

Solving Equations Systems Using the Ant Lion Optimizer and Monkey Algorithm

Liviu Octavian Maftciu-Scai¹, Mihail Crivoi¹, Roxana Teodora Maftciu-Scai¹

¹ West University of Timisoara, Computer Science Department, Romania

Abstract - In this paper, two solving equations systems methods based on Ant Lion Optimizer (ALO) and Monkey Algorithm (MA) algorithms are proposed. Results are obtained with proposed methods. Experimental results obtained without proposed methods were compared with results obtained by classical math methods, from the points of view of accuracy and runtime.

Keywords - linear equation systems, monkey algorithm, antlion optimizer, metaheuristics, convergence, runtime

1. Introduction

This section provides brief descriptions of the mathematical methods used to solve equation systems (ES), as well as the two metaheuristics used in our proposed approach.

1.1. Solving Equations Systems

Solving equations systems is an old concern in mathematics which comes from ancient China.

Over time, many methods have been proposed to solve ES. Currently, these are divided into two large classes, depending on the way in which they solve ES: direct and iterative methods. In case of direct methods, rounding errors and high complexity $O(n^3)$ are their main disadvantage, especially when solving medium and large-sized systems. The second class is based on an iterative process which starts with an initial approximation of the system solution. But these methods can work well only in cases of a well-conditioned system. The rounding and truncations are very small in most cases, so the iterative solution have a better accuracy [3] in general than direct methods [1].

In recent years, Artificial Intelligence (AI) techniques are increasingly used to solve systems of equations [2], [3]. Regarding the convergence of these methods, there are only a few published papers in which the convergence is mathematically proved [4], [5].

1.2. Ant Lion Optimizer Method

The Ant Lion Optimizer (ALO) proposed by Seyedali Mirjalili in paper [6], is a nature-inspired algorithm that mimics the hunting mechanism of antlions in nature. The algorithm was inspired by the life of antlions, a species belonging to the Myrmeleontidae family. The lifecycle of these insects has two phases: larvae and adult, the first being the most important in ALO. In fact, the ALO algorithm mimics interaction between antlions and ants in the trap. The behavior of antlions and their prey in nature was modelled mathematically, the ALO being based on that mathematical model [6].

DOI: 10.18421/SAR82-02

<https://doi.org/10.18421/SAR82-02>

Corresponding author: Liviu Octavian Maftciu-Scai,
DSPM Research Laboratory, Ibn Badis University,
Mostaganem, Algeria


Email: liviu.mafteiu@e-uvt.ro

Received: 02 May 2025.

Revised: 15 June 2025.

Accepted: 23 June 2025.

Published: 27 June 2025.

 © 2025 Liviu Octavian Maftciu-Scai, Mihail Crivoi & Roxana Teodora Maftciu-Scai; published by UIKTEN. This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 4.0 License.

The article is published with Open Access at <https://www.sarjournal.com/>

The ALO algorithm is described below in pseudocode [6]:

Algorithm 1

```

Initialize the first population of ants and antlions
randomly
Calculate the fitness of ants and antlions
Find the best antlions and assume it as the elite
(determined optimum)
while the end criterion is not satisfied
for every ant
Select an antlion using Roulette wheel
Update minimum and maximum for all variables
(at each iteration)
Create a random walk and normalize it
Update the position of ant
end for
Calculate the fitness of all ants
Replace an antlion with its corresponding ant if it
becomes fitter (Eq. (2.12))
Update elite if an antlion becomes fitter than the
elite
end while
return elite.

```

ALO algorithm is able to find a good approximations of the global optimum in the cases of optimization problems, some main reasons that can help this being: random selection of antlions, random walks of ants around them, adaptive shrinking boundaries of antlions' traps, adaptive reduction of ants' movement, the computing of random walks for every ant and every dimension, the comparison between the best antlion in each iteration and the best antlion obtained so far (elite). Note that the random walks of ants and the roulette wheel selection of antlions help avoid stagnating in local optima, and these are very important for population based metaheuristics.

ALO has been used to solve a lot of optimization problems like: automatic generation control of a multi-area system [7], optimum design of skeletal structures [8], optimal load dispatch problem [9], flexible process planning [10], route planning for unmanned aerial vehicle [11], integrated process planning and scheduling [12], structural design problems [13], feature selection [14], performance improvement of distribution network [15], etc.

