Knowledge-Based Systems 89 (2015) 228–249

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm

Seyedali Mirjalili *

School of Information and Communication Technology, Griffith University, Nathan Campus, Brisbane, QLD 4111, Australia
Queensland Institute of Business and Technology, Mt Gravatt, Brisbane, QLD 4122, Australia

ARTICLE INFO

Article history:
Received 23 May 2015
Received in revised form 25 June 2015
Accepted 11 July 2015
Available online 18 July 2015

Keywords:
Optimization
Stochastic optimization
Constrained optimization
Meta-heuristic
Population-based algorithm

ABSTRACT

In this paper a novel nature-inspired optimization paradigm is proposed called Moth-Flame Optimization (MFO) algorithm. The main inspiration of this optimizer is the navigation method of moths in nature called transverse orientation. Moths fly in night by maintaining a fixed angle with respect to the moon, a very effective mechanism for travelling in a straight line for long distances. However, these fancy insects are trapped in a useless/deadly spiral path around artificial lights. This paper mathematically models this behaviour to perform optimization. The MFO algorithm is compared with other well-known nature-inspired algorithms on 29 benchmark and 7 real engineering problems. The statistical results on the benchmark functions show that this algorithm is able to provide very promising and competitive results. Additionally, the results of the real problems demonstrate the merits of this algorithm in solving challenging problems with constrained and unknown search spaces. The paper also considers the application of the proposed algorithm in the field of marine propeller design to further investigate its effectiveness in practice. Note that the source codes of the MFO algorithm are publicly available at http://www.alimirjalili.com/MFO.html.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

Optimization refers to the process of finding the best possible solution(s) for a particular problem. As the complexity of problems increases, over the last few decades, the need for new optimization techniques becomes evident more than before. Mathematical optimization techniques used to be the only tools for optimizing problems before the proposal of heuristic optimization techniques. Mathematical optimization methods are mostly deterministic that suffer from one major problem: local optima entrapment. Some of them such as gradient-based algorithms require derivation of the search space as well. This makes them highly inefficient in solving real problems.

The so-called Genetic Algorithm (GA) [1], which is undoubtedly the most popular stochastic optimization algorithm, was proposed to alleviate the aforementioned drawbacks of the deterministic algorithms. The key success of GA algorithm mostly relies on the stochastic components of this algorithm. The selection, re-production, and mutation have all stochastic behaviours that assist GA to avoid local optima much more efficient than mathematical optimization algorithms. Since the probability of selection and re-production of best individuals is higher than worst individuals, the overall average fitness of population is improved over the course of generations. These two simple concepts are the key reasons of the success of GA in solving optimization problems. Another fact about this algorithm is that there is no need to have gradient information of the problems since GA only evaluates the individuals based on the fitness. This makes this algorithm highly suitable for solving real problems with unknown search spaces. Nowadays, the application of the GA algorithm can be found in a wide range of fields.

The years after the proposal of the GA were accompanied by the highest attention to such algorithms, which resulted in the proposal of the well-known algorithms like PSO [2] algorithm, Ant Colony Optimization (ACO) [3], Differential Evolution (DE) [4], Evolutionary Strategy (ES) [5], and Evolutionary Programming (EP) [6,7]. The application of these algorithms can be found in different branches of science and industry as well. Despite the merits of these optimizers, there is a fundamental question here as if there is any optimizer for solving all optimization problems. According to the No-Free-Lunch (NFL) theorem [8], there is no algorithm for solving all optimization problems. This means that an optimizer may perform well in a set of problems and fail to solve a different set of problems. In other words, the average performance of optimizes is equal when considering all optimization problems. Therefore, there are still problems that can be solved by new optimizers better than the current optimizers.

* Address: School of Information and Communication Technology, Griffith University, Nathan Campus, Brisbane, QLD 4111, Australia.
E-mail address: seyedali.mirjalili@griffithuni.edu.au

http://dx.doi.org/10.1016/j.knosys.2015.07.006
0950-7051/© 2015 Elsevier B.V. All rights reserved.

This is the motivation of this work, in which a novel nature-inspired algorithm is proposed to compete with the current optimization algorithms. The main inspiration of the proposed algorithm is the navigating mechanism of moths in nature called transverse orientation. The paper first proposes the mathematical model of spiral flying path of moths around artificial lights (flames). An optimization algorithm is then proposed using the mathematical model to solve optimization problems in different fields. The rest of the paper is organized as follows.

Section 2 reviews the literature of stochastic and heuristic optimization algorithms. Section 3 presents the inspiration of this work and proposes the MFO algorithm. The experimental setup, results, discussion, and analysis are provided in Section 4. Section 5 investigates the effectiveness of the proposed MFO algorithm in solving nine constrained engineering design problems: welded beam, gear train, three-bar truss, pressure vessel, cantilever beam, I-beam, tension/compression spring, 15-bar truss, and 52-bar truss design problems. In addition, Section 6 demonstrates the application of MFO in the field of marine propeller design. Eventually, Section 7 concludes the work and suggests several directions for future studies.

2. Literature review

This section first reviews the state-of-the-art and then discusses the motivations of this work.

A brief history of stochastic optimization techniques was provided in the introduction. A general classification of the algorithms in this field is based on the number of candidate solutions that is improved during optimization. An algorithm may start and perform the optimization process by single or multiple random solution(s). In the former case the optimization process begins with a single random solution, and it is iteratively improved over the iterations. In the latter case a set of solutions (more than one) is created and improved during optimization. These two families are called individual-based and population-based algorithms.

There are several advantages and disadvantages for each of these families. Individual-based algorithms need less computational cost and function evaluation but suffer from premature convergence. Premature convergence refers to the stagnation of an optimization technique in local optima, which prevents it from convergence towards the global optimum. In contrary, population-based algorithms have high ability to avoid local optima since a set of solutions are involved during optimization. In addition, information can be exchanged between the candidate solutions, which assist them to overcome different difficulties of search spaces. However, high computational cost and the need for more function evaluation are two major drawbacks of population-based algorithms.

The well-known algorithms in the individual-based family are: Tabu Search (TS) [6,9], hill climbing [10], Iterated Local Search (ILS) [11], and Simulated Annealing (SA) [12]. TS is an improved local search technique that utilizes short-term, intermediate-term, and long-term memories to ban and truncate unpromising/repeated solutions. Hill climbing is also another local search and individual-based technique that starts optimization by a single solution. This algorithm then iteratively attempts to improve the solution by changing its variables. ILS is an improved hill climbing algorithm to decrease the probability of trapping in local optima. In this algorithm, the obtained optimum at the end of each run is perturbed and considered as the starting point in the next iteration. Eventually, the SA algorithm tends to accept worse solutions proportional to a variable called cooling factor. This assists SA to promote exploration of the search space and consequently avoid local optima.

Although different improvements of individual-based algorithms promote local optima avoidance, the literature shows that population-based algorithms are better in handling this issue. Regardless of the differences between population-based algorithms, the common is the division of optimization process to two conflicting milestones: exploration versus exploitation [13]. The exploration milestone encourages candidate solutions to change abruptly and stochastically. This mechanism improves the diversity of the solutions and causes high exploration of the search space. In PSO, for instance, the inertia weight maintains the tendency of particles toward their previous directions and emphasizes exploration. In GA, high probability of crossover causes more combination of individuals and is the main mechanism for the exploration milestone.

In contrast, the exploitation milestone aims for improving the quality of solutions by searching locally around the obtained promising solutions in the exploration milestone. In this milestone, candidate solutions are obliged to change less suddenly and search locally. In PSO, for instance, low inertia rate causes low exploration and high tendency toward to best personal/global solutions obtained. Therefore, the particles converge toward best points instead of churning around the search space. The mechanism that brings GA exploitation is the mutation operators. Mutation causes slight random changes in the individuals and local search around the candidate solutions.

Exploration and exploitation are two conflicting milestones where promoting one results in degrading the other [14]. A right balance between these two milestones can guarantee a very accurate approximation of the global optimum using population-based algorithms. On one hand, mere exploration of the search space prevents an algorithm from finding an accurate approximation of the global optimum. On the other hand, mere exploitation results in local optima stagnation and again low quality of the approximated optimum. Due to the unknown shape of the search space for optimization problems, in addition, there is no clear accurate timing for transition between these two milestones. Therefore, population-based algorithms balance exploration and exploitation milestones to firstly find a rough approximation of the global optimum, and then improve its accuracy.

The general framework of population-based algorithms is almost identical. The first step is to generate a set of random initial solutions $(\overrightarrow{X}) = \{\overrightarrow{X_1}, \overrightarrow{X_2}, \ldots, \overrightarrow{X_n}\}$. Each of these solutions is considered as a candidate solution for a given problem, assessed by the objective function, and assigned an objective value: $(\overrightarrow{O}) = \{O_1, O_2, \ldots, O_n\}$. The algorithm then combines/moves/updates the candidate solutions based on their fitness values with the hope to improve them. The created solutions are again assessed by the objective function and assigned their relevant fitness values. This process is iterated until the satisfaction of an end condition. At the end of this process, the best solution obtained is reported as the best approximation for the global optimum.

