Moth-Flame Optimization and Ant Nesting Algorithm: A Systematic Evaluation

Hanan K. AbdulKarim[1(✉)] and Tarik A. Rashid[2]

[1] Software Engineering Department, College of Engineering, Salahaddin University - Erbil, Erbil, Iraq
hanan.abdulkarim@su.edu.krd

[2] Computer Science and Engineering Department, University of Kurdistan Hewler, Erbil, Iraq

Abstract. In this paper, some swarm metaheuristic algorithms are analyzed to show the performance of algorithms. Swarm Intelligence algorithms are more trapping in local optima because of low exploration. The Moth-Flame Optimization algorithm is a widely applied metaheuristic algorithm. Nevertheless, the Ant Nesting Algorithm is another recent powerful algorithm. Both algorithms are theoretically and practically studied and applied to a simple optimization problem with a simple objective function. All steps of the algorithms are implemented and discussed providing results to show and compare the exploration and exploitation performance between both algorithms. A simple comparison between both algorithms is conducted using the same sample of data, and it is concluded that convergence within cycles of implementation shows that Ant Nesting Algorithm is fast converged but it might get stuck in local optima because of low exploration. Additionally, the merits of the Ant Nesting Algorithm show the algorithm will stuck in local optima when solving highly complex problems or if the initial population can't be explored efficiently. Moth-Flame Optimization also may have exploration problems when the size of flames during cycles is decreased.

Keywords: Metaheuristics · Moth-Flame Optimization · Ant Nesting Algorithm

1 Introduction

Optimization is everywhere, the way of finding the best solution among many good solutions. There are many optimization algorithms to solve the problems. Metaheuristic algorithms are most used to solve complex and large multimodal problems. A multimodal problem needs a global search to find the best solution overall search space. The metaheuristic algorithm is higher level than the heuristic, it works better than the heuristic. Metaheuristic algorithms use randomness and local search. Randomization gives a good way to escape from local search to search in global search space, therefore all metaheuristics are suitable for global searching space [1]. Major components of metaheuristic algorithms are intensification and diversification, search during a local search by selecting the best solution will converge the algorithm to the optimal solution, which increases intensification, while using randomization will lead to the diverging algorithm from local optima and search the space globally, which increase diversification [1].

N. Bacanin and H. Shaker (Eds.): ICIITB 2022, ACSR 104, pp. 139–152, 2023.
https://doi.org/10.2991/978-94-6463-110-4_11

Many creatures' behaviors inspired by optimization algorithms almost are metaheuristic because their behaviors are stochastics. Ant Colony Optimization algorithm (ACO) [2] works on the ants finding a path to the food sources, Ant Nesting Algorithm (ANA) mimics ants choosing a place to build a nest [3], Bat Algorithm (BA) [4] works on how bat hunts food, Artificial Bee Colony (ABC) [5] finds food sources, Particle Swarm Optimization (PSO) [6] mimics swarm of bird's behavior, Moth-Flame Optimization algorithm (MFO) [7] represents the moths flying around lights. All the above algorithms work depend on creatures' changing position. Many metaheuristic algorithms suffer from low exploration and many of them improved by applying levy flight to add more randomness within the algorithm like ACO improved with levy flight in [8, 9], BA in [10, 11], ABC for multi-objective improved by levy flight in [12] and also ABC improved by Bayesian estimation in [13], exploration of PSO improved by levy flight also [14, 15], and in recent researches, MFO also improved by adding more randomness, for example, add levy flight in [16, 17]. However, some algorithms had fast convergence to the best solutions while leading to local optima.

Recently many problems were solved by metaheuristic algorithms like some detection problems in [18–21], swarm intelligence used to solve classification problems in [22–24], modified optimization algorithms used in prediction [25], workflow scheduling problems in a cloud-edge environment enhanced by some swarm algorithm in [26], metaheuristic and swarm-based algorithm are used to solve deep learning and wireless device problems respectively in [27, 28].

The objective of this paper is to study and implement both MFO and ANA with a simple optimization problem, implementation of algorithms within simple objective function supposed and steps are written in detail to show the algorithms running result, and some comparisons for both algorithms with the same initial population and parameters are conducted.

The following section of the paper includes a background of algorithms MFO and ANA as methodology in Sect. 2, implementation of case study in Sect. 3, then discussion and conclusion are provided in the last section.

