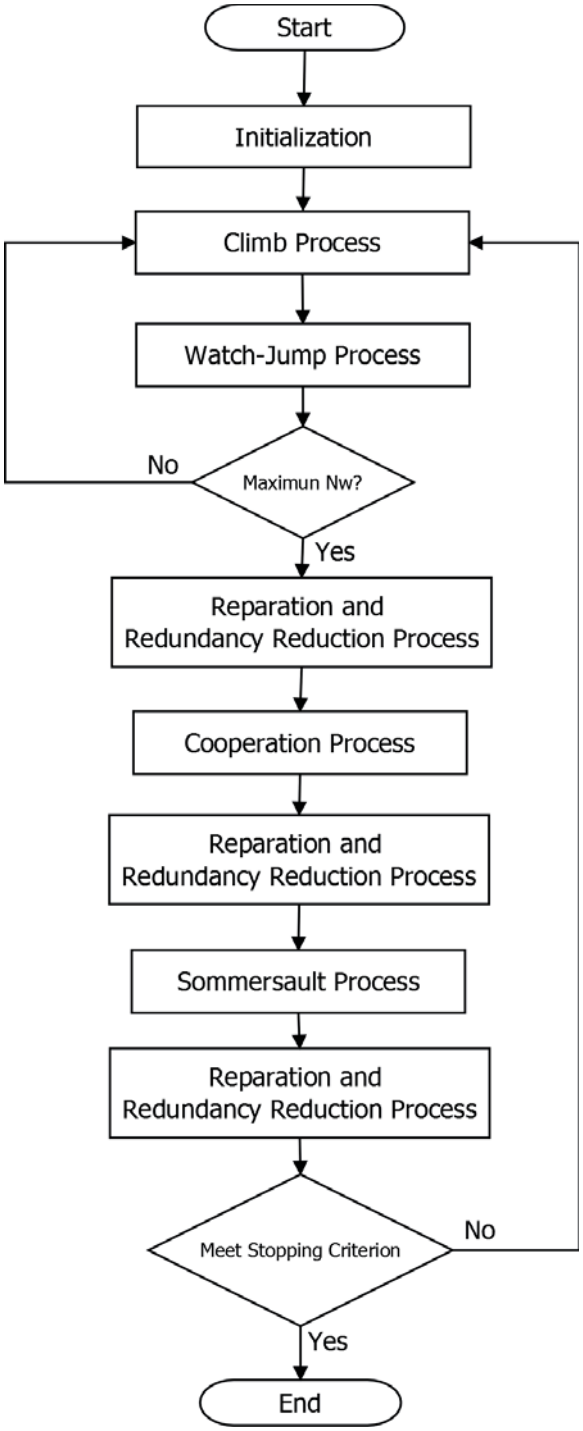

Figure 2: Improved Binary Monkey Algorithm

6.2 Initial Population

In Improved Binary Monkey Algorithm, and in our variation, the initial population is randomly generated. The random initialization process of M solutions (monkeys) and n decision variables (items) is defined as follows:

```
for i = 1 to M do
    for j = 1 to N do
        Randomly generate rand;
        If rand < 0.5 then
            Xij = 0 ;
        else
           Xij = 1 ;
        endif
    endfor
endfor
```

where X_{ij} represents the jth component in the vector X_i

6.3 Climb Process

According to the idea of pseudo-gradient based simultaneous perturbation stochastic approximation (SPSA) [16], this process is a step-by-step procedure to improve the objective function value by choosing the best between two positions that are generated around the current one. For the monkey i, X_{ij}, and $f(X_i)$ the objective function value. The Climb Process used in the IBMA is given as follows:

1. Randomly generate two vectors $\triangle x_i{}'$ and $\triangle x_i{}''$, where $\triangle x_i{}' = (\triangle x_{i1}{}', \triangle x_{i2}{}', \ldots, \triangle x_{in}{}')$ and $\triangle x_i{}'' = (\triangle x_{i1}{}'', \triangle x_{i2}{}'', \ldots, \triangle x_{in}{}'')$, where

$$\triangle x_{ij}{}', \triangle x_{ij}{}'' = \begin{cases} a & \text{with probability } 1/2 \\ -a & \text{with probability } 1/2 \end{cases}$$

   $j \in 1, 2, \ldots, n$, respectively. The parameter a, is called step of the Climb Process. This parameter can be determined depending the situation but always had to be greater than zero ($a > 0$). Here A is set to 1 for the SCP.