Recently, many population-based algorithms have been proposed. They can be classified to three main categories based on the source of inspiration: evolution, physic, or swarm. Evolutionary algorithms are those who mimic the evolutionary processes in nature. Some of the recently proposed evolutionary algorithms are Biogeography-based Optimization (BBO) algorithm [15], evolutionary membrane algorithm [16], human evolutionary model [17], and Asexual Reproduction Optimization (ARO) [18].

The number of recently proposed swarm-based algorithms is larger than evolutionary algorithms. Some of the most recent ones are Glowworm Swarm Optimization (GSO) [19], Bees Algorithm (BA) [20], Artificial Bee Colony (ABC) algorithm [21], Bat Algorithm (BA) [22], Firefly Algorithm (FA) [23], Cuckoo Search (CS) algorithm [24], Cuckoo Optimization Algorithm (COA) [25], Grey Wolf Opti-

mizer (GWO) [26]. Dolphin Echolocation (DE) [27], Hunting Search (HS) [28], and Fruit Fly Optimization Algorithm (FFOA) [29].

The third class of algorithms is inspired from physical phenomena in nature. The most recent algorithms in this category are: Gravitational Search Algorithm (GSA) [30], Chemical Reaction Optimization (CRO) [31], Artificial Chemical Reaction Optimization Algorithm (ACROA) [32], Charged System Search (CSS) algorithm [33], Ray Optimization (RO) [34], Black Hole (BH) algorithm [35], Central Force Optimization (CFO) [36], Kinetic Gas Molecules Optimization algorithm (KGMO) [37], and Gases Brownian Motion Optimization (GBMO) [38].

In addition the above-mentioned algorithms, there are also other population-based algorithms with different source of inspirations. The most recent ones are: Harmony Search (HS) optimization algorithm [39], Mine Blast Algorithm (MBA) [40], Symbiotic Organisms Search (SOS) [41], Soccer League Competition (SLC) algorithm [42], Seeker Optimization Algorithm (SOA) [43], Coral Reef Optimization (CRO) algorithm [44], Flower Pollination Algorithm (FPA) [45], and State of Mater Search (SMS) [46].

As the above paragraphs shows, there are many algorithms in this field, which indicates the popularity of these techniques in the literature. If we consider the hybrid, multi-objective, discrete, and constrained methods, the number of publications will be increased dramatically. The reputation of these algorithms is due to several reasons. Firstly, simplicity is the main advantage of the population-based algorithm. The majority of algorithms in this field follows a simple framework and have been inspired from simple concepts. Secondly, these algorithms consider problems as black boxes, so they do not need derivative information of the search space in contrast to mathematical optimization algorithms. Thirdly, the local optima avoidance of population-based stochastic optimization algorithms is very high, making them suitable for practical applications. Lastly, population-based algorithms are highly flexible, meaning that they are readily applicable for solving different optimization problems without structural modifications. In fact, the problem representation becomes more important than the optimizer when using population-based algorithms.

Despite the high number of new algorithms and their applications in science and industry, there is a question here that if we need more algorithms in this field. The answer to this questions is positive according to the No-Free-Lunch (NFL) [8] theorem for optimization. This theorem logically proves that there is no optimization algorithm for solving all optimization problems. This means that an algorithm can be useful for a set of problems but useless of other types of problems. In other words, the algorithms perform similar in average over all the possible optimization problems. This theorem allows the proposal of new algorithms with the hope to solve a wider range of problems or specific types of unsolved problems. This is also the motivation of this study where it is tried to get inspiration from the navigation of moths in nature and design an optimization algorithm.

3. Moth-flame optimiser

3.1. Inspiration

Moths are fancy insects, which are highly similar to the family of butterflies. Basically, there are over 160,000 various species of this insect in nature. They have two main milestones in their lifetime: larvae and adult. The larvae is converted to moth in cocoons.

The most interesting fact about moths is their special navigation methods in night. They have been evolved to fly in night using the moon light. They utilized a mechanism called transverse orientation for navigation. In this method, a moth flies by maintaining a fixed angle with respect to the moon, a very effective mechanism for travelling long distances in a straight path [47,48]. Fig. 1 shows a conceptual model of transverse orientation. Since the moon is far away from the moth, this mechanism guarantees flying in straight line. The same navigation method can be done by humans. Suppose that the moon is in the south side of the sky and a human wants to go the east. If he keeps moon of his left side when walking, he would be able to move towards the east on a straight line.

Despite the effectiveness of transverse orientation, we usually observe that moths fly spirally around the lights. In fact, moths are tricked by artificial lights and show such behaviours. This is due to the inefficiency of the transverse orientation, in which it is only helpful for moving in straight line when the light source is very far. When moths see a human-made artificial light, they try to maintain a similar angle with the light to fly in straight line.

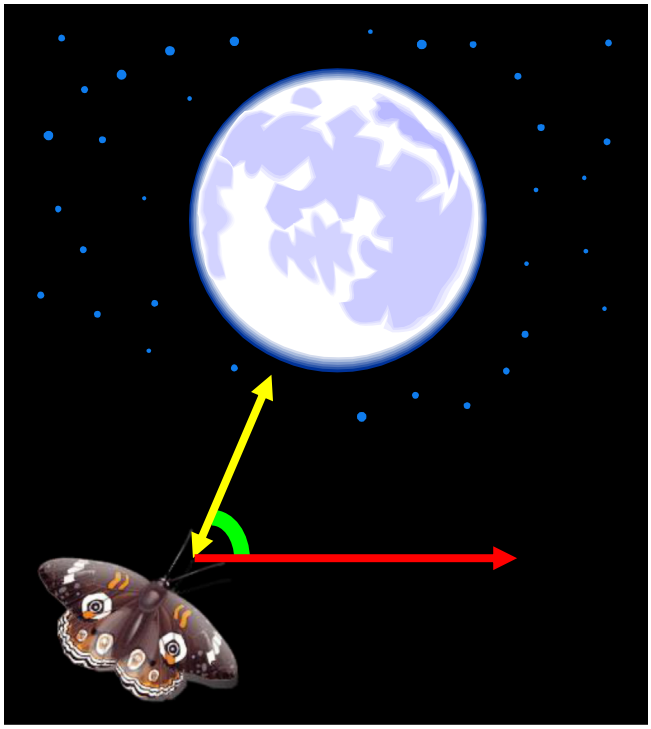

Fig. 1. Transverse orientation.

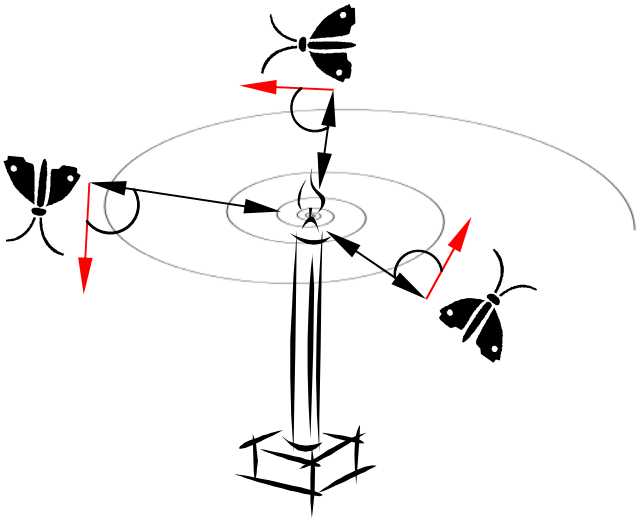

Fig. 2. Spiral flying path around close light sources.

Since such a light is extremely close compared to the moon, however, maintaining a similar angle to the light source causes a useless or deadly spiral fly path for moths [48]. A conceptual model of this behaviour is illustrated in Fig. 2.

It may be observed in Fig. 2 that the moth eventually converges towards the light. This behaviour is modeled mathematically to propose an optimizer called Moth-Flame Optimization (MFO) algorithm in the following subsection.

3.2. MFO algorithm

In the proposed MFO algorithm, it is assumed that the candidate solutions are moths and the problem's variables are the position of moths in the space. Therefore, the moths can fly in 1-D, 2-D, 3-D, or hyper dimensional space with changing their position vectors. Since the MFO algorithm is a population-based algorithm, the set of moths is represented in a matrix as follows:

$$M = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & \cdots & m_{1,d} \\ m_{2,1} & m_{2,2} & \cdots & \cdots & m_{2,d} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m_{n,1} & m_{n,2} & \cdots & \cdots & m_{n,d} \end{bmatrix} \tag{3.1}$$

where n is the number of moths and d is the number of variables (dimension).

For all the moths, we also assume that there is an array for storing the corresponding fitness values as follows:

$$OM = \begin{bmatrix} OM_1 \\ OM_2 \\ \vdots \\ OM_n \end{bmatrix} \tag{3.2}$$

where n is the number of moths.

Note that the fitness value is the return value of the fitness (objective) function for each moth. The position vector (first row in the matrix M for instance) of each moth is passed to the fitness function and the output of the fitness function is assigned to the corresponding moth as its fitness value (OM_1 in the matrix OM for instance).

Another key components in the proposed algorithm are flames. A matrix similar to the moth matrix is considered as follows:

$$F = \begin{bmatrix} F_{1,1} & F_{1,2} & \cdots & \cdots & F_{1,d} \\ F_{2,1} & F_{2,2} & \cdots & \cdots & F_{2,d} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ F_{n,1} & m_{n,2} & \cdots & \cdots & F_{n,d} \end{bmatrix} \tag{3.3}$$

where n is the number of moths and d is the number of variables (dimension).