2 Methodology

2.1 MFO

MFO is explained in this section depending on [7]. Two components are described here:

2.1.1 Moths

The candidate solutions are assumed to be moths representing the positions in search, depending on the dimension of the problems the dimension could be *1D, 2D,3D…* etc. and represented as follows:

M(i,j), is matrix used to store moth solutions where $i = 1,2, \ldots,n,$ and $j = 1,2,\ldots,d.$

n is the number of moths and number of solutions at the same time, and d is the number of dimensions of search space. The population of the moth is:

$$\mathrm{M} = \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1d} \\ \vdots & & \ddots & \vdots \\ m_{n1} & m_{n2} & \cdots & m_{nd} \end{bmatrix} \tag{1}$$

Each solution in the moth has an optimization function as an objective and it stores as a vector of n solutions for n moths, *OM* is a vector used to store the fitness of corresponding moths where $i = 1,2,\ldots,n,$ represented as follows:

$$\mathrm{OM} = \begin{bmatrix} om_1 \\ om_2 \\ . \\ . \\ om_n \end{bmatrix} \tag{2}$$

The n is equal to the number of moths, which is equal to the number of solutions.

2.1.2 Flames

There are also a set of flames that represents the best solution of moths found so far, the flames are represented by matrix *F(i,j)*, which is a matrix used to store moth solutions where $i = 1,2, \ldots,n,$ and $j = 1,2,\ldots,d$. n is the number of moths and number of solutions at the same time, and d is the number of dimensions of search space. The flame's representations are similar to the moth as follows:

$$\mathrm{F} = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1d} \\ \vdots & & \ddots & \vdots \\ f_{n1} & f_{n2} & \cdots & f_{nd} \end{bmatrix} \tag{3}$$

Each solution in the moth has an optimization function as an objective and it stores as a vector of n solutions for n moths, *OF*, is a vector used to store the fitness of corresponding flames where $i = 1,2,\ldots,n,$ represented as follows:

$$\mathrm{OF} = \begin{bmatrix} of_1 \\ of_2 \\ . \\ . \\ of_n \end{bmatrix} \tag{4}$$

The n is equal to the number of the moths at the start of the run, but later n will be calculated depending on the formula.

$$\textit{flame no} = \textit{round}\ (N - c * \frac{N-1}{T}) \tag{5}$$

where N is the maximum number of flames, c is the current number of an iteration, and T is the maximum number of iterations. Both moths and flames are solutions, but the difference is the way the modifications are in each iteration. Moths search around the space, while flames are the best solution for moths (local best).

Initial Solutions

The initial solution of moths and fitness of moths are generated using the following statements in algorithm 1:

Algorithm (1) Random Solution Generator

```
foreach i : n
    foreach j : d
        M(i,j) = ( ub(i) -lb(i) ) * rand() + lb(i)
    end
end
```

Moths are generated randomly using LB and UB, which are lower and upper bounds respectively, the bounds are two vectors of decisions:

$$UB = [ub_1,\ ub_2, \ldots\ldots\ldots,\ ub_n]; \quad LB = [lb_1, lb_2, \ldots\ldots\ldots, lb_n]$$

During the algorithm, running moth searches the space depending on flames, which is the best solution of previous iterations. The update mechanism of moths depends on formula (6), where *Mi* is moth and *Fi* is flame, $\mathrm{S}(M_i, F_j)$ represents spiral function using formula (7) to find updated solution depending on the distance between flame and moth using formula (8)::

$$\mathrm{M(i,\ j)}\ = \mathrm{S}(Mi, Fj) \tag{6}$$

$$\mathrm{S}(M_i, F_j) = D_i.e^{bt}\cos(2\pi t) + F_j) \tag{7}$$

D_i is the distance calculated as:

$$Di = \left|F_j - M_i\right| \tag{8}$$

In the formula (7) t is a random number in the range [–1,1], and b is the constant value $b = 1$ in [2]. The flowchart of MFO is shown in Fig. 1.