2. Set $x_{ij}^{'} = |x_{ij} - \triangle x_{ij}^{'}|$ and $x_{ij}^{''} = |x_{ij} - \triangle x_{ij}^{''}|$, $j \in 1, 2, \ldots, n$, respectively, where $|x|$ represents the absolute value of x. Set $X_i^{'} = (x_{i1}^{'}, x_{i2}^{'}, \ldots, x_{in}^{'}), X_i^{''} = (x_{i1}^{''}, x_{i2}^{''}, \ldots, x_{in}^{''})$. If $\triangle {x_i}^{'}$ or $\triangle {x_i}^{''}$ is unfeasible, keep Xi unchanged.

3. Calculate $f(X_i^{'})$ and $f(X_i^{''})$, $i \in 1, 2, \ldots, M$, respectively.

4. If $f(X_i^{'}) < f(X_i^{''})$ and $f(X_i^{'}) < f(X_i)$, set $X_i = X_i^{'}$. If $f(X_i^{''}) < f(X_i^{'})$ and $f(X_i^{''}) < f(X_i), X_i = X_i^{''}$.

5. Repeat steps (1) to (4) until the maximum allowable number of iterations has been reached. This limit is denoted by Nc.

In the IBMA variation proposed in this work, only the step (2) is modified, instead of keep X_i unchanged if the generated solution is inviable, we “fix" those solutions.

(2) Set $x_{ij}^{'} = |x_{ij} - \triangle x_{ij}^{'}|$ and $x_{ij}^{''} = |x_{ij} - \triangle x_{ij}^{''}|$, $j \in 1, 2, \ldots, n$, respectively, where $|x|$ represents the absolute value of x. Set $X_i^{'} = (x_{i1}^{'}, x_{i2}^{'}, \ldots, x_{in}^{'})$and $X_i^{''} = (x_{i1}^{''}, x_{i2}^{''}, \ldots, x_{in}^{''})$. If $\triangle {x_i}^{'}$ or $\triangle {x_i}^{''}$ is out of domain (unfeasible), generate a random number between 0 and 1, if random < 0.5 set X_{ij} to 0, otherwise 1.

6.4 Watch Jump Process

When the monkey reach the top of the mountains, each monkey will try to find a higher point than his current position within its sight. If find one, it will jump somewhere of the new mountain from his current position, and then it will be start the Climb Process again. For the monkey i, its position is $X_i = (x_{i1}, x_{i2}, \ldots, x_{in})$, $i \in 1, 2, \ldots,$ M. The Watch-Jump Process is given as follows:

1. Randomly generate a real y_j in the interval $[x_{ij} - b, x_{ij} + b]$, $j \in 1, 2, \ldots, n$ respectively, where b is the monkey's sight and it can be determined depending of the situation. Set $Y = (y_1, y_2, \ldots, y_n)$.

2. Because of the real $y_j \in [-1, 2]$, if $y_j < 0.5$, set y_j to 0; otherwise, set y_j to 1.

3. Calculate $f(Y_i), i \in 1, 2, \ldots, M$, respectively.

4. If $f(Y_i) < f(X_i)$, then $X_i = Y_i$.

5. Repeat steps (1) to (4) until the maximum allowable number of iterations has been reached. This limit is denoted by Nw.

6.5 Greedy Strategy: Reparation Process.

Solving the SCP, some monkeys may have an abnormal encode (that does not meet the constraints), for this reason, the local search strategy-greedy algorithm is implemented to repair the infeasible solutions and to improve the quality of feasibility. Assuming the monkey X_i, is not a feasible solution, there $X_i = (x_{i1}, x_{i2}, \ldots, x_{in}), i \in 1, 2, \ldots, M$, And A represent the constraint matrix. The greedy Algorithm is given as follows:

1. Search for an uncovered row according to the constraint matrix A

2. Create a vector with the possible candidates to cover the row previously

3. For each candidates calculate his weight (W): $W=$ (number of rows covered by the candidate (not yet covered at the moment))/Cost of the candidate)