1.3. Monkey Algorithm (MA) Method

Monkey Algorithm (MA) is a metaheuristic search algorithm proposed by R. Zhao and W. Tang in 2007 [16]. The algorithm simulates the behavior of monkeys as they move over mountains in search of food, assuming a direct proportionality relation between the quantity of food and the height of the mountain.

Thus, the highest mountain corresponds to the solution of the problem in the case of global maximization problems.

There are two types of monkey movements:

- the monkey moves up until it reaches the top of the mountain, the climb being gradually to improve the value of the target function. At the same time, the monkey can make a series of local jumps (local search) in a random direction in the hope of finding a higher mountain;

- after the monkey performs a certain number of climbs and local jumps, the monkey believes that it has sufficiently explored the landscape in the vicinity of its initial position and she wants to explore a new area of the search space. To do this, the monkey performs a long global jump (global search), the goal being to accelerate the convergence rate.

The above steps are repeated a specified number of times, with this number being one of the algorithm's parameters. The solution to the problem is the highest mountain vertex found by the population of monkeys.

The MA algorithm is described below in pseudo code [16]:

Algorithm 2

1. Initialize the monkey population and distribute them randomly over the search space.
2. Measure the height of the monkey position for each monkey.
3. Perform local jumps a fixed number of times.
4. If there is a higher vertex, the monkey makes local jumps from that place.
5. If the limit of the of local jumps is exhausted without found a better vertex, a global jump will be performed
6. If stop criteria didn't meet repeat the entire process from step 2.

Regarding the long global jump, a monkey makes this jump along a line passing through the center of coordinates of all monkeys and its current position, in either direction along this line.

MA can solve a lot of complex optimization problems characterized by non-linearity, non-differentiability and high dimensionality as: robot path planning and vehicle routing [17], transmission network expansion planning [18], solving 0-1 knapsack [19], renewable energy integration [20], clustering analysis [21], feature selection [22], ECE components partitioning [23], task scheduling in cloud [24], packing circles with binary variables [25], solving incapacitated facility location [26], optimal sensor placement for health monitoring [27], identifying essential proteins [28], solving the set covering problem [29], detect neris botnet [30], planning of electrical energy distribution systems [31], drug design [32], flow shop scheduling [33], etc.

2. Proposed Methods for Solving Equations Systems

To solve an ES means to solve a multi-objective optimization problem, that can be defined:

$$\text{minimize } E(X) = (e_1(x), e_2(x), \dots, e_n(x))^T \quad (1)$$

with $x \in A$, where $A \subset \mathbb{R}^n$ is the decision space, $x(x_1, x_2, \dots, x_n) \in A$ is the candidate solution, the function $E(X): A \rightarrow \mathbb{R}^m$ constitutes from m real-valued objective functions.

The main idea of the proposed algorithms is to generate random solutions and improve them with proposed methods until one of termination conditions is satisfied (an exact solution, an approximate solution or a preset number of maximum iterations).

The solution set is represented by the set of all possible solutions with three possibilities: unique, infinitely or no solution. In our approach the proposed methods find the solutions by minimizing a given objective function (fitness function) which shows how good the founded solution is. The condition given by relation $E(X) \geq 0$ has to be met by definition and x^* to be a global minimum of $E(X)$ (an exact solution).

As can be seen in the Experimental results section, our proposed methods are able to find solutions even in cases off over-determinate and under-determinate ES or in cases when classical math methods fail.

2.1. Proposed Ant Lion Optimizer Method for Solving ES

This proposed method is based on the ALO algorithm and is designed to solve linear equations systems.

As input, the proposed algorithm has:

- A, b : matrix, vector, parts of the matrix equation $Ax = b$;
- number of ants (num_ants) that correspond to candidate solutions;
- number of antlions (num_antlions) that correspond to guide solutions;
- dimension represents the size of the solution x ;
- boundaries of each dimension (initial_bounds);
- maximum number of iterations (max_iter).