2.2 ANA

The algorithm steps show ants building their nests [2]. The important part of the algorithm is working ants, which search for a suitable place and build the nest for deposition. The algorithm starts with a random solution representing ants $X_i(i = 1,2,3,\ldots..n)$, each worker ant i try to find a better position than the current position, for these, it is working iteratively, in each iteration it finds the deposition rate change and adds to current deposition of positions to find a new position, the new position of a worker ant is changed depending formula (9):

$$\mathrm{X_{t+1,i}} = \mathrm{X_{t,i}} + \Delta\,\mathrm{X_{t+1,i}} \tag{9}$$

where t represents the current iteration, i current worker ant and X is a solution in the algorithm (which is deposition position), and $\Delta X_{t+1,i}$ can be calculated using the following formulas:

$$\Delta X_{t+1,i} = dw \times (X_{t,ibest} - X_{t,i}) \tag{10}$$

If the current ant is equal to local best, then formula (11) is used to calculate $\Delta X_{t+1,i}$, otherwise if the current solution is equal to the previous solution, then formula (12) is used to calculate $\Delta X_{t+1,i}$. Where dw is deposition weight can be calculated using formula (13).

$$\Delta X_{t+1,i} = r \times X_{t,i} \tag{11}$$

$$\Delta X_{t+1,i} = r \times \left(X_{t,ibest} - X_{t,i}\right) \tag{12}$$

$$dw = r \times \left(\frac{T}{Tprevious}\right) \tag{13}$$

where r is a random number in the range [–1, 1], T is the tendency rate that can be calculated using the Pythagorean theorem, and formulas (14) and (15) are used to calculate T and $T\,previous$ simultaneously:

$$T = \sqrt{\left(X_{t,ibest} - X_{t,i}\right)^2 - \left(X_{t,ibest}fitness - X_{t,i}fitness\right)^2} \tag{14}$$

$$T_{prevoiuos} = \sqrt{\left(X_{t,ibest} - X_{t,ipreviously}\right)^2 - \left(X_{t,ibest}fitness - X_{t,ipreviously}fitness\right)^2} \tag{15}$$

A simple representation of algorithm steps is represented as a flowchart in Fig. 2.

3 Case Study

Optimization problems are either maximization or minimization, in this work, a minimization problem is considered, which $f(x) = \sum_{i=1}^{D} x_i^2$, where $D = 2$, which is the dimension of a problem for the current example, and x in the range $[-100, 100]$, two iterations considered, $N = 3$ is the population size, the step implementation for both MFO and ANA algorithms are shown in the following sections:

3.1 MFO Implementation Steps

Apply MFO for the study case for two iterations as supposed above, finally result for iterations are explained as follows:

Initialize parameters: $b = 1$, $r = 0.3$ (randomly generated during algorithm running).