4. Select the candidate with the highest W, and put it in the monkey X_i.

5. Repeat step (1) to (4) until the monkey X_i becomes feasible

6.6 Redundancy Reduction Process

Once the monkeys have gone through the Reparation Process, the Redundancy Reduction Process is initiated [14] to reduce the overall cost of the monkey. For each monkey i, $X_i = (x_{i1}, x_{i2}, \ldots, x_{in}), i \in 1, 2, \ldots, M$, the redundancy reduction process is given as follows:

1. Set the vector $Y = (Y_1, Y_2, \ldots, Y_n)$ equal to X_i $(Y = X_i)$

2. Starting from Y_k, $k \in N, N-1, \ldots, 0$, do $Yk = 0$

3. Then, check the feasibility of the vector Y, if Y is still feasible, it means that column is redundant, then set X_{ik} to 0 and decrease k in 1. Otherwise (Y is not feasible) do $Y_k = 1$, and decrease k in 1.

4. Repeat step (1) to (3), until there is no redundancy in the monkey X_i

6.7 Cooperation Process

After the Climb and Watch-Jump Process, each monkey will arrive at the highest mountain in his neighborhood, however, they will differ among all the monkeys. The purpose of the cooperation process is to improve the monkey's solution by cooperating with the monkey that has the higher mountain (best position), in other words, they will move along the direction of the best monkey. This process can speed up the convergence rate. Assume that the optimal position is $X^* = (x_1^*, x_2^*, \ldots, x_n^*)$. For the monkey i, $X_i = (x_{i1}, x_{i2}, \ldots, x_{in})$, $i \in 1, 2, \ldots, M$. The cooperation process is given as follows:

1. Randomly generate a real number α from the interval [0, 1].

2. If $\alpha < 0.5$, then $y_j = x_j$, otherwise, $y_j = x_j^*$, $j \in 1, 2, \ldots, n$, respectively.

3. Update the monkey's position X_i with Y

In the IBMA variation proposed in this work, a new step to the Cooperation Process is added. This new step check if $f(Y)$ have less cost than $f(X)$ before replacing the values. Otherwise, the process is repeated, if after L iterations (in this work, L is set to 3) the algorithm cannot find a better $f(Y)$, xi is maintained.

(3) Calculate $f(Y)$ and $f(X_i)$, if $f(Y) < f(Xi)$ updates monkey X_i with Y, otherwise do nothing.

(4) Repeat steps (1) to (3) until X_i is updated or the limit L is meet.

6.8 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, is limited within a certain region, which is limited by the pivot and the somersault interval. This process can effectively prevent monkeys falling into local search, however, after many iterations, the somersault process may lose efficacy, resulting in monkeys

falling into the local optimal domain, decreasing the population diversity. In the original MA, the monkeys will somersault along the direction pointing to the barycenter of all monkeys' current positions. Here, we randomly choose a monkey's position as the pivot replacing the one proposed by the original algorithm. For each monkey i, $X_i = (x_{i1}, x_{i2}, \ldots, x_{in}), i \in 1, 2, \ldots, M$, 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, is called the somersault interval)

2. Randomly generate an integer k, $k \in 1, 2, \ldots, M$, respectively. The monkey $k(X_k)$ will be the somersault pivot

3. Calculate $y_j = x_{kj} + \theta(x_{kj} - x_{ij})$

4. If $Y_j < 0.5$, set Y_j to 0, otherwise, set Y_j to 1. Update the monkeys positions X_i with Y

After repetitions of the somersault process, monkeys may reach the same domain to make the somersault process lose efficacy. In case of this problem, we set a parameter called "limit" to control monkeys running into the local optima solution. If the global optimal solution is not improved by a predetermined number of trials, the monkeys are abandoned and then reinitialized.

6.9 Termination Condition

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

7 Experimentation and Implementation details

The executions of both algorithms, the IBMAV and IBMAV, presented in this work where done for the purpose to compare thems. The input files used for this test were all SCP files available in OR-library [7] aside from the files SCPNRG and SCPNRH. Both algorithms were coded in Java, using eclipse neon IDE 1.8, and executed on 5 desktops, one with 3.1GHz Intel Pentium I5-4440 processor with 8 GB Ram under Windows 10, and four with 3.3GHz Intel Pentium I3-2100 processor with 16 GB ram under Windows 7. The results will be shown in the following tables, and the configuration used was the following: $Iterations = 10, Monkeys = 10, A = 1, b = 1, c = -1, d = 1, Nc = 2, Nw = 2$.