It may be seen in Eq. (3.3) that the dimensions of M and F arrays are equal. For the flames, it is also assumed that there is an array for storing the corresponding fitness values as follows:

$$OF = \begin{bmatrix} OF_1 \\ OF_2 \\ \vdots \\ OF_n \end{bmatrix} \tag{3.4}$$

where n is the number of moths.

It should be noted here that moths and flames are both solutions. The difference between them is the way we treat and update them in each iteration. The moths are actual search agents that move around the search space, whereas flames are the best position of moths that obtains so far. In other words, flames can be considered as flags or pins that are dropped by moths when searching the search space. Therefore, each moth searches around a flag (flame) and updates it in case of finding a better solution. With this mechanism, a moth never loses its best solution.

The MFO algorithm is a three-tuple that approximates the global optimal of the optimization problems and defined as follows:

$$MFO = (I, P, T) \tag{3.5}$$

I is a function that generates a random population of moths and corresponding fitness values. The methodical model of this function is as follows:

$$I : \emptyset \rightarrow \{M, OM\} \tag{3.6}$$

The P function, which is the main function, moves the moths around the search space. This function received the matrix of M and returns its updated one eventually.

$$P : M \rightarrow M \tag{3.7}$$

The T function returns true if the termination criterion is satisfied and false if the termination criterion is not satisfied:

$$T : M \rightarrow \{true, false\} \tag{3.8}$$

With I, P, and T, the general framework of the MFO algorithm is defined as follows:

```
M = I();
while T(M) is equal to false
  M = P(M);
end
```

The function I has to generate initial solutions and calculate the objective function values. Any random distribution can be used in this function. The following method is utilized as the default:

```
for i = 1: n
  for j = 1: d
    M(i,j) = (ub(i) − lb(i)) ∗ rand() + lb(i);
  end
end
OM = FitnessFunction(M);
```

As can be seen, there are two other arrays called ub and lb. These matrixes define the upper and lower bounds of the variables as follows:

$$ub = [ub_1, ub_2, ub_3, \ldots, ub_{n-1}, ub_n] \tag{3.9}$$

where ub_i indicates the upper bound of the i-th variable.

$$lb = [lb_1, lb_2, lb_3, \ldots, lb_{n-1}, lb_n] \tag{3.10}$$

where lb_i indicates the lower bound of the i-th variable.

After the initialization, the P function is iteratively run until the T function returns true. The P function is the main function that moves the moths around the search space. As mentioned above the inspiration of this algorithm is the transverse orientation. In order to mathematically model this behaviour, the position of each moth is updated with respect to a flame using the following equation:

$$M_i = S(M_i, F_j) \tag{3.11}$$

where M_i indicate the i-th moth, F_j indicates the j-th flame, and S is the spiral function.

A logarithmic spiral is chosen as the main update mechanism of moths in this paper. However, any types of spiral can be utilized here subject to the following conditions:

- Spiral's initial point should start from the moth.
- Spiral's final point should be the position of the flame.
- Fluctuation of the range of spiral should not exceed from the search space.

Considering these points, a logarithmic spiral is defined for the MFO algorithm as follows:

$$S(M_i, F_j) = D_i \cdot e^{bt} \cdot \cos(2\pi t) + F_j \tag{3.12}$$

where D_i indicates the distance of the i-th moth for the j-th flame, b is a constant for defining the shape of the logarithmic spiral, and t is a random number in $[-1, 1]$.

D is calculated as follows:

$$D_i = |F_j - M_i| \tag{3.13}$$

where M_i indicate the i-th moth, F_j indicates the j-th flame, and D_i indicates the distance of the i-th moth for the j-th flame.

Eq. (3.12) is where the spiral flying path of moths is simulated. As may be seen in this equation, the next position of a moth is defined with respect to a flame. The t parameter in the spiral equation defines how much the next position of the moth should be close to the flame ($t = -1$ is the closest position to the flame, while $t = 1$ shows the farthest). Therefore, a hyper ellipse can be assumed around the flame in all directions and the next position of the moth would be within this space. The spiral movement is the main component of the proposed method because it dictates how the moths update their positions around flames. The spiral equation allows a moth to fly "around" a flame and not necessarily in the space between them. Therefore, the exploration and exploitation of the search space can be guaranteed. The logarithmic spiral, space around the flame, and the position considering different t on the curve are illustrated in Fig. 3.

Fig. 4 shows a conceptual model of position updating of a moth around a flame. Note that the vertical axis shows only one dimension (1 variable/parameter of a given problem), but the proposed method can be utilized for changing all the variables of the problem. The possible positions (dashed black lines) that can be chosen as the next position of the moth (blue horizontal line) around the flame (green horizontal line) in Fig. 4[1] clearly show that a moth can explore and exploit the search space around the flame in one dimension. Exploration occurs when the next position is outside the space between the moth and flam as can be seen in the arrows labelled by 1, 3, and 4. Exploitation happens when the next position lies inside the space between the moth and flame as can be observed in the arrow labelled by 2. There are some interesting observations for this model as follow:

- A moth can converge to any point in the neighbourhood of the flame by changing t.
- The lower t, the closer distance to the flame.
- The frequency of position updating on both sides of the flame is increased as the moth get closer to the flame.

The proposed position updating procedure can guarantee the exploitation around the flames. In order to improve the probability of finding better solutions, the best solutions obtained so far are considered as the flames. So, the matrix F in Eq. (3.3) always includes n recent best solutions obtained so far. The moths are required to update their positions with respect to this matrix

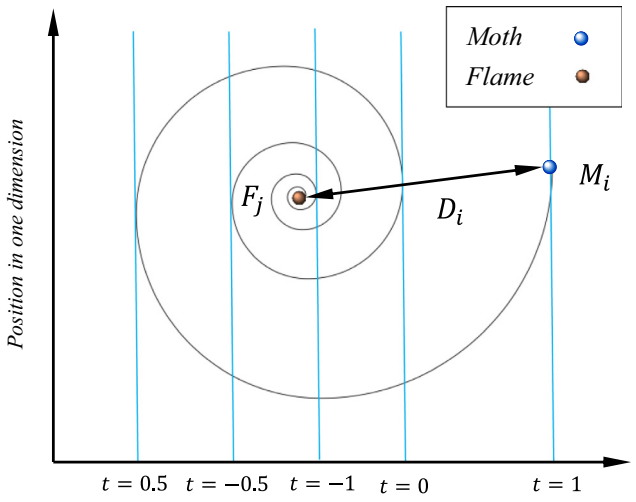

Fig. 3. Logarithmic spiral, space around a flame, and the position with respect to t.

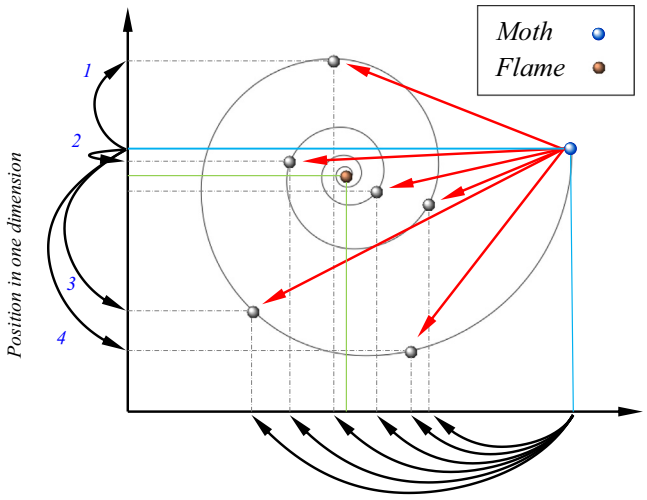

Fig. 4. Some of the possible positions that can be reached by a moth with respect to a flame using the logarithmic spiral.

during optimization. In order to further emphasize exploitation, it is assumed that t is a random number in $[r, 1]$ where r is linearly decreased from -1 to -2 over the course of iteration. Note that r is named as the convergence constant. With this method, moths tend to exploit their corresponding flames more accurately proportional to the number of iterations.

A question that may rise here is that the position updating in Eq. (3.12) only requires the moths to move towards a flame, yet it causes the MFO algorithm to be trapped in local optima quickly. In order to prevent this, each moth is obliged to update its position using only one of the flames in Eq. (3.12). It each iteration and after updating the list of flames, the flames are sorted based on their fitness values. The moths then update their positions with respect to their corresponding flames. The first moth always updates its position with respect to the best flame, whereas the last moth updates its position with respect to the worst flame in the list. Fig. 5 shows how each moth is assigned to a flame in the list of flames.

It should be noted that this assumption is done for designing the MFO algorithm, while possibly it is not the actual behaviour of moths in nature. However, the transverse orientation is still done by the artificial moths. The reason that why a specific flame is assigned to each moth is to prevent local optimum stagnation. If all of the moths get attracted to a single flame, all of them

[1] For interpretation of color in 'Fig. 4', the reader is referred to the web version of this article.