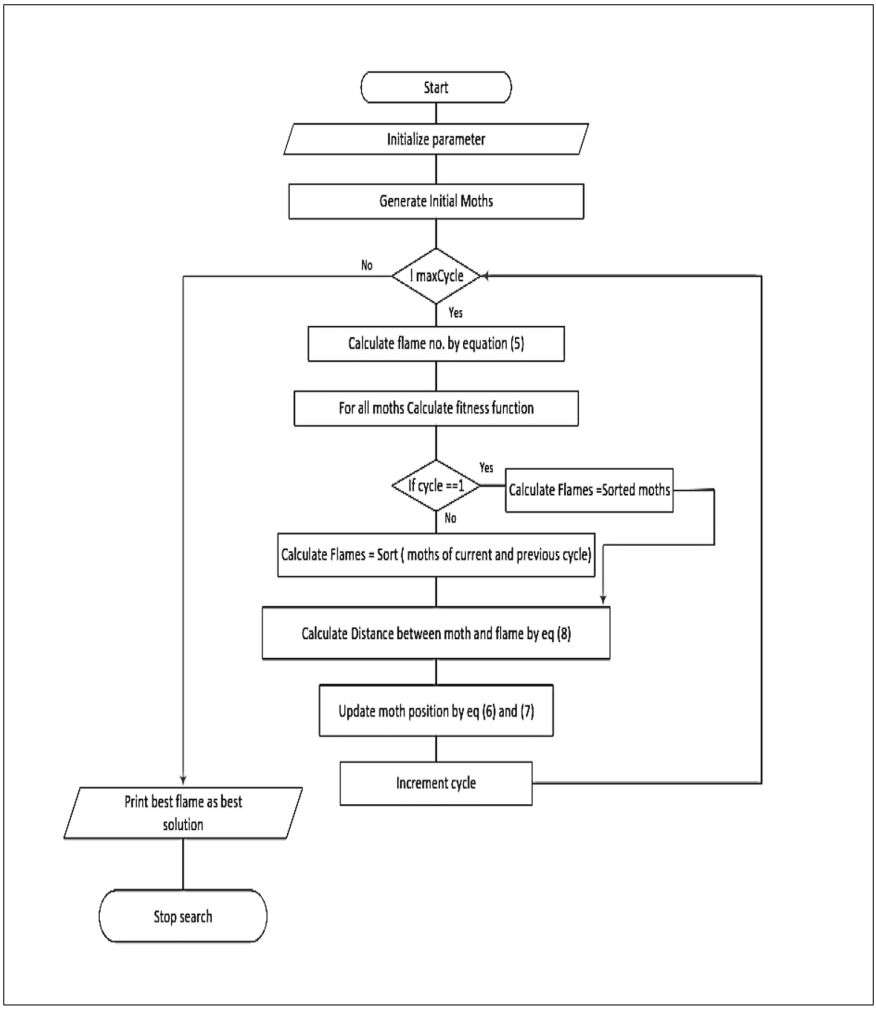

Fig. 1. Moth-Flame Optimization Flowchart

Initial solutions randomly generate as shown in Table 1:

– Iteration 1:

For this example, there is no flame number calculation because from the beginning supposed the flame number = 3, if generate the flame number using the formula (5).

$flameno = round\left(3 - 1 * \frac{3-1}{2}\right) = 1$, the result is just one flame and every moth should update its position depending on this solution.

Generate flames by sorting moths of the initial solution because it is the first iteration, the result of flames is shown in Table 2.