Table 1: Results table

Inst.	Type	Z_{opt}	Z_{best}	Z_{avg}	rpd	$\overline{rpd}(avg)$
SCP41	IBMAV.	429	432	441.87	0.70	3.00
SCP42	IBMAV.	512	518	543.67	1.17	6.19
SCP43	IBMAV.	516	520	537.87	0.78	4.24
SCP44	IBMAV.	494	496	523.45	0.40	5.96
SCP45	IBMAV.	512	518	536.29	1.17	4.74
SCP46	IBMAV.	560	564	582.16	0.71	3.96
SCP47	IBMAV.	430	432	442.16	0.47	2.83
SCP48	IBMAV.	492	495	512.12	0.61	4.09
SCP49	IBMAV.	641	662	683.41	3.28	6.62
SCP410	IBMAV.	514	520	541.16	1.17	5.28
SCP51	IBMAV.	253	258	269.22	1.98	6.41
SCP52	IBMAV.	302	316	325.06	4.64	7.64
SCP53	IBMAV.	226	229	236.19	1.33	4.51
SCP54	IBMAV.	242	242	248.80	0.00	2.81
SCP55	IBMAV.	211	212	219.06	0.47	3.82
SCP56	IBMAV.	213	213	226.09	0.00	6.15
SCP57	IBMAV.	293	295	309.45	0.68	5.61
SCP58	IBMAV.	288	289	300.45	0.35	4.32
SCP59	IBMAV.	279	281	289.87	0.72	3.90
SCP510	IBMAV.	265	267	273.38	0.75	3.16
SCP61	IBMAV.	138	141	146.80	2.17	6.38
SCP62	IBMAV.	146	149	157.12	2.05	7.62
SCP63	IBMAV.	145	148	154.74	2.07	6.72
SCP64	IBMAV.	131	133	137.77	1.53	5.17
SCP65	IBMAV.	161	165	173.48	2.48	7.75

Table 2: Results table

Inst.	Type	Z_{opt}	Z_{best}	Z_{avg}	rpd	$\overline{rpd}(avg)$
SCPA1	IBMAV.	253	256	261.83	1.19	3.49
SCPA2	IBMAV.	252	256	269.16	1.59	6.81
SCPA3	IBMAV.	232	238	246.41	2.59	6.22
SCPA4	IBMAV.	234	237	250.12	1.28	6.89
SCPA5	IBMAV.	236	238	246.22	0.85	4.33
SCPB1	IBMAV.	69	69	74.74	0.00	8.32
SCPB2	IBMAV.	76	76	82.09	0.00	8.02
SCPB3	IBMAV.	80	80	84.12	0.00	5.16
SCPB4	IBMAV.	79	81	84.70	2.53	7.23
SCPB5	IBMAV.	72	72	76.06	0.00	5.65
SCPC1	IBMAV.	227	231	243.16	1.76	7.12
SCPC2	IBMAV.	219	220	237.32	0.46	8.37
SCPC3	IBMAV.	243	251	260.83	3.29	7.34
SCPC4	IBMAV.	219	222	234.54	1.37	7.10
SCPC5	IBMAV.	215	219	228.38	1.86	6.23
SCPD1	IBMAV.	60	61	64.74	1.67	7.90
SCPD2	IBMAV.	66	67	70.16	1.52	6.30
SCPD3	IBMAV.	72	72	77.38	0.00	7.48
SCPD4	IBMAV.	62	63	66.35	1.61	7.02
SCPD5	IBMAV.	61	61	66.03	0.00	8.25
SCPNRE1	IBMAV.	29	29	30.64	0.00	5.67
SCPNRE2	IBMAV.	30	31	32.74	3.33	9.14
SCPNRE3	IBMAV.	27	28	29.74	3.70	10.16
SCPNRE4	IBMAV.	28	28	30.58	0.00	9.22
SCPNRE5	IBMAV.	28	28	29.83	0.00	6.57
SCPNRF1	IBMAV.	14	14	14.93	0.00	6.68
SCPNRF2	IBMAV.	15	15	15.64	0.00	4.30
SCPNRF3	IBMAV.	14	14	15.58	0.00	11.29
SCPNRF4	IBMAV.	14	14	15.06	0.00	7.60
SCPNRF5	IBMAV.	13	14	14.61	7.69	12.41