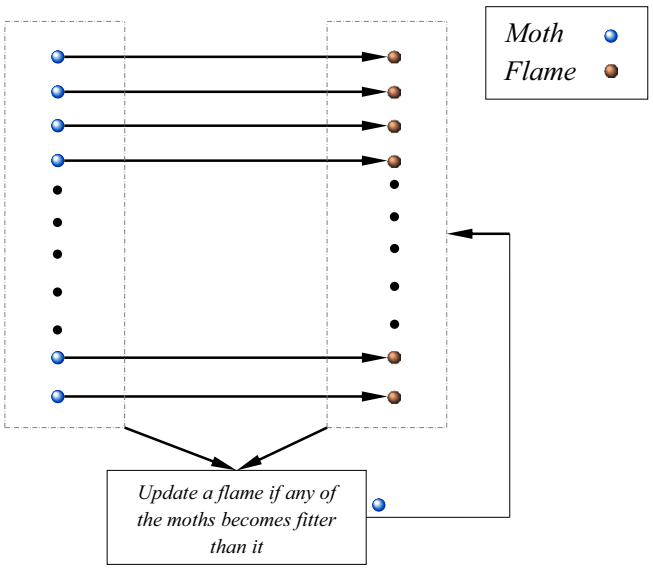

Fig. 5. Each moth is assigned to a flame.

converge to a point in the search spaces because they can only fly towards a flame and not outwards. Requiring them to move around different flames, however, causes higher exploration of the search space and lower probability of local optima stagnation.

Therefore, the exploration of the search space around the best locations obtained so far is guaranteed with this method due to the following reasons:

- Moths update their positions in hyper spheres around the best solutions obtained so far.
- The sequence of flames is changed based on the best solutions in each iteration, and the moths are required to update their positions with respect to the updated flames. Therefore, the position updating of moths may occur around different flames, a mechanism that causes sudden movement of moths in the search space and promotes exploration.

Another concern here is that the position updating of moths with respect to n different locations in the search space may degrade the exploitation of the best promising solutions. To resolve this concern, an adaptive mechanism is proposed for the number of flames. Fig. 6 shows that how the number of flames is decreased adaptively over the course of iterations. The following formula is utilized in this regard:

$$flame\ no = round\left(N - l * \frac{N-1}{T}\right) \tag{3.14}$$

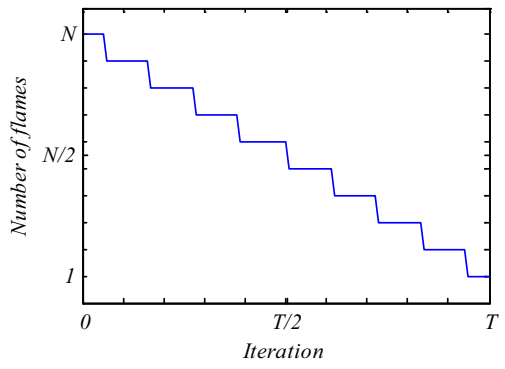

Fig. 6. Number of flame is decreased adaptively over the course of iterations.

where l is the current number of iteration, N is the maximum number of flames, and T indicates the maximum number of iterations.

Fig. 6 shows that there is N number of flames in the initial steps of iterations. However, the moths update their positions only with respect to the best flame in the final steps of iterations. The gradual decrement in number of flames balances exploration and exploitation of the search space. After all, the general steps of the P function are as follows.

```
Update flame no using Eq. (3.14)
OM = FitnessFunction(M);
if iteration == 1
    F = sort(M);
    OF = sort(OM);
else
    F = sort(M_{t-1}, M_t);
    OF = sort(M_{t-1}, M_t);
end
for i = 1: n
    for j = 1: d
        Update r and t
        Calculate D using Eq. (3.13) with respect to the
  corresponding moth
        Update M(i,j) using Eqs. (3.11) and (3.12) with respect to
  the corresponding moth
    end
end
```

As discussed above, the P function is executed until the T function returns true. After termination the P function, the best moth is returned as the best obtained approximation of the optimum.

3.3. Computational complexity of the MFO algorithm

Computation complexity of an algorithm is a key metric for evaluating its run time, which can be defined based on the structure and implementation of the algorithm. The computational complexity of the MFO algorithm depends on the number of moths, number of variables, maximum number of iterations, and sorting mechanism of flames in each iteration. Since the Quicksort algorithm is utilized, the sort's computational complexity is of $O(nlogn)$ and $O(n^2)$ in the best and worst case, respectively. Considering the P function, therefore, the overall computational complexity is defined as follows:

$$O(MFO) = O(t(O(Quick\ sort) + O(position\ update))) \tag{3.15}$$

$$O(MFO) = O(t(n^2 + n \times d)) = O(tn^2 + tnd) \tag{3.16}$$

where n is the number of moths, t is the maximum number of iterations, and d is the number of variables.

To see how the MFO algorithm can theoretically be effective in solving optimization problems some observations are:

- Procedure of updating positions allows obtaining neighbouring solutions around the flames, a mechanism for mostly promoting exploitation.
- Since MFO utilizes a population of moths, local optima avoidance is high.
- Assigning each moth a flame and updating the sequence of flames in each iteration increase exploration of the search space and decreases the probability of local optima stagnation.
- Considering recent best solutions obtained so far as the flames saves the promising solutions as the guides for moths.

- The best solutions are saved in the F matrix so they never get lost.
- Adaptive number of flames balances exploration and exploitation.
- Adaptive convergence constant (r) causes accelerated convergence around the flames over the course of iterations.

These observations make the MFO algorithm potentially able to improve the initial random solutions and convergence to a better point in the search space. The next section investigates the effectiveness of MFO in practice.

4. Results and discussion

It is a common in this field to benchmark the performance of algorithms on a set of mathematical functions with known global optima. The same process is followed, in which 19 benchmark functions are employed from the literature as test beds for comparison [7,49–51]. The test functions are divided to three groups: unimodal, multi-modal, and composite. The unimodal functions ($F1$—$F7$) are suitable for benchmarking the exploitation of algorithms since they have one global optimum and no local optima. In contrary, multi-modal functions ($F8$—$F13$) have a massive number of local optima and are helpful to examine exploration and local optima avoidance of algorithms. Eventually, composite functions ($F14$—$F19$) are the combination of different rotated, shifted, and biased multi-modal test functions. Since the search space of these functions is very challenging, as illustrated in Fig. 7, they are highly similar to the real search spaces and useful for benchmarking the performance of algorithms in terms of balanced exploration and exploitation.

The mathematical formulation of the employed test functions are presented in Tables 1–3. Since the original version of unimodal and multi-modal test functions are too simple, they are shifted to increase the difficulty of these functions. The shifted positions of global optima are provided in the Tables 1 and 2 as well. Hundred variables are also considered for unimodal and multi-modal test functions for further improving their difficulties. Note that the composite test functions are taken from CEC 2005 special session [52,53].

Since heuristic algorithms are stochastic optimization techniques, they have to be run at least more than 10 times for generating meaningful statistical results. It is again a common that an algorithm is run on a problem m times and average/standard deviation/median of the best obtained solution in the last iteration are calculated as the metrics of performance. The same method is selected to generate and report the results over 30 independent runs. However, average and standard deviation only compare the overall performance of algorithms. In addition to average and standard deviation, statistical tests should be done to confirm the significance of the results based on every single runs [54]. With the statistical test, we can make sure that the results are not generated by chance. The non-parametric Wilcoxon statistical test is conducted and the calculated p-values are reported as metrics of significance as well.

In order to verify the performance of the proposed MFO algorithm, some of the well-known and recent algorithms in the literature are chosen: PSO [55], GSA [30], BA [22], FPA [45], SMS [46], FA [23], and GA [56]. Note that 30 number search agents and 1000 iterations are utilized for each of the algorithms. It should be noted that selection of the number of moths (or other candidate solutions in other algorithms) should be done experimentally. The larger the number of artificial moths, the higher probability of determining the global optimum. However, it is observed that 30 is a reasonable number of moths for solving optimization problems. For expensive problems, this number can be reduced to 20 or 10.

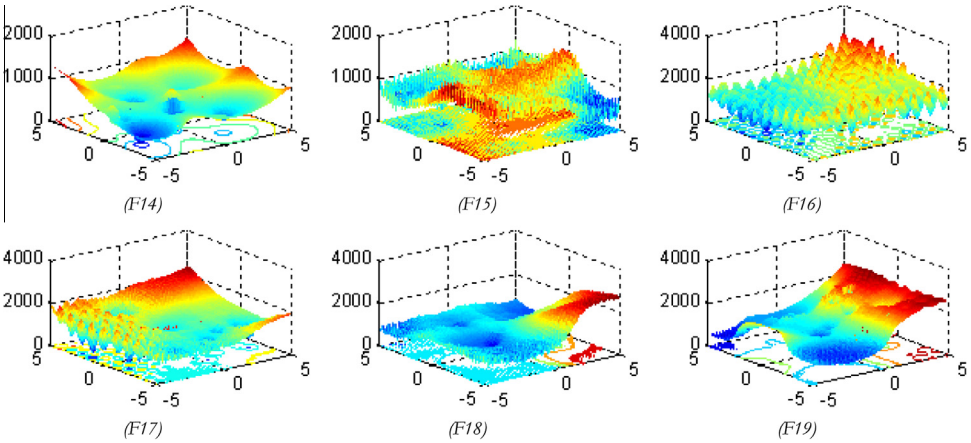

Fig. 7. Search space of composite benchmark functions.