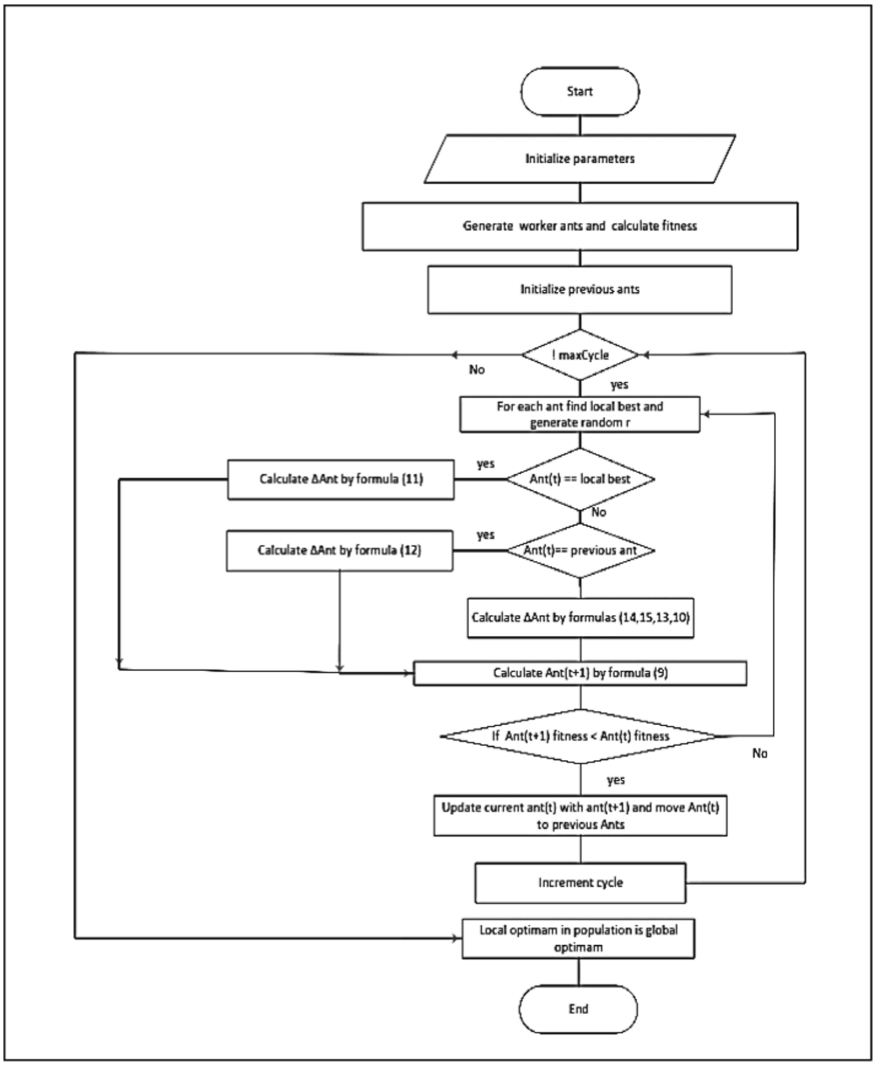

Fig. 2. Simple Ant Nesting Algorithm Flowchart

Table 1. Moths during the first iteration

Moth	X_1	X_2	$F(x)$	Rank
1	-2	3	13	3
2	-1	0	1	1
3	1	1	2	2

Table 2. Flames during the first iteration

Flame	X_1	X_2	F(x)
1	-1	0	1
2	1	1	2
3	-2	3	13

Table 3. Moths generated for the next iteration

Moth	X_1	X_2	F(x)	Rank
1	-0.5	-1.5	2.5	2
2	1	1.5	3.3	1
3	-3.5	4	28.5	3

Calculate movement M(i, j) depending on formula (6), calculate the distance of each moth with the corresponding flame using formula (8), then apply formula (7) to find

$$S(M_i,\ F_j) = D_i.\, e^{bt} \cos(2\pi\, t) + F_j$$

D_i is the distance calculated as:

$$D_i = |F_j - M_i|$$

Moth1 $D1, 1 = |(-1, 0) - (-2, 3)| = (1, -3)$

$S(M1, F1) = (1, -3)$ *e^(0.3) * $\cos(2 * \pi * 0.3) + (-1, 0) = (-0.5, -1.5)$

Moth2 $D2, 2 = |(1, 1) - (-1, 0)| = (2, 1)$

$S(M1, F1) = (2, 1)$ *e^(0.3) * $\cos(2 * \pi * 0.3) + (-1, 0) = (1, 1.5)$

Moth 3 $D3, 1 = |(-2, 3) - (1, 1)| = (-3, 2)$

$S(M1, F1) = (-3, 2) * e^{\wedge}(0.3) * \cos(2 * \pi * 0.3) + (-1, 0) = (-3.5, 4)$

Calculate fitness for generate moth and the result are shown in Table 3 as new moths and their fitness:

– Iteration 2:

Moths generated in Table 3 are used for this iteration, and the number of flames is also 3. Generate flames by sorting moths of the current iteration in Table 3 and previous iteration flame in Table 2 (where it is a moth of the first iteration after sorting) because

Table 4. Flames generated for the second iteration

Flame	X_1	X_2	F(x)
1	-1	0	1
2	1	1	2
3	-0.5	-1.5	2.5

Table 5. Moths generated for the new iteration

Moth	X_1	X_2	F(x)	Rank
1	-1.3	0.8	2.1	2
2	1	0.7	1.5	1
3	1.3	-2.8	9.3	3

it is not the first iteration, the result of flames for the current iteration is shown in Table 4.

The same first iteration calculates movement M(i, j) depending on formula (6), calculating the distance of each moth with the corresponding flame using formula (8), then calculating the moths' new position depending on formula (7), the distance and result for each moth, is generated after changing its position is represented below $D_{1..3}$ are distances, and $moth_{1..3}$ are positions of moths:

D1 = (-0.5, 1.5)
D2= (0, -0.5)
D3= (3.5, -2.5)
Moth 1 = (-1.3, 0.8)
Moth2 = (1, 0.7)
Moth 3 = (2.3, -2.8)

The result of moths generated in iteration 2, which will be used for the next iteration shown in Table 5.

The best solution for two iterations is flame 1 from the last iteration, which is: $M(-1, 0) and f(x) = 1$.

3.2 ANA Implementation Steps

Appling ANA for the study case for two iterations as assumed above finally results for iterations are explained as follows:

At the beginning, we generate the initial solution as population (worker ant), the result of population is shown in Table 6, which assumes the same initial solution generated for MFO in Table 1:

For the first iteration, the previous population is also the current population, which is shown in Table 6.

Table 6. Initial worker Ants generated randomly

Ant	X_1	X_2	F(x)	note
1	-2	3	13	
2	-1	0	1	Local Best
3	1	1	2	

– Iteration 1:

For each ant, find the local best ant, which is the local best solution for the current iteration, and generate random r = 0.3 (suppose it is generated randomly for every ant).