Table 3: Results table

Inst.	Type	Z_{opt}	Z_{best}	Z_{avg}	rpd	$\overline{rpd}(avg)$
SCP41	IBMA	429	431	440.00	0.47	2.56
SCP42	IBMA	512	525	552.45	2.54	7.90
SCP43	IBMA	516	520	538.03	0.78	4.27
SCP44	IBMA	494	495	521.74	0.20	5.62
SCP45	IBMA	512	516	534.58	0.78	4.41
SCP46	IBMA	560	566	580.70	1.07	3.70
SCP47	IBMA	430	434	442.48	0.93	2.90
SCP48	IBMA	492	497	511.54	1.02	3.97
SCP49	IBMA	641	655	687.22	2.18	7.21
SCP410	IBMA	514	521	539.83	1.36	5.03
SCP51	IBMA	253	259	270.93	2.37	7.09
SCP52	IBMA	302	314	322.64	3.97	6.84
SCP53	IBMA	226	229	235.38	1.33	4.15
SCP54	IBMA	242	242	249.67	0.00	3.17
SCP55	IBMA	211	212	219.45	0.47	4.01
SCP56	IBMA	213	214	225.32	0.47	5.79
SCP57	IBMA	293	298	310.87	1.71	6.10
SCP58	IBMA	288	289	301.41	0.35	4.66
SCP59	IBMA	279	280	293.51	0.36	5.20
SCP510	IBMA	265	269	275.51	1.51	3.97
SCP61	IBMA	138	141	147.83	2.17	7.13
SCP62	IBMA	146	150	156.00	2.74	6.85
SCP63	IBMA	145	148	155.06	2.07	6.94
SCP64	IBMA	131	132	137.29	0.76	4.80
SCP65	IBMA	161	163	171.16	1.24	6.31

Table 4: Results table

Inst.	Type	Z_{opt}	Z_{best}	Z_{avg}	rpd	$\overline{rpd}(avg)$
SCPA1	IBMA	253	255	263,25	0,79	4,05
SCPA2	IBMA	252	258	270,80	2,38	7,46
SCPA3	IBMA	232	238	246,32	2,59	6,17
SCPA4	IBMA	234	238	247,96	1,71	5,97
SCPA5	IBMA	236	236	246	0,00	4,24
SCPB1	IBMA	69	70	73,77	1,45	6,92
SCPB2	IBMA	76	76	82	0,00	7,89
SCPB3	IBMA	80	81	84,70	1,25	5,89
SCPB4	IBMA	79	80	84,67	1,27	7,19
SCPB5	IBMA	72	72	74,83	0,00	3,94
SCPC1	IBMA	227	235	242,77	3,52	6,95
SCPC2	IBMA	219	224	235,74	2,28	7,64
SCPC3	IBMA	243	247	259,45	1,65	6,77
SCPC4	IBMA	219	224	236,41	2,28	7,95
SCPC5	IBMA	215	217	227,93	0,93	6,02
SCPD1	IBMA	60	61	65,12	1,67	8,55
SCPD2	IBMA	66	66	69,74	0,00	5,67
SCPD3	IBMA	72	73	77,22	1,39	7,26
SCPD4	IBMA	62	62	65,32	0,00	5,36
SCPD5	IBMA	61	61	66,83	0,00	9,57
SCPNRE1	IBMA	29	29	30,45	0,00	5,01
SCPNRE2	IBMA	30	31	32,80	3,33	9,35
SCPNRE3	IBMA	27	28	29,58	3,70	9,56
SCPNRE4	IBMA	28	28	30,19	0,00	7,83
SCPNRE5	IBMA	28	28	29,96	0,00	7,03
SCPNRF1	IBMA	14	14	14,96	0,00	6,91
SCPNRF2	IBMA	15	15	15,58	0,00	3,87
SCPNRF3	IBMA	14	14	15,48	0,00	10,60
SCPNRF4	IBMA	14	14	15,06	0,00	7,60
SCPNRF5	IBMA	13	14	14,77	7,69	13,65

In the following tables, the IBMAV results will be compared with the Binary Swarm Cat Optimization results [14].