Table 1
Unimodal benchmark functions.

Function	Dim	Range	Shift position	f_{min}
$f_1(x)=\sum_{i=1}^{n}x_i^2$	100	[−100, 100]	[−30, −30, ..., −30]	0
$f_2(x)=\sum_{i=1}^{n}\lvert x_i\rvert+\prod_{i=1}^{n}\lvert x_i\rvert$	100	[−10, 10]	[−3, −3, ..., −3]	0
$f_3(x)=\sum_{i=1}^{n}\left(\sum_{j-1}^{i}x_j\right)^2$	100	[−100, 100]	[−30, −30, ..., −30]	0
$f_4(x)=\max_i\{\lvert x_i\rvert, 1\leqslant i\leqslant n\}$	100	[−100, 100]	[−30, −30, ..., −30]	0
$f_5(x)=\sum_{i=1}^{n-1}[100(x_{i+1}-x_i^2)^2+(x_i-1)^2]$	100	[−30, 30]	[−15, −15, ..., −15]	0
$f_6(x)=\sum_{i=1}^{n}([x_i+0.5])^2$	100	[−100, 100]	[−750, ..., −750]	0
$f_7(x)=\sum_{i=1}^{n}ix_i^4+random[0,1)$	100	[−1.28, 1.28]	[−0.25, ..., −0.25]	0

Table 2
Multimodal benchmark functions.

Function	Dim	Range	Shift position	f_{min}
$F_8(x)=\sum_{i=1}^{n}-x_i sin\left(\sqrt{\lvert x_i\rvert}\right)$	100	[−500, 500]	[−300, . . ., −300]	-418.9829×5
$F_9(x)=\sum_{i=1}^{n}[x_i^2-10cos(2\pi x_i)+10]$	100	[−5.12, 5.12]	[−2, −2, . . ., −2]	0
$F_{10}(x)=-20exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right)-exp\left(\frac{1}{n}\sum_{i=1}^{n}cos(2\pi x_i)\right)+20+e$	100	[−32, 32]		0
$F_{11}(x)=\frac{1}{4000}\sum_{i=1}^{n}x_i^2-\prod_{i=1}^{n}cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	100	[−600, 600]	[−400, . . ., −400]	0
$F_{12}(x)=\frac{\pi}{n}\left\{10sin(\pi y_1)+\sum_{i=1}^{n-1}(y_i-1)^2[1+10sin^2(\pi y_{i+1})]+(y_n-1)^2\right\}+\sum_{i=1}^{n}u(x_i,10,100,4)$ $y_i=1+\frac{x_i+1}{4}$ $u(x_i,a,k,m)=\begin{cases}k(x_i-a)^m & x_i>a\\ 0 & -a<x_i<a\\ k(-x_i-a)^m & x_i<-a\end{cases}$	100	[−50, 50]	[−30, −30, . . ., −30]	0
$F_{13}(x)=0.1\left\{sin^2(3\pi x_1)+\sum_{i=1}^{n}(x_i-1)^2[1+sin^2(3\pi x_i+1)]+(x_n-1)^2[1+sin^2(2\pi x_n)]\right\}$ $+\sum_{i=1}^{n}u(x_i,5,100,4)$	100	[−50, 50]	[−100, . . ., −100]	0

Table 3
Composite benchmark functions.

Function	Dim	Range	f_{min}
F_{14} (CF1): $f_1, f_2, f_3, \ldots, f_{10}$ = Sphere Function $[\sigma_1,\sigma_2,\sigma_3,\ldots,\sigma_{10}]=[1,1,1,\ldots,1]$ $[\lambda_1,\lambda_2,\lambda_3\ldots,\lambda_{10}]=[5/100,5/100,5/100,\ldots,5/100]$	30	[−5, 5]	0
F_{15} (CF2): $f_1, f_2, f_3, \ldots, f_{10}$ = *Griewank's Function* $[\sigma_1,\sigma_2,\sigma_3,\ldots,\sigma_{10}]=[1,1,1,\ldots,1]$ $[\lambda_1,\lambda_2,\lambda_3,\ldots,\lambda_{10}]=[5/100,5/100,5/100,\ldots,5/100]$	30	[−5, 5]	0
F_{16} (CF3): $f_1, f_2, f_3, \ldots, f_{10}$ = *Griewank's Function* $[\sigma_1,\sigma_2,\sigma_3,\ldots,\sigma_{10}]=[1,1,1,\ldots,1]$ $[\lambda_1,\lambda_2,\lambda_3,\ldots,\lambda_{10}]=[1,1,1,\ldots,1]$	30	[−5, 5]	0
f_{17} (CF4): f_1, f_2 = Ackley's Function f_3, f_4 = Rastrigin's Function f_5, f_6 = Weierstrass Function f_7, f_8 = Griewank's Function f_9, f_{10} = Sphere Function $[\sigma_1,\sigma_2,\sigma_3,\ldots,\sigma_{10}]=[1,1,1,\ldots,1]$ $[\lambda_1,\lambda_2,\lambda_3,\ldots,\lambda_{10}]=[5/32,5/32,1,1,5/0.5,5/0.5,5/100,5/100,5/100,5/100]$	30	[−5, 5]	0
f_{18} (CF5): f_1, f_2 = Rastrigin's Function f_3, f_4 = Weierstrass Function f_5, f_6 = Griewank's Function f_7, f_8 = Ackley's Function f_9, f_{10} = Sphere Function $[\sigma_1,\sigma_2,\sigma_3,\ldots,\sigma_{10}]=[1,1,1,\ldots,1]$ $[\lambda_1,\lambda_2,\lambda_3,\ldots,\lambda_{10}]=[1/5,1/5,5/0.5,5/0.5,5/100,5/100,5/32,5/32,5/100,5/100]$	30	[−5, 5]	0
f_{19} (CF6): f_1, f_2 = Rastrigin's Function f_3, f_4 = Weierstrass Function f_5, f_6 = Griewank's Function f_7, f_8 = Ackley's Function f_9, f_{10} = Sphere Function $[\sigma_1,\sigma_2,\sigma_3,\ldots,\sigma_{10}]=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]$ $[\lambda_1,\lambda_2,\lambda_3,\ldots,\lambda_{10}]=[0.1*1/5,0.2*1/5,0.3*5/0.5,0.4*5/0.5,0.5*5/100,0.6*5/100,0.7*5/32,0.8*5/32,0.9*5/100,1*5/100]$	30	[−5, 5]	0

As Table 4 shows, the MFO algorithm provides the best results on four of test functions. The results are followed by the FPA, PSO, and SMS algorithms. The *p*-values in Table 5, in addition, show that the superiority of the MFO algorithm is statistically significant. The MFO algorithm also provide very competitive results compared to GSA on F3, F4, and F7. The reason why the MFO algorithm does not provide superior results on three of the unimodal test functions is due to the selection of different flames for updating the position of moths. This mechanism mostly promotes exploration, so the search agents spend a large number of iterations to explore the search spaces and avoid local solutions. Since there is no local solution in unimodal test functions, this mechanism slows down the exploitation of MFO and prevents the algorithm from finding a very accurate approximation of the global optimum. Since the MFO algorithm shows the best results in 4 out of 7 unimodal test functions, it seems that this behaviour is not a major concern. It is evident that requiring moths to update their positions with respect to only the best flame will accelerate convergence and improves the accuracy of the results, but it also has negative impacts of the exploration of the algorithm, which is a very

Table 4
Results of unimodal benchmark functions.

F	*ave*	*std*	*ave*	*std*	*ave*	*std*	*ave*	*std*
	MFO		PSO		GSA		BA	
F1	0.000117	0.00015	1.321152	1.153887	608.2328	464.6545	20792.44	5892.402
F2	0.000639	0.000877	7.715564	4.132128	22.75268	3.365135	89.78561	41.95771
F3	696.7309	188.5279	736.3931	361.7818	135760.8	48652.63	62481.35	29769.17
F4	70.68646	5.275051	12.97281	2.634432	78.78198	2.814108	49.74324	10.14363
F5	139.1487	120.2607	77360.83	51156.15	741.003	781.2393	1995125	1252388
F6	0.000113	9.87E-05	286.6518	107.0796	3080.964	898.6345	17053.41	4917.567
F7	0.091155	0.04642	1.037316	0.310315	0.112975	0.037607	6.045055	3.045277
	FPA		SMS		FA		GA	
F1	203.6389	78.39843	120	0	7480.746	894.8491	21886.03	2879.58
F2	11.1687	2.919591	0.020531	0.004718	39.32533	2.465865	56.51757	5.660857
F3	237.5681	136.6463	37820	0	17357.32	1740.111	37010.29	5572.212
F4	12.57284	2.29	69.17001	3.876667	33.95356	1.86966	59.14331	4.648526
F5	10974.95	12057.29	6382246	729967	3795009	759030.3	31321418	5264496
F6	175.3808	63.45257	41439.39	3295.23	7828.726	975.2106	20964.83	3868.109
F7	0.135944	0.061212	0.04952	0.024015	1.906313	0.460056	13.37504	3.08149

Table 5
P-values of the Wilcoxon ranksum test over all runs ($p \geqslant 0.05$ have been underlined).