As output, the proposed algorithm has:

- elite_antlion: best solution found
- number of iterations
- final/best fitness obtained

The pseudocode of the proposed algorithm:

1. Randomly initialize the positions of the ants and antlions, within the interval $a;es$
2. Calculate the fitness for each ant and antlion according to the relation: $\text{fitness}(x) = \text{MSE}(Ax, b)$
3. Choose the antlion with the best fitness as elite_antlion
4. While termination criteria aren't met (maximum number of iteration or a good enough fitness)
 - 4.1. Calculate the intensity factor I (exploration control)
 - 4.2. For each ant:
 - Select a guide antlion using roulette selection (probability $\propto 1/\text{fitness}$)
 - For each dimension j :
 - Calculate the min and max bounds for the random movement, using the position of the elite and the selected antlion
 - Generate two unidimensional random walks: one guided by the selected antlion and the other guided by the elite
 - Normalize the result to fit into $[\min, \max]$
 - Update the ant's position in that dimension
 - Calculate the fitness for the ant's new position
 - 4.3. If an ant has a better fitness than the corresponding antlion replace that antlion with that ant
 - 4.4. If there is a better solution than elite_antlion update elite_antlion and its fitness
 - 4.5. If elite_antlion has not improved enough times reinitialize the weak antlions with random positions.

2.2. Proposed Monkey Algorithm (MA) Method for Solving ES

This proposed method is based on the MA algorithm and is designed to solve linear equations systems.

As input, the proposed algorithm has:

- A, b : matrix and vector parts of the matrix equation $Ax = b$;
- num_monkeys: number of monkeys (candidate solutions);
- dimension: size of the vector solution x ;
- initial_bounds: limits of possible values for x ;
- max_iterations: maximum number of iterations.

As output, the proposed algorithm has:

- best_monkey: the best solution found
- number of iterations
- final fitness

The pseudocode of the proposed algorithm:

1. Initialize the monkey population with random values in [initial_bounds[0], initial_bounds[1]]
2. Calculate the fitness of each monkey: $\text{fitness}(x) = \text{MSE}(Ax, b)$
3. Find the monkey with the best fitness and save it as best_monkey
4. While termination criteria aren't met (maximum number of iteration or a good enough fitness)
 - 4.1 Adjust the parameters to control learning and exploration depending on the number of iterations
 - 4.2 For each monkey (except the best one):
 - Apply Climbing (SPSA) for local improvement
 - Apply Watch Jump to jump to a nearby region, if fitness improves
 - Apply Climbing again
 - Apply Somersault to explore the area around the population center of gravity
 - 4.3 Recalculate the fitness for the entire population
 - 4.4 If a better solution was found, update best_monkey
 - 4.5 If it has not improved after many iterations, renew some monkeys that have very poor fitness with variations of best_monkey
- end while.

3. Experimental Results

In our research, all algorithms (AntLion, Monkey, Conjugate gradient, Gaussian elimination with total pivoting, Gauss-Seidel and Broyden) were implemented in the Python language using PyCharm 2024.3.5. The used computer was Dell Inspiron 5515 AMD Ryzen 7 5700U with Radeon Graphics, 1801 Mhz, 8 Core(s), 16 Logical Processor(s) 16 Gb RAM, OS Microsoft Windows 11 Home.

Four classical methods for solving linear equations systems were used for comparison: Conjugate gradient, Gaussian elimination with total pivoting, Gauss-Seidel and Broyden, implemented in Python, like ALO and MA.

In our experiments, for the systems of equations, both eigen matrices and public matrices from The Harwell-Boeing Sparse Matrix Collection were used, the latter being standard test matrices arising from problems in linear systems <https://math.nist.gov/MatrixMarket/collections/hb.html>.

The working parameters for our proposed methods were:

ALO:

A. Parameters (User-defined or from the problem configuration)

1. num_ants (int): Number of ants (candidate solutions) in the population.
2. num_antlions (int): Number of predator ants (antlions) in the population.
3. dimension (int): Problem dimension (number of variables).
4. bounds (list of floats): Search bounds for each dimension $[-1, 1]$.
5. max_iterations (int): Maximum number of iterations.
6. A (numpy matrix): Matrix of coefficients in the system of equations $Ax=b$.
7. b (numpy vector): Vector of free terms in the system $Ax=b$.

B. ALO algorithm specific settings

1. fitness_function(ant, A, b): Evaluation function of each solution, calculating the mean square error (MSE).

2. roulette_wheel_selection(fitness): Predator ant selection based on inverted fitness.

3. value_for_I(t, T_max): Parameter that determines the aggressiveness of predators depending on the progress of the iteration.

$$I = (10^{**w}) * (t/T_max)$$

The values of w vary as follows:

$$w=6 \text{ if } 0.95 * T_max \leq t \leq T_max$$

$$w=5 \text{ if } 0.9 * T_max \leq t < 0.95 * T_max$$

$$w=4 \text{ if } 0.75 * T_max \leq t < 0.9 * T_max$$

$$w=3 \text{ if } 0.5 * T_max \leq t < 0.75 * T_max$$

$$w=2 \text{ if } 0.1 * T_max \leq t < 0.5 * T_max$$

$$w=1 \text{ otherwise}$$

4. minimum_c_i, maximum_d_i (numpy array): Lower and upper bounds adjusted based on selection.