Ant 1,1 = (-2,3), which represents the first iteration, the first ant, and it is equal to the previous solution depending on algorithm steps, then formula (12) will be used to generate ants step size and generate ant 1 for iteration 2 depending formula (9) for next iteration:

$\Delta X_{2,1} = r \times (X_{1,1best} - X_{1,1})$
$\Delta X = 0.3((-1, 0) - (-2, 3)) = (0.3, -0.9)$
$X_{2,1} = Ant_{1,1} + \Delta\ Ant_{2,1}$
$X2, 1 = (-2, 3) + (0.3, -0.9) = (-1.7, 2.1)(newsolution)$
$F(X2, 1) = 7.3$

F(X 2,1) is better than F(X1,1) then the solution of Ant 2,1 will be accepted for the next iteration.

Ant 1,2 = (-1,0) which represents the first iteration, the second ant, and it is equal to a local best solution then formula (11) will be used to generate step size, which is used to generate ant 2 for iteration 2 using formula (9), $r = 0.3$ (randomly):

$\Delta X_{1,2} = r \times X_{1,2}$
$\Delta X = 0.3*(-1, 0) = (-0.3, 0).$
X2,2 = (−1,0) + (−0.3,0) = (−1.3,0)(*new solution*)
$F(X2, 2) = 1.69$

F(Ant 2,2) is not better than F(Ant 1,2) then the solution of Ant 2,2 will not be accepted for the next iteration.

Ant 1, 3 = (1,1), which represents the first iteration, the third ant, and it is equal to the previous solution depending on algorithm steps, then formula (12) will be used to generate ants step size and generate ant 1 for iteration 2 depending on formula (9) for next iteration, suppose r = 0.3 (randomly):

$\Delta X_{2,3} = r \times (X_{1,3best} - X_{1,3}).$
$\Delta X = 0.3*((-1, 0) - (1, 1)) = (-0.6, -0.3).$
X2,3 = (1,1) + (−.06, − 0.3) = (0.4, 0.7) (*new solution*)
$F(X2, 2) = 0.65$

Table 7. Ants generated for the second iteration

Ant	X_1	X_2	F(x)	note
1	-1.7	2.1	7.3	
2	-1	0	1	
3	0.4	0.7	0.65	Local Best

F(X 2,3) is better than F(X 1,3) then the solution of X2,3 will be accepted for the next iteration. In iteration 1, all solutions will be accepted as a new solution for the next iteration, except, the solution for ant 2, because it is worse than the previous ant, the result of the ants is shown in Table 7:

– Iteration 2:

In the second iteration, Table 6, shows the previous ants and Table 7 shows the current ants.

For each ant generate new random r = 0.3 (supposed randomly for each ant):

Ant 1,1 = (-1.7, 2.1), which is representing the second iteration, the first ant, and it is neither local best nor equal to the previous solution depending on algorithm steps, then Eqs. (14) , (15), (13) , (10), respectively, will be used to generate ants step size and to generate ant 1 for next iteration, formula (9) will be used:

Tx1 = 8.75, Tprevx1 = 14.8, Dwx1 = 0.17
Δx1 = 0.17 *(0.4 + 1.7)) = 0.35 (step change for parameter x1)
Tx2 = 5.25, Tprevx2 = 10.05, Dwx2 = 0.15
Δx2 = 0.15 *(0.7 – 2.1) = -0.21 (step change for parameter x2)
ΔX 3,1 = (0.35, -0.21)
X3,1 = (-1.7, 2.1) + (0.35, -0.21) = (0.4, 1.89) (new solution)
F(X3,1) = 3.73 will accepted solution for next iteration

Ant 2, 2 = (-1,0)**,** which represents the second iteration, the second ant, and is equal to the previous solution depending on algorithm steps, then formula (12) will be used to generate ants step size and generate ant 1 for iteration 2 depending formula (9) for next iteration, suppose r = 0.3 (randomly):

$$\Delta X_{2,3} = \mathrm{r}\left(X_{1,3\mathrm{best}} - X_{1,3}\right)$$
$$\Delta X = 0.3*((0.4, 0.7) - (-1, 0)) = (0.42, 0.21)$$
$$X3, 2 = (-1, 0) + (0.42, 0.21) = (-0.58, 0.21)$$
$$F(\mathrm{X3}, 2) = 0.38 \text{ will accepted solution for next iteration.}$$

Table 8. Final population

Ant	X_1	X_2	F(x)	note
1	0.4	1.89	3.73	
2	0.58	0.21	0.38	Local Best
3	0.4	0.7	0.65	