F	MFO	PSO	GSA	BA	FPA	SMS	FA	GA
F1	N/A	0.000183	0.000183	0.000183	0.000183	6.39E−05	0.000183	0.000183
F2	N/A	0.000183	0.000183	0.000183	0.000183	0.000183	0.000183	0.000183
F3	0.000583	0.002202	0.000183	0.000183	N/A	6.39E−05	0.000183	0.000183
F4	0.000183	0.791337	0.000183	0.000183	N/A	0.000183	0.000183	0.000183
F5	N/A	0.000183	0.005795	0.000183	0.000183	0.000183	0.000183	0.000183
F6	N/A	0.000183	0.000183	0.000183	0.000183	0.000183	0.000183	0.000183
F7	0.014019	0.000183	0.000583	0.000183	0.001008	N/A	0.000183	0.000183

important mechanism to avoid local solutions. As discussed above, unimodal functions are suitable for benchmarking exploitation of the algorithms. Therefore, these results evidence high exploitation capability of the MFO algorithm.

The statistical results of the algorithms on multimodal test function are presented in Table 6. It may be seen that the MFO algorithm highly outperforms other algorithms on F8, F10, F11, and F13. Table 7 suggests that this superiority is statistically significant. The MFO algorithm failed to show the best results on F10 and F12. As per the *p*-values in Table 7, the MFO algorithm is the only algorithm that provides a *p*-value greater than 0.05 on F12, which means that the superiority of the GSA algorithm is not statistically significant. In other words, MFO and GSA perform very similar and can be considered as the best algorithms when solving F12. In F9, however, the superiority of the GSA algorithm is statistically significant. It should be noted that the MFO algorithm outperforms other algorithms on this test function except GSA. Due to the low discrepancy of the results of MFO and GSA, it is evident that both algorithms determined the optimum, but it seems that MFO failed to exploit the obtained optimum and improve its accuracy. This is again due to the higher exploration of the MFO algorithm, which prevents this algorithm from finding an accurate approximation of the F9's global optimum. Despite this behaviour on only one function, the results of other multi-modal test functions strongly prove that high exploration of the MFO algorithm is fruitful for avoiding local solutions.

Since the multi-modal functions have an exponential number of local solutions, there results show that the MFO algorithm is able to explore the search space extensively and find promising regions of the search space. In addition, high local optima avoidance of this algorithm is another finding that can be inferred from these results.

The rest of the results, which belong to F14–F19, are provided in Tables 8 and 9. The results are consistent with those of other test

Table 6
Results of multimodal benchmark functions.

F	*ave*	*std*	*ave*	*std*	*ave*	*std*	*ave*	*std*
	MFO		PSO		GSA		BA	
F8	−8496.78	725.8737	−3571	430.7989	−2352.32	382.167	65535	0
F9	84.60009	16.16658	124.2973	14.25096	31.00014	13.66054	96.21527	19.58755
F10	1.260383	0.72956	9.167938	1.568982	3.740988	0.171265	15.94609	0.774952
F11	0.01908	0.021732	12.41865	4.165835	0.486826	0.049785	220.2812	54.70668
F12	0.894006	0.88127	13.87378	5.85373	0.46344	0.137598	28934354	2178683
F13	0.115824	0.193042	11813.5	30701.9	7.617114	1.22532	1.09E+08	1.05E+08
	FPA		SMS		FA		GA	
F8	−8086.74	155.3466	−3942.82	404.1603	−3662.05	214.1636	−6331.19	332.5668
F9	92.69172	14.22398	152.8442	18.55352	214.8951	17.21912	236.8264	19.03359
F10	6.844839	1.249984	19.13259	0.238525	14.56769	0.467512	17.84619	0.531147
F11	2.716079	0.727717	420.5251	25.25612	69.65755	12.11393	179.9046	32.43956
F12	4.105339	1.043492	8742814	1405679	368400.8	172132.9	34131682	1893429
F13	62.3985	94.84298	1E+08	0	5557661	1689995	1.08E+08	3849748

functions, in which the MFO shows very competitive results compared to other algorithms. The *p*-values also prove that the superiorities are statistically significant occasionally. Although the MFO algorithm does not provide better results on half of the composite test functions (F15, F17, and F19), the *p*-values in Table 9 show that the results of this algorithm are very competitive. The composite functions have very difficult search spaces, so the accurate approximation of their global optima needs high exploration and exploitation combined. The results evidence that the MFO algorithm properly balances these two conflicting milestones.

So far, the results are discussed in terms of exploration and exploitation. Although these results indirectly show that the MFO algorithm convergences to a point in a search space and improves the initial solutions, the convergence of the MFO algorithm is investigated further in the following paragraphs. To confirm the convergence of the MFO algorithm, four metrics are employed as follows:

- Search history.
- Trajectory of the first moth in its first dimension.
- Average fitness of all moths.
- Convergence rate.

The experiments are re-done on some of the test functions with 2 variables and using 5 moths over 100 iterations. Tue results are provided in Fig. 8.

The first metric is a qualitative metric that show the history of sampled points over the course of iterations. The sampled points during optimization are illustrated using black points in Fig. 8. It seems that the MFO algorithm follows a similar pattern on all of the test functions, in which the moths tend to explore promising regions of the search space and exploit very accurately around the global optima. These observations prove that the MFO algorithm can be very effective in approximating the global optimum of optimization problems.

The second metric, which is another qualitative metric, shows the changes in the first dimension of the first moth during optimization. This metric assists us to observe if the first moth (as a representative of all moths) faces abrupt movements in the initial iterations and gradual changes in the final iterations. According to Berg et al. [57], this behaviour can guarantee that a population-based algorithm eventually convergences to a point and searches locally in a search space. The trajectories in Fig. 8 show that the first moth starts the optimization with sudden changes (more than 50% of the search space). This behaviour can guarantee the exploration of the search space. It may also be observed that the fluctuations are decreased gradually over the course of iteration, a behaviour that guarantees transition between exploration and exploitation. Eventually, the movement of moth becomes very gradual which causes the exploitation of the search space.

Table 7
P-values of the Wilcoxon ranksum test over all runs ($p \geqslant 0.05$ have been underlined).

F	MFO	PSO	GSA	BA	FPA	SMS	FA	GA
F8	N/A	0.000183	0.000183	6.39E−05	0.161972	0.000183	0.000183	0.000183
F9	0.000181	0.000181	N/A	0.000181	0.000181	0.000181	0.000181	0.000181
F10	N/A	0.000183	0.000183	0.000183	0.000183	0.000183	0.000183	0.000183
F11	N/A	0.000183	0.000183	0.000183	0.000183	0.000183	0.000183	0.000183
F12	0.472676	0.000183	N/A	0.000183	0.000183	0.000182	0.000183	0.000183
F13	N/A	0.000183	0.000183	0.000183	0.000183	6.39E−05	0.000183	0.000183

Table 8
Results of composite benchmark functions.

F	*ave*	*std*	*ave*	*std*	*ave*	*std*	*ave*	*std*
	MFO		PSO		GSA		BA	
F14	8.25E−31	1.08E−30	137.7789	116.3128	5.43E−19	1.35E−19	130.3125	118.8206
F15	66.73272	53.22555	166.6643	164.3894	20.35852	63.12427	544.1045	149.381
F16	119.0146	28.3318	394.507	121.949	245.3021	49.05264	696.9752	190.5441
F17	345.4688	43.11578	486.3534	67.31685	315.2086	100.7477	745.1403	143.1577
F18	10.4086	3.747669	256.5258	200.3816	70	48.30459	543.8894	198.8883
F19	706.9953	194.9068	790.1284	189.4915	881.6392	45.17728	896.355	86.29955
	FPA		SMS		FA		GA	
F14	10.09454	31.59138	105.7572	26.8788	175.9715	86.928	92.13909	27.90131
F15	11.41158	3.380957	156.463	68.24926	353.6269	103.423	96.70927	9.703147
F16	234.9341	39.60663	406.9962	65.39732	308.0516	37.435	369.1036	42.84275
F17	355.3807	20.61705	518.6931	42.74199	548.5276	162.8993	450.829	31.54446
F18	54.78722	42.05824	153.6984	96.91419	175.1975	83.15078	95.92017	53.79146
F19	573.0955	149.1538	611.5401	154.8529	829.5929	157.2787	523.7037	22.92001

Table 9
P-values of the Wilcoxon ranksum test over all runs ($p \geqslant 0.05$ have been underlined).

F	MFO	PSO	GSA	BA	FPA	SMS	FA	GA
F14	N/A	0.000172	0.000172	0.000172	0.000172	0.000172	0.000172	0.000172
F15	0.000583	0.002202	0.002786	0.000183	N/A	0.000183	0.000183	0.000183
F16	N/A	0.000183	0.000183	0.000183	0.000246	0.000183	0.000183	0.000183
F17	0.004586	0.002202	N/A	0.00033	0.002827	0.002827	0.001008	0.002827
F18	N/A	0.000183	0.140465	0.000183	0.002827	0.000183	0.000183	0.000183
F19	0.241322	0.088973	0.000183	0.000183	0.10411	0.014019	0.001315	N/A

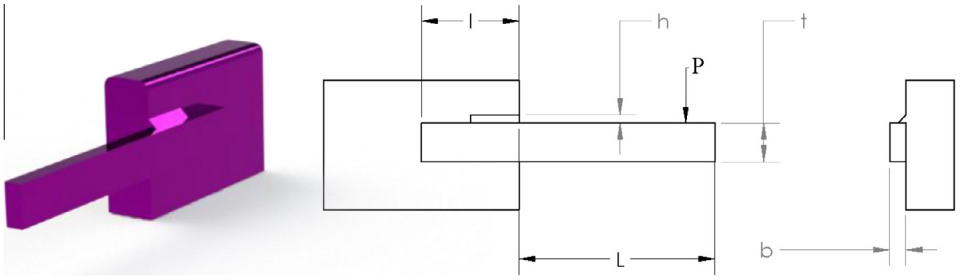

Fig. 9. Design parameters of the welded beam design problem.