5. minimum_c_e, maximum_d_e (numpy array): Lower and upper bounds adjusted based on elite.

6. unchanged_steps (int): Number of iterations without improvement of the elite.

MA:

A. Parameters

1. num_monkeys (int): Number of monkeys (solutions) in the population.
2. dimension (int): Number of variables in the problem.
3. initial_bounds (list of floats): Lower and upper bounds for each dimension $[-1, 1]$
4. max_iterations (int): Maximum number of iterations of the algorithm.

5. **A** (numpy matrix): Matrix of coefficients for the system of equations.

6. **b** (numpy vector): Right-hand side of the equation $Ax=b$

B. Hyperparameters (Algorithm-specific settings)

- SPSA (Stochastic Perturbation Stochastic Approximation) climb

7. **A_const** (float): A small constant for gradually reducing the learning rate in SPSA (0.10).

8. **alpha** (float): Exponent for reducing the learning rate (0.6).

9. **gamma** (float): Exponent for perturbation reduction (0.1).

10. **a** (float): Initial step size in SPSA (dynamically adjusted).

11. **c** (float): Initial perturbation size in SPSA (dynamically adjusted).

- Watch-Jump

12. **b_eye** (float): Random jump interval (dynamically adjusted).

13. **improvement_threshold** (float): Minimum fitness improvement required to accept a new jump ($1e-2$).

14. **max_attempts** (int): Maximum number of attempts to improve fitness by watch-jump (10).

- Somersault Process

15. **alpha_range** (tuple of float): Random coefficient range for somersault step ($-0.01, 0.01$)

16. **somersaultinterval** (list of float): Update interval for somersault step $[-0.01, 0.01]$

- Parameter tuning during iterations

17. **factor** (int): Parameter scaling factor, depending on the problem size.

18. **a_initial** (float): Initial learning rate 0.1/factor.

19. **a_final** (float): Final learning rate 0.01/factor.

20. **c_initial** (float): Initial perturbation size 0.1/factor.

21. **c_final** (float): Final perturbation size 0.01/factor.

22. **b_eye_initial** (float): Initial interval for random jumps 0.2/factor.

23. **b_eye_final** (float): Final interval for random jumps 0.02/factor.

24. **decay** (float): Parameter reduction factor based on progress iteration/max_iterations

- Stagnation handling

25. **unchanged_steps** (int): Number of consecutive iterations without improvement.

3.1. First Set of Experiments

The first set of experiments aimed to verify whether the proposed methods find the correct solution, identical or very close to those found by traditional methods such as Conjugate gradient, Gaussian elimination, Gauss-Seidel and Broyden. We also wanted to see if our proposed methods are capable of finding a solution for systems of linear equations when traditional methods fail. We also wanted to see if our proposed methods can find solutions for overdetermined and underdetermined systems (systems 10 and 11 in Table 1). Some experimental results could be seen in Table 1. As conclusions to this first set of experiments, the proposed MA and ALO methods always find a solution (MA generally with better accuracy), even when for various reasons, traditional methods fail. Moreover, our proposed methods can find solutions for overdetermined and underdetermined systems.

Table 1. Experimental results-solutions and their accuracy

No.	Equations system	MA solution accuracy	ALO solution accuracy	CG	GE	GS	B
1	bcpwr01 *	√ 0.01	√ 0.1	√	√	√	√
2	can_24 *	√ 0.001	√ 0.1	√	√	√	√
3	jgl011 *	√ 0.0001	√ 0.001	√	X det(A)=0	X Zero on diagonal	√
4	jgl009 *	√ 0.0001	√ 0.001	√	X det(A)=0	X Zero on diagonal	√
5	will57 *	√ 0.0001	√ 0.01	√	X det(A)=0	√	√
6	ibm32 *	√ 0.1	√ 0.01	√	√	√	√
7	pores_1 *	√ 1	√ 1	√	√	√	√
8	$x + 2y = -5$ $3x - y = 13$	√ 0.00001	√ 0.01	√	√	X doesn't conv.	√
9	$x - y + z = 1$ $x + y + z = 3$ $-x + y + z = -3$	√ 0.00001	√ 0.01	X doesn't conv.	√	X doesn't conv.	√
10	$5x+4y-z=8$ $x-y+z=4$ Underdetermined system	√ 0.00001 an infinity of solutions, with the relationships between the roots $x_1=8/3 - (1/3)x_3$ $x_2=-4/3+2/3)x_3$ x_3 – free	√ 0.01 an infinity of solutions, with the relationships between the roots $x_1=8/3 - (1/3)x_3$ $x_2=-4/3+2/3)x_3$ x_3 – free	X not squared matrix	X not squared matrix	X not squared matrix	X not squared matrix
11	$x+y=2$ $3x-y=2$ $-x+y=0$ Overdetermined system	√ 0.00001	√ 0.01	X not squared matrix	X not squared matrix	X not squared matrix	√ Only one solution