Table 5: Results table

SCP		IBMAV		BCSO	
Inst.	Z_{opt}	Z_{best}	$\overline{rpd}(avg)$	Z_{best}	$\overline{rpd}(avg)$
41	429	432	3.00	432	2.58
42	512	518	6.19	517	3.49
43	516	520	4.24	531	7.13
44	494	496	5.96	496	3.29
45	512	518	4.74	514	2.19
46	560	564	3.96	560	1.09
47	430	432	2.83	434	1.75
48	492	495	4.09	494	3.88
49	641	662	6.62	660	5.21
410	514	520	5.28	518	2.13
51	253	258	6.41	258	3.37
52	302	316	7.64	306	3.74
53	226	229	4.51	229	2.98
54	242	242	2.81	242	1.29
55	211	212	3.82	216	4.00
56	213	213	6.15	217	4.89
57	293	295	5.61	294	3.55
58	288	289	4.32	294	6.15
59	279	281	3.90	280	0.51
510	265	267	3.16	271	3.70
61	138	141	6.38	143	5.94
62	146	149	7.62	146	2.15
63	145	148	6.72	148	4.67
64	131	133	5.17	133	2.60
65	161	165	7.75	165	4.39

Table 6: Results table

SCP		IBMAV		BCSO	
Inst.	Z_{opt}	Z_{best}	$\overline{rpd}(avg)$	Z_{best}	$\overline{rpd}(avg)$
A1	253	256	3.49	271	8.56
A2	252	256	6.81	259	4.87
A3	232	238	6.22	238	4.54
A4	234	237	6.89	241	4.66
A5	236	238	4.33	237	1.05
B1	69	69	8.32	70	6.81
B2	76	76	8.02	80	10.26
B3	80	80	5.16	80	2.83
B4	79	81	7.23	81	5.86
B5	72	72	5.65	73	1.39
C1	227	231	7.12	232	3.22
C2	219	220	8.37	225	4.60
C3	243	251	7.34	251	8.67
C4	219	222	7.10	231	8.54
C5	215	219	6.23	222	6.33
D1	60	61	7.90	60	6.72
D2	66	67	6.30	69	5.61
D3	72	72	7.48	76	9.03
D4	62	63	7.02	63	5.43
D5	61	61	8.25	64	6.28
NRE1	29	29	5.67	30	3.45
NRE2	30	31	9.14	34	13.33
NRE3	27	28	10.16	29	18.02
NRE4	28	28	9.22	32	16.90
NRE5	28	28	6.57	30	7.14
NRF1	14	14	6.68	17	21.43
NRF2	15	15	4.30	16	18.00
NRF3	14	14	11.29	17	21.43
NRF4	14	14	7.60	15	20.48
NRF5	13	14	12.41	16	23.08

As we can see, both algorithms are fairly similar, but the IBMAV as shown to be better, because it got a lower RPD than the IBMA, and also, got a lower RPD average. When the IBMAV is compared with the BCSO [14], we can see that they are fairly similar, but the IBMAV is better if we only take in consideration the best value obtained, but when comparing the RPD(avg) the BCSO is better, in other words, his range of solutions were smaller than the IBMAV. Because of this facts, it can be safely stated that the changes made in the variation of the Improved Binary Monkey Algorithm where done for the better, because it gives good results overall.

8 Conclusion

The Set Covering Problem (SCP) is a well-known NP-hard problem of combinatorial analytic. This problem consists in to find solutions covering the needs at lower cost. Those needs can be services to cities, load balancing in production lines or data-banks selections. In this work, we study the resolution of the SCP through an algorithm based in swarm intelligence inspired from the mountain-climbing behavior of monkeys, the so called Improved Binary Monkey Algorithm (IBMA), wich was proposed, in 2016, as a variation of the Monkey Algorithm (MA). Also in this work a new IBMA varation is proposed by us (IBMAV) wich implements changes in some steps of the algorithm, in order to reduce the execution times and improve the solutions. Finally an experimental comparative test was perform in both algorithms, to compare theyr performance solving the Set Covering Problem, and at last, we compare those results, with the Binary Cat Swarm Optimization (BCSO) ones [14].