The third metric is a quantitative measure and averages the fitness of all moths in each iteration. If an algorithm improves its candidate solutions, obviously, the average of fitness should be improved over the course of iterations. As the average fitness curves in Fig. 8 suggest, the MFO algorithm shows degrading fitness on all of the test functions. Another fact worth mentioning here is the accelerated decrease in the average fitness curves, which shows that the improvement of the candidate solutions becomes faster and better over the course of iterations. This is due to the fact that the MFO algorithm is required to adaptively decrease the number of flames, so the moths tend to converge and fly around fewer flames as the iteration counter increases. In addition, the adaptive convergence constraint (r) promotes this behaviour.

The last quantitative comparison metric here is the convergence rate of the MFO algorithm. The fitness of the best flame in each iteration is saved and drawn as the convergence curves in Fig. 8. The reduction of fitness over the iterations proves the convergence of the MFO algorithm. It is also interesting that the accelerated degrade can also be observed in the convergence curves as well, which is due the above-discussed reason.

Table 10
Comparison results of the welded beam design problem.

Algorithm	Optimal values for variables				Optimal cost
	h	*l*	*t*	*b*	
MFO	0.2057	3.4703	9.0364	0.2057	1.72452
GSA	0.182129	3.856979	10.0000	0.202376	1.87995
CPSO [66]	0.202369	3.544214	9.048210	0.205723	1.73148
GA [60]	0.1829	4.0483	9.3666	0.2059	1.82420
GA [62]	0.2489	6.1730	8.1789	0.2533	2.43312
Coello [58]	0.208800	3.420500	8.997500	0.2100	1.74831
Coello and Montes [67]	0.205986	3.471328	9.020224	0.206480	1.72822
Siddall [68]	0.2444	6.2189	8.2915	0.2444	2.38154
Ragsdell [65]	0.2455	6.1960	8.2730	0.2455	2.38594
Random [65]	0.4575	4.7313	5.0853	0.6600	4.11856
Simplex [65]	0.2792	5.6256	7.7512	0.2796	2.53073
David [65]	0.2434	6.2552	8.2915	0.2444	2.38411
APPROX [65]	0.2444	6.2189	8.2915	0.2444	2.38154

As a summary, the results of this section experimentally proved that he MFO algorithm is able to show very competitive results and occasionally outperforms other well-known algorithms on the test functions. In addition, the convergence of the MFO algorithm was experimentally proved by two qualitative and two quantitative measures. Therefore, it can be stated that the MFO algorithm is able to be effective in solving real problem as well. Since constraints are one of the major challenges in solving real problems and the main objective of designing the MFO algorithm is to solve real problems, nine constrained real engineering problems are employed in the next section to further investigate the performance of the MFO algorithm and provide a comprehensive study.

5. Constrained optimization using the MFO algorithm

Constraints handling refers to the process of considering both inequality and equality constraints during optimization. Constraints divide the candidate solutions of heuristic algorithms into two groups: feasible and infeasible. According to Coello Coello [58], there are different methods of handling constraints: penalty functions, special operators, repair algorithms, separation of objectives and constraints, and hybrid methods. Among these techniques, the most straightforward method is the penalty function. Such methods penalize the infeasible candidate solutions and convert constrained optimization to an unconstrained optimization. There are different types of penalty functions as well [58]: static, dynamic, annealing, adaptive, co-evolutionary, and death penalty. The last penalty function, death penalty, is the simplest method, which assigns a big objective function (in case of minimization). This process automatically causes discarding the infeasible solutions by the heuristic algorithms during optimization. The advantages of this method are simplicity and low computational cost. However, this method does not utilize the information of infeasible solutions that might be helpful when solving problems with dominated infeasible regions. For the sake of simplicity, the MFO algorithm is equipped with a death penalty function in this section to handle constraints. Therefore, a moth would be assigned a big objective had it violates any of the constraints.

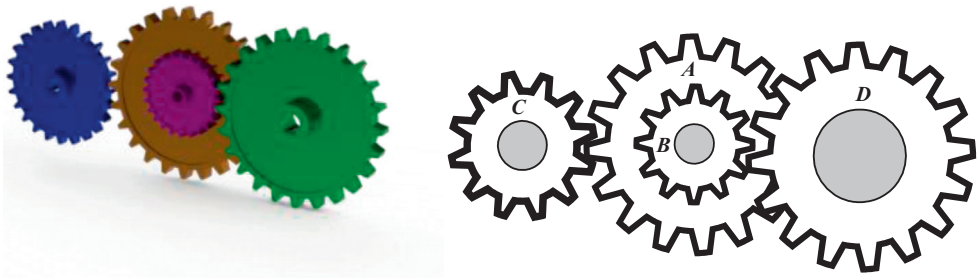

Fig. 10. Gear train design problem.

5.1. Welded beam design problem

This problem is a well-known problem in the field of structural optimization [59], in which the fabrication cost of a welded beam should be minimized. As can be seen in Fig. 9 and Appendix A, there are four parameters for this problem and seven constraints.

Table 11
Comparison results of the gear train design problem.

Algorithm	Optimal values for variables				f
	n_A	n_B	n_C	n_D	
MFO	43	19	16	49	2.7009e-012
ABC [40]	49	16	19	43	2.7009e−012
MBA [40]	43	16	19	49	2.7009e−012
GA [71]	49	16	19	43	2.701 9e−012
CS [72]	43	16	19	49	2.7009e−012
ISA [69]	N/A	N/A	N/A	N/A	2.7009e−012
Kannan and Kramer [73]	33	15	13	41	2.1469e−08

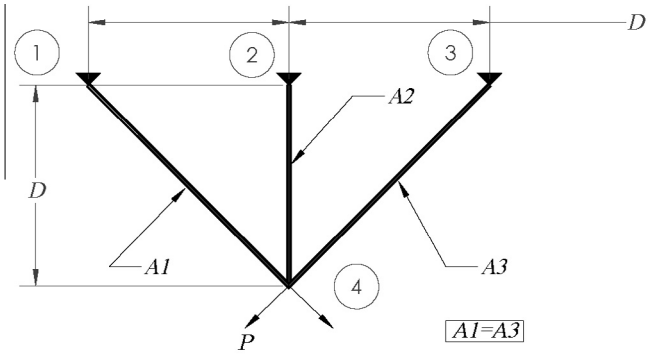

Fig. 11. Three-bar truss design problem.

Table 12
Comparison results of the three-bar truss design problem.

Algorithm	Optimal values for variables		Optimal weight
	x_1	x_2	
MFO	0.788244770931922	0.409466905784741	263.895979682
DEDS [74]	0.78867513	0.40824828	263.8958434
PSO-DE [75]	0.7886751	0.4082482	263.8958433
MBA [40]	0.7885650	0.4085597	263.8958522
Ray and Sain [76]	0.795	0.395	264.3
Tsa [77]	0.788	0.408	263.68
CS [72]	0.78867	0.40902	263.9716

This problem is solved by the MFO algorithm and compared to GSA, GA [60–62], CPSO [63], HS [64], Richardson's random method, Simplex method, Davidon–Fletcher–Powell, and Griffith and Stewart's successive linear approximation [65]. Table 10 shows the best obtained results.

The results of Table 10 show that the MFO algorithm is able to find the best optimal design compared to other algorithms. The results of MFO are closely followed by the CPSO algorithm.

5.2. Gear train design problem

This is a mechanical engineering problem which aims for the minimization of gear ratio $\left(Gear\ ratio = \frac{angular\ velocity\ of\ output\ shaft}{angular\ velocity\ of\ input\ shaft}\right)$ for a given set of four gears of a train [69,70]. The parameters are the number of teeth of the gears, so there is a total of 4 variables for this problem. There is no constraint in this problem, but the range of variables are considered as constraints. The overall schematic of the system is illustrated in Fig. 10.

This problem is solved with MFO and the results are compared to ABC, MBA, GA, CS, and ISA in Table 11.

The gear train is a discrete problem, so the position of moths is rounded in each iteration for solving this problem. Table 11 shows that the MFO algorithm finds the same optimal gear ratio value compared to ABC, MBA, CS, and ISA. This proves that MFO can be effective in solving discrete problems as well. It is also worth noticing here that although the gear ratio is equal, the obtained optimal design parameters are different. So, MFO finds a new optimal design for this problem.

5.3. Three-bar truss design problem

Three-bas truss design problem is another structural optimization problem in the field of civil engineering, in which two parameters should be manipulated in order to achieve the least weight subject to stress, deflection, and buckling constraints. This problem has been mostly utilized because of its difficult constrained search space [40,72]. Different components of this problem can be seen in Fig. 11. The formulation of this problem is also available in Appendix A.