Legend:

MA – Proposed Monkey Algorithm

ALO - Proposed AntLion Optimization

CG - Conjugate Gradient

GE - Gaussian Elimination

GS - Gauss-Seidel

B - Broyden

√ - solution found

X - NO solution found

*- matrices from <https://math.nist.gov/MatrixMarket/collections/hb.html>

3.2. Second Set of Experiments

In the second set of experiments, we wanted to see how feasible the proposed methods are compared to traditional methods in terms of execution time. Some experimental examples can be seen in Table 2.

From the execution time (runtime) point of view, the experimental results are not encouraging regarding the metaheuristics. Also, a final conclusion cannot be drawn regarding a ranking of the two metaheuristics in relation to each other, the random functions and algorithms strategies used being the main causes. This can be seen on Table 2.

Table 2. Experimental results-runtime in seconds

No.	Equations system	MA (average)	ALO (average)	CG	GE	GS	B
1	bcpwr01 *	76	99	< 1	< 1	< 1	< 1
2	can_24 *	64	60	< 1	< 1	< 1	< 1
3	jgl011 *	75	33	< 1	X	X	< 1
4	jgl009 *	49	18	< 1	X	X	< 1
5	will57 *	48	454	< 1	X	< 1	< 1
6	ibm32 *	70	145	< 1	< 1	< 1	< 1
7	pores_1 *	307	783	< 1	< 1	< 1	< 1

3.3. Third Set of Experiments

It is clear that the execution time is directly proportional to the size of the problem to be solved and to the search space in the case of metaheuristics. In this third set of experiments we wanted to find out for each metaheuristic in part, how:

- the fitness is affected by the dimension of the population;
- the fitness is affected by the number of iterations;
- the number of necessary iterations for a given fitness is affected by the dimension of the population.

Thus, in the case of each system of equations, for each instance we performed 100 tests, with the average values being used in the graphical representations.

Moreover, for each metaheuristic, for each of the 100 runs performed, we generated different initial populations, with differences of at least 20%.

Regarding the influence of population size on fitness, the experiments show the following:

- a larger population improves the fitness value;
- from a certain population size, no major improvements in the fitness value were observed
- the MA algorithm ensures better fitness values compared to those obtained with the AntLion algorithm, approximately 10^3 times;
- the MA algorithm seems to have better convergence than the ALO algorithm, as can be seen in Figure 1 where the descending part at MA has a higher slope than at ALO.

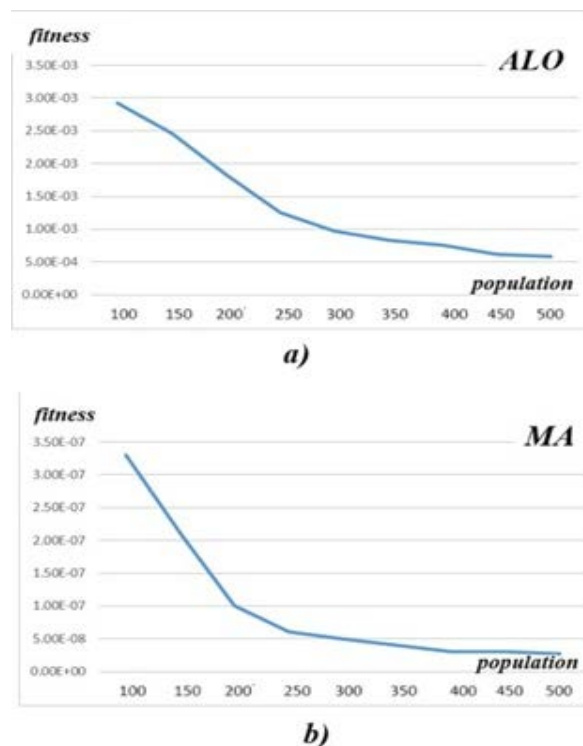


Figure 1. Fitness (population) for will57 ES

Regarding the influence of iterations number on fitness, the experiments show the following:

- a bigger number of iterations improves the fitness value;

-from a certain number of iterations, no major improvements in the fitness value were observed.