Ant 2,3 = (0.4,0.7), which represents the second iteration, the third ant, and it is equal to the local best solution, then formula (11) will be used to generate step size which is used and generate ant 2 for iteration 2 using formula (9), r = 0.3 (randomly):

$$\Delta X_{1,2} = \mathrm{r} \times X_{1,2}$$
$$\Delta X = 0.3*(0.4, 0.7) = (0.12, 0.21)$$
$$X3, 3 = (0.4, 0.7) + (0.12, 0.21) = (0.52, 0.91)$$
$$F(\mathrm{X3}, 3) = 1.09 \text{ will not accepted solution for the next iteration.}$$

The result of worker ants is represented in Table 8 as the final population and the local best is the global best which ant 2 with positons values $X_1 = -0.58$, $X_{2=}$ 0.21 and f(x) = 0.38.

4 Conclusion

Metaheuristic algorithms are working on randomness, which gives a global search in the space search. Both MFO and ANA working on changing their position depending on the step size with the current position. Both algorithms have some randomness for changing positions but most part updating positions depends on a local best solution, which leads to fast convergence but low exploration. The results of implementation steps show that ANA is faster converge than MFO but it may stuck in the local solution if the initial solution is bad or complex problems are given because step size depends on the current position or some formulas for the local best solution when a new ant position is worse than the current ant, then ant will not change its current position, this is low exploration. The mplementation steps of MFO shows that when flame numbers decrease along the iteration, the rest moths will update their location depending on a single flame, this also may lead to the stuck algorithm in the local optima. In the algorithm of MFO flames of next iteration is generated from current and previous iteration moths while may some time global best is in old flame and the solution will lost during change then flame elements also effect with low explotation mechanism.

Limitation of the MFO low exploration and flame element decision during itrations, while ANA has fast convergence and low explorations depending on position change, and has problems with some complex problems.

For future work apply some mechanism to improve exploration for both MFO and ANA and compare the results again, then apply in different domain of problems like solving binary and discrete problems to show the performance of both algorithms.

References

1. Xin-She Yang ,Nature-Inspired Metaheuristic Algorithms, 2nd Edition; Publisher: Luniver Press; (2010).
2. M. Dorigo, Optimization, Learning and Natural Algorithms, PhD thesis, Politecnico di Milano, Italy (1992).
3. Hama Rashid, D.N.; Rashid, T.A.; Mirjalili, S. ANA: Ant Nesting Algorithm for Optimizing Real-World Problems. Mathematics 9, 3111, 2021.
4. Yang, XS, A New Metaheuristic Bat-Inspired Algorithm. In: González, J.R., Pelta, D.A., Cruz, C., Terrazas, G., Krasnogor, N. (eds) Nature Inspired Cooperative Strategies for Optimization (NICSO 2010). Studies in Computational Intelligence, vol 284. Springer, Berlin, Heidelberg (2010).
5. Dervis Karaboga, Artificial bee colony algorithm. Scholarpedia, 5(3):6915, (2010).
6. Kennedy, J.; Eberhart, R, "Particle Swarm Optimization". Proceedings of IEEE International Conference on Neural Networks. Vol. IV. pp. 1942–1948, . (1995).
7. Seyedali Mirjalili, Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm, Knowledge-Based Systems, Volume 89, Pages 228-249, (2015).
8. Y. Liu and B. Cao, "A Novel Ant Colony Optimization Algorithm With Levy Flight," in IEEE Access, vol. 8, pp. 67205-67213, 2020.
9. Liu, Y., Cao, B. & Li, H. Improving ant colony optimization algorithm with epsilon greedy and Levy flight. Complex Intell. Syst. 7, 1711–1722 (2021).
10. Shekhawat, S., Saxena, A., Kumar, R., Singh, V.P. Levy Flight Opposition Embedded BAT Algorithm for Model Order Reduction. In: Dey, N., Rajinikanth, V. (eds) Applications of Bat Algorithm and its Variants. Springer Tracts in Nature-Inspired Computing. Springer, Singapore, (2021).
11. Mridul Chawla & Manoj Duhan , Levy Flights in Metaheuristics Optimization Algorithms – A Review, Applied Artificial Intelligence, 32:9-10, 802-821 , (2018)
12. M. Yahya, M.P. Saka, Construction site layout planning using multi-objective artificial bee colony algorithm with Levy flights, Automation in Construction, Volume 38, Pages 14-29, (2014).
13. Wang, C., Shang, P. & Shen, P. An improved artificial bee colony algorithm based on Bayesian estimation. Complex Intell. Syst. (2022).
14. R. Jensi, G. Wiselin Jiji, An enhanced particle swarm optimization with levy flight for global optimization, Applied Soft Computing, Volume 43, Pages 248–261, (2016).
15. Hüseyin Haklı, Harun Uğuz, A novel particle swarm optimization algorithm with Levy flight, Applied Soft Computing, Volume 23, Pages 333-345, (2014).
16. bhishek Sharma, Abhinav Sharma, Moshe Averbukh, Shailendra Rajput, Vibhu Jately, Sushabhan Choudhury, Brian Azzopardi, Improved moth flame optimization algorithm based on opposition-based learning and Lévy flight distribution for parameter estimation of solar module, Energy Reports, Volume 8, Pages 6576–6592, (2022).
17. Oheil Mohseni, Alan C. Brent, Daniel Burmester, Will N. Browne, Lévy-flight moth-flame optimisation algorithm-based micro-grid equipment sizing: An integrated investment and operational planning approach, Energy and AI, Volume 3, 100047, (2021).
18. D. Jovanovic, M. Antonijevic, M. Stankovic, M. Zivkovic, M. Tanaskovic, N. Bacanin, Tuning Machine Learning Models Using a Group Search Firefly Algorithm for Credit Card Fraud Detection, Mathematics, Volume 10, No. 13, pp. 1 – 30, (2022)
19. Zivkovic, M. *et al.* , Novel Harris Hawks Optimization and Deep Neural Network Approach for Intrusion Detection , Proceedings of International Joint Conference on Advances in Computational Intelligence, Algorithms for Intelligent Systems, Springer, Singapore (2022)