From the comparative experiments, we can conclude the following: The IBMAV, in the 72.72% of the tests, got better values (optimal or near-optimal values) than the IBMA algorithm, but in the other hand, the Improved Binary Monkey Algorithm got better average RPD than his variation (in the 56.37% of the tests), wich means, that the said algorithm gives a smaller range of solutions than the IBMAV. Given those facts we can say without a doub that the variation of the Improved Binary Monkey Algorithm, it will give better results when solving the SCP. Finally, when comparing the the performance between the IBMAV and the BCSO, we can see that the variation of the IBMAV it gives better results in the 80% of the test, but again, the IBMAV fails at give a small range of solutions (average RPD) when compared with the Binary Cat Swarm Optimization wich gives a lower average RPD in the 65.46% of the tests. Based on the results we strongly believe that the changes to the IBMA (IBMAV) where for the better (comparing the performance with the original IBMA) and it is on par with other metaheuristics of the same characteristics.

References

[1] E. Balas and M. Carrera. A dynamic subgradient-based branch-and-bound procedure for set covering. *Operations Research*, 44(6):875–890, 1996.

[2] J.E. Beasley. An algorithm for set covering problem. *European Journal of Operational Research*, 31(1):85 – 93, 1987.

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

[4] B. Crawford and C. Castro. Integrating lookahead and post processing procedures with ACO for solving set partitioning and covering problems. In *ICAISC*, volume 4029 of *Lecture Notes in Computer Science*, pages 1082–1090. Springer, 2006.

[5] B. Crawford, R. Soto, I. Fuenzalida, and E. Olguín. A binary invasive weed optimization algorithm for the set covering problem. In *Artificial Intelligence Perspectives in Intelligent Systems - Proceedings of the 5th Computer Science On-line Conference 2016 (CSOC2016), Vol 1*, pages 459–468, 2016.

[6] R. Day. Letter to the editorâon optimal extracting from a multiple file data storage system: An application of integer programming. *Operations Research*, 13(3):482–494, 1965.

[7] J. E. Or-library. urlhttp://people.brunel.ac.uk/ mastjjb/jeb/orlib/scpinfo.html.

[8] T. Feo and M. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Oper. Res. Lett.*, 8(2):67–71, April 1989.

[9] M. Fisher and P. Kedia. Optimal solution of set covering/partitioning problems using dual heuristics. *Management Science*, 36(6):674–688, 1990.

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

[11] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[12] E. Housos and T. Elmoth. Automatic optimization of subproblems in scheduling airlines crews. *Interfaces*, 1997.

[13] C. Ituarte-Villarreal, N. Lopez, and J. Espiritu. Using the monkey algorithm for hybrid power systems optimization. In *Complex Adaptive Systems*, volume 12 of *Procedia Computer Science*, pages 344–349. Elsevier, 2012.

[14] J. Lanza-Gutierrez, B. Crawford, R. Soto, N. Berrios, J. Gomez-Pulido, and F. Paredes. Analyzing the effects of binarization techniques when solving the set covering problem through swarm optimization. *Expert Systems with Applications*, 70:67 – 82, 2017.

[15] M. Salveson. The assembly line balancing problem. *Journal of Industrial Engineering*, 6:18–25, 1955.

[16] J. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, Mar 1992.

[17] F. Vasko and G. Wilson. Using a facility location algorithm to solve large set covering problems. *Operations Research Letters*, 3(2):85 – 90, 1984.

[18] F. Vasko, F. Wolf, and K. Stott Jr. Optimal selection of ingot sizes via set covering. *Operations Research*, 35(3):346–353, 1987.

[19] R. Zhao and W. Tang. Monkey algorithm for global numerical optimization. *J. Uncertain Syst*, 2007.

[20] Y. Zhou, X. Chen, and G. Zhou. An improved monkey algorithm for a 0-1 knapsack problem. *Appl. Soft Comput.*, 38:817–830, 2016.

[21] Y. Zhou, X. Chen, and G. Zhou. An improved monkey algorithm for a 0-1 knapsack problem. *Appl. Soft Comput.*, 38(C):817–830, January 2016.