Such an experiment for the willk57 equations system could be seen in Figure 2.

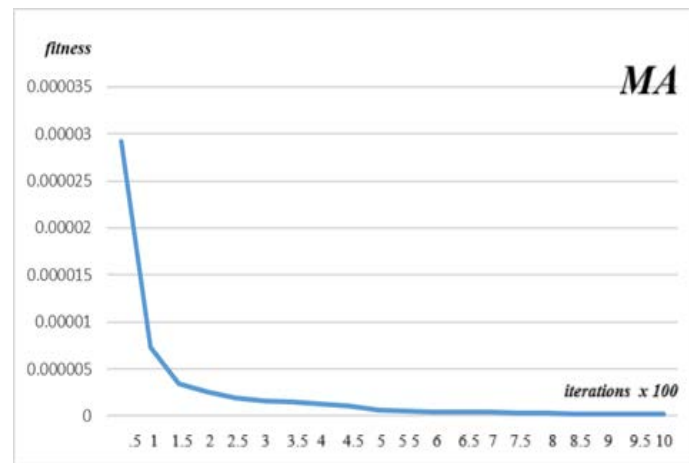


Figure 2. Fitness (iterations) for will57 ES

4. Conclusions and Future Work

In this paper, two metaheuristics inspired from nature ALO and MA were used to solve linear equations systems. The optimization is viewed as a minimization problem that uses an objective function. It was experimentally proven that the both algorithms can be used, with the necessary adaptations, to solve linear ES, having the ability to find a fairly good solution compared to the solutions found by classical methods. Our proposed algorithms have a good accuracy and were able to find a solution even in cases where math methods fail. As a final conclusion, the MA algorithm, after experiments, seems to have a better convergence than the ALO algorithm.

The parallelization of the proposed algorithms and the addition of the ability to find all solutions that exist within a given interval will be our future research directions.

References:

- [1]. Kelley, C. T. (1995). *Iterative methods for linear and nonlinear equations*. Society for Industrial and Applied Mathematics.
- [2]. Maftciu-Scai, L., Maftciu, E., & Maftciu-Scai, R. (2018). Solving Equations Systems Using Artificial Intelligence--a Survey. *International Journal of New Computer Architectures and their Applications*, 8(3), 102-119.
- [3]. Rajwar, K., Deep, K., & Das, S. (2023). An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges. *Artificial Intelligence Review*, 56(11), 13187-13257. Doi: 10.1007/s10462-023-10470-y
- [4]. Gutjahr, W. J. (2009). Convergence analysis of metaheuristics. *Matheuristics: hybridizing metaheuristics and mathematical programming*, 159-187. Springer. Doi: 10.1007/978-1-4419-1306-7_6
- [5]. Thomaidis, N. S., & Vassiliadis, V. (2013). Stochastic convergence analysis of metaheuristic optimisation techniques. *Towards advanced data analysis by combining soft computing and statistics*, 343-357. Springer.
- [6]. Mirjalili, S. (2015). The ant lion optimizer. *Advances in engineering software*, 83, 80-98. Doi: 10.1016/j.advengsoft.2015.01.010
- [7]. Saikia, L. C., & Sinha, N. (2016). Automatic generation control of a multi-area system using ant lion optimizer algorithm based PID plus second order derivative controller. *International Journal of Electrical Power & Energy Systems*, 80, 52-63.
- [8]. Talatahari, S. (2016). Optimum design of skeletal structures using ant lion optimizer. *International Journal of Optimization in Civil Engineering*, 6(1), 13-25.
- [9]. Nischal, M. M., & Mehta, S. (2015). Optimal load dispatch using ant lion optimization. *Int J Eng Res Appl*, 5(8), 10-19.
- [10]. Petrović, M., et al. (2015). The ant lion optimization algorithm for flexible process planning. *Journal of Production Engineering*, 18(2), 65-68.
- [11]. Yao, P., & Wang, H. (2017). Dynamic Adaptive Ant Lion Optimizer applied to route planning for unmanned aerial vehicle. *Soft Computing*, 21, 5475-5488.
- [12]. Petrović, M., et al. (2016). The ant lion optimization algorithm for integrated process planning and scheduling. *Applied Mechanics and Materials*, 834, 187-192.
- [13]. Yıldız, B. S. (2017). A comparative investigation of eight recent population-based optimisation algorithms for mechanical and structural design problems. *International Journal of Vehicle Design*, 73(1-3), 208-218.
- [14]. Zawbaa, H. M., Emary, E., & Grosan, C. (2016). Feature selection via chaotic antlion optimization. *PloS one*, 11(3), e0150652.
- [15]. Soesanti, I., & Syahputra, R. (2022). Multiobjective ant lion optimization for performance improvement of modern distribution network. *IEEE Access*, 10, 12753-12773.
- [16]. Zhao, R., & Tang, W. (2008). Monkey algorithm for global numerical optimization. *Journal of Uncertain Systems*, 2(3), 165-176.
- [17]. Devi, R. V., Sathya, S. S., & Kumar, N. (2017). Monkey algorithm for robot path planning and vehicle routing problems. *2017 International Conference on Information Communication and Embedded Systems (ICICES)*, 1-6.