20. Zivkovic, M. et al. Novel Chaotic Best Firefly Algorithm: COVID-19 Fake News Detection Application. Advances in Swarm Intelligence. Studies in Computational Intelligence, vol 1054. Springer, Cham. (2023).
21. Prakash, S. et al. Hybrid GLFIL Enhancement and Encoder Animal Migration Classification for Breast Cancer Detection, COMPUTER SYSTEMS SCIENCE AND ENGINEERING, Vol. 41, No. 2, pp. 735 - 749, 2022
22. Salb, M. et al. Training Logistic Regression Model by Enhanced Moth Flame Optimizer for Spam Email Classification. In: Smys, S., Lafata, P., Palanisamy, R., Kamel, K.A. (eds) Computer Networks and Inventive Communication Technologies. Lecture Notes on Data Engineering and Communications Technologies, vol 141. Springer, Singapore, (2023).
23. Bacanin, N. *et al.* A Novel Multiswarm Firefly Algorithm: An Application for Plant Classification. In: Kahraman, C., Tolga, A.C., Cevik Onar, S., Cebi, S., Oztaysi, B., Sari, I.U. (eds) Intelligent and Fuzzy Systems. INFUS 2022. Lecture Notes in Networks and Systems, vol 504. Springer, Cham. (2022).
24. Budimirovic , N. et al. COVID-19 Severity Prediction Using Enhanced Whale with Salp Swarm Feature Classification, CMC-Computers, Materials & Continua, Vol. 72, No. 1, p. 1685 - 1698, Feb, (2022).
25. Bacanin, N. et al. Training a Multilayer Perception for Modeling Stock Price Index Predictions Using Modified Whale Optimization Algorithm. In: Smys, S., Tavares, J.M.R.S., Balas, V.E. (eds) Computational Vision and Bio-Inspired Computing. Advances in Intelligent Systems and Computing, vol 1420. Springer, Singapore.
26. Bacanin, N., Zivkovic, M., Bezdan, T. *et al.* Modified firefly algorithm for workflow scheduling in cloud-edge environment. *Neural Comput & Applic* 34, 9043–9068 (2022).
27. Bacanin, N., Antonijevic, M., Bezdan, T. *et al.* Energy efficient offloading mechanism using particle swarm optimization in 5G enabled edge nodes. *Cluster Comput* (2022).
28. Bacanin, N., Zivkovic, M., Al-Turjman, F. *et al.* Hybridized sine cosine algorithm with convolutional neural networks dropout regularization application. *Sci Rep* 12, 6302 (2022).