- [18]. Wang, J., et al. (2010). Discrete monkey algorithm and its application in transmission network expansion planning. *IEEE PES General Meeting*, 1-5.
- [19]. Zhou, Y., Chen, X., & Zhou, G. (2016). An improved monkey algorithm for a 0-1 knapsack problem. *Applied Soft Computing*, 38, 817-830.
- [20]. Ituarte-Villarreal, C. M., Lopez, N., & Espiritu, J. F. (2012). Using the monkey algorithm for hybrid power systems optimization. *Procedia Computer Science*, 12, 344-349.
- [21]. Chen, X., Zhou, Y., & Luo, Q. (2014). A hybrid monkey search algorithm for clustering analysis. *The Scientific World Journal*, 2014(1), 938239.
- [22]. Hafez, A. I., et al. (2015). Hybrid monkey algorithm with krill herd algorithm optimization for feature selection. *2015 11th international computer engineering conference (ICENCO)*, 273-277.
- [23]. Kuliev, E., & Kureichik, V. (2018). Monkey search algorithm for ECE components partitioning. *Journal of Physics: Conference Series*, 1015(4), 042026.
- [24]. Gupta, P., & Tewari, P. (2017). Monkey search algorithm for task scheduling in cloud IaaS. *2017 Fourth International Conference on Image Information Processing (ICIIP)*, 1-6.
- [25]. Torres-Escobar, R., et al. (2019). Monkey algorithm for packing circles with binary variables. *Intelligent Computing & Optimization I*, 547-559. Springer.
- [26]. Atta, S., Mahapatra, P. R. S., & Mukhopadhyay, A. (2018). Solving uncapacitated facility location problem using monkey algorithm. *Intelligent Engineering Informatics: Proceedings of the 6th International Conference on FICTA*, 71-78. Springer.
- [27]. Yi, T. H., et al. (2015). Optimal sensor placement for health monitoring of high-rise structure based on collaborative-climb monkey algorithm. *Structural Engineering and Mechanics*, 54(2), 305-317.
- [28]. Payra, A. K., & Ghosh, A. (2020). Identifying essential proteins using modified-monkey algorithm (MMA). *Computational Biology and Chemistry*, 88, 107324.
- [29]. Crawford, B., et al. (2020). A binary monkey search algorithm variation for solving the set covering problem. *Natural Computing*, 19(4), 825-841.
- [30]. Saleh, I. A., et al. (2020). Using monkey optimization algorithm to detect neris botnet. *Journal of Physics: Conference Series*.
- [31]. Duque, F. G., et al. (2020). Modified monkey search technique applied for planning of electrical energy distribution systems. *Frontier Applications of Nature Inspired Computation*, 240-265.
- [32]. Devi, R. V., et al. (2019). Multi-objective monkey algorithm for drug design. *International Journal of Intelligent Systems and Applications*, 11(3), 31.
- [33]. Marichelvam, M. K., Tosun, Ö., & Geetha, M. (2017). Hybrid monkey search algorithm for flow shop scheduling problem under makespan and total flow time. *Applied Soft Computing*, 55, 82-92.