This problem is again solved using MFO and compared to DEDS, PSO-DE, MBA, Tsa, and CS algorithms in the literature. Table 12 includes the optimal values for the variables and the optimal weights obtained.

The results of the algorithms in three-bar truss design problem show that MFO outperforms three of the algorithms.

5.4. Pressure vessel design problem

This problem, which is very popular in the literature, has four parameters and four constraints. The objective is to obtain a design

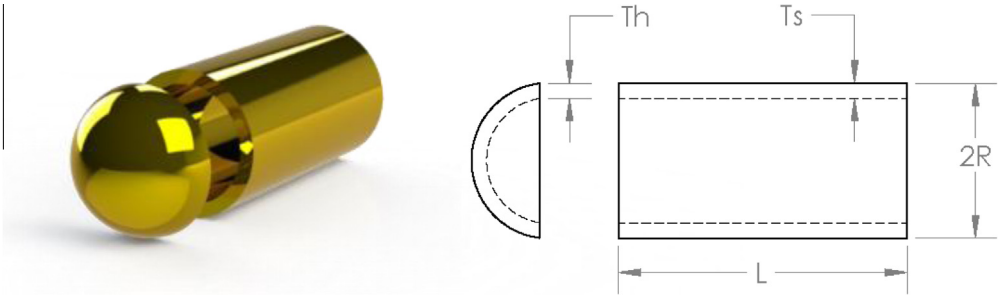

Fig. 12. Pressure vessel design problem.

for a pressure vessel with the least fabrication cost. Fig. 12 shows the pressure vessel and parameters involved in the design [72,78].

The structure of the pressure vessel is optimized with MFO and the results are compared to GSA, PSO [66], GA [67,79,80], ES [81], DE [82], and ACO [83], augmented Lagrangian Multiplier [84], and branch-and-bound [85] in Table 13.

This table shows that the MFO algorithm finds the second low-cost design. The problem formulation in Appendix A shows that this problem is highly constrained, so the results evidence the merits of MFO in solving such problems.

Table 13
Comparison results for pressure vessel design problem.

Algorithm	Optimal values for variables				Optimum cost
	T_s	T_h	R	L	
MFO	0.8125	0.4375	42.098445	176.636596	6059.7143
GSA	1.1250	0.6250	55.988659	84.4542025	8538.8359
PSO [66]	0.8125	0.4375	42.091266	176.746500	6061.0777
GA [79]	0.8125	0.4345	40.323900	200.000000	6288.7445
GA [67]	0.8125	0.4375	42.097398	176.654050	6059.9463
GA [80]	0.9375	0.5000	48.329000	112.679000	6410.3811
ES [81]	0.8125	0.4375	42.098087	176.640518	6059.7456
DE [82]	0.8125	0.4375	42.098411	176.637690	6059.7340
ACO [83]	0.8125	0.4375	42.103624	176.572656	6059.0888
Lagrangian multiplier [84]	1.1250	0.6250	58.291000	43.6900000	7198.0428
Branch-bound [85]	1.1250	0.6250	47.700000	117.701000	8129.1036

5.5. Cantilever beam design problem

Cantilever beam consists of five hollow blocks. Fig. 13 and the problem formulation in Appendix A show that the blocks are square so the number of parameters is five. There is also one constraint that should not be violated by the final optimal design. The comparison results between MFO and MMA, GCA_I, GCA_II, CS, and SOS are provided in Table 14.

The problem formulation for this case study in Appendix A shows that the constraints are only applied to the variables ranges. This problem is different from other employed problems in this section, so it can mimic another characteristic of real problems. The results in Table 14 testify that the MFO algorithm is able to solve these types of problems efficiently as well. The results evidence that the design with minimum weight belongs to the proposed algorithm.

5.6. I-beam design problem

Another structural optimization problem employed in this section is the I-beam design problem. This problem deals with designing of an I-shaped beam (as shown in Fig. 14) for achieving minimal vertical deflection. Length, height, and two thicknesses are the structural parameters for this problem. Formulation of this problem in Appendix A shows that this problem has a constraint as well.

The results of MFO on this problem are compared to those of adaptive response surface method (ARSM), Improved ARSM (IARSM), CS, and SOS in the literature. Table 15 shows the experimental results on this problem.

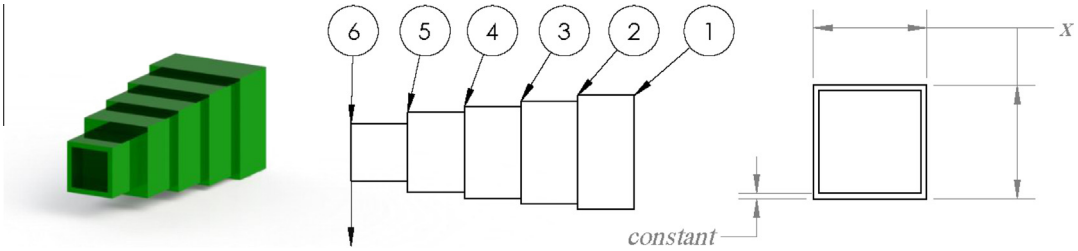

Fig. 13. Cantilever beam design problem.

Table 14
Comparison results for cantilever design problem.

Algorithm	Optimal values for variables					Optimum weight
	x_1	x_2	x_3	x_4	x_5	
MFO	5.9848717732166	5.31672692429783	4.49733258583062	3.51361646768954	2.16162029338550	1.33998808597181
MMA [86]	6.0100	5.3000	4.4900	3.4900	2.1500	1.3400
GCA_I [86]	6.0100	5.3000	4.4900	3.4900	2.1500	1.3400
GCA_II [86]	6.0100	5.3000	4.4900	3.4900	2.1500	1.3400
CS [72]	6.0089	5.3049	4.5023	3.5077	2.1504	1.33999
SOS [41]	6.01878	5.30344	4.49587	3.49896	2.15564	1.33996

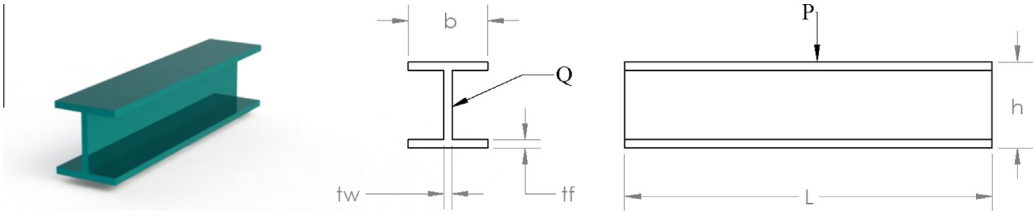

Fig. 14. I-beam design problem.

This table shows that the MFO algorithm is able to find a design with minimal vertical deflection compared to other algorithms. It is worth mentioning here that the improved vertical deflection is very significant in this case study.

5.7. Tension/compression spring design

The other utilized engineering test problem is the tension/compression spring design problem. The objective is again the minimization of the fabrication cost of a spring with three structural parameters [67,88,89]: wire diameter (d), mean coil diameter (D), and the number of active coils (N). Fig. 15 shows the spring and its parameters.

There are several solutions for this problem in the literature. This problem was solved using meta-heuristics such as PSO [66], ES [81], GA [79], HS [78], and DE [82]. The mathematical approaches are the numerical optimization technique (constraints correction at constant cost) [88] and mathematical optimization technique [89]. The best results of MFO are compared with those of all the above-mentioned methods in Table 16. Note that a similar penalty function in [90] is used for MFO to perform a fair comparison.

Table 16 shows that MFO performs very effectively when solving this problem and provides the best design. The results of MFO are very close to those of HS and DE, and PSO.

Table 15
Comparison results for I-beam design problem.

Algorithm	Optimal values for variables				Optimum vertical deflection
	b	h	t_w	t_f	
MFO	50	80	1.7647	5.0000	0.0066259
ARSM [87]	37.05	80	1.71	2.31	0.0157
IARSM [87]	48.42	79.99	0.90	2.40	0.131
CS [72]	50	80	0.9	2.321675	0.0130747
SOS [41]	50	80	0.9	2.32179	0.0130741

5.8. 15-bar truss design

This problem is a structural design problem, in which the objective is to minimize the weight of a 15-bar truss. The final optimal design for this problem should satisfy 46 constraints such as 15 tension, 15 compression, and 16 displacement constraints. There are also 8 nodes and 15 bars as shown in Fig. 16, so there is the total number of 15 variables. It also may be seen in this figure that three loads are applied to the nodes $P1$, $P2$, and $P3$. Other assumptions for this problem are as follows:

- $\rho = 7800\ \text{kg/m}^3$
- $E = 200\ \text{MPa}$
- *Stress limitation* $= \pm 120\ \text{MPa}$
- *Displacement in both directions* $= \pm 10\ \text{mm}$
- *Design variabe set* $=$
$$\left\{ \begin{matrix} 113.2, 143.2, 145.9, 174.9, 185.9, 235.9, 265.9, 297.1 \\ 308.6, 334.3, 338.2, 497.8, 507.6, 736.7, 791.2, 1063.7 \end{matrix} \right\}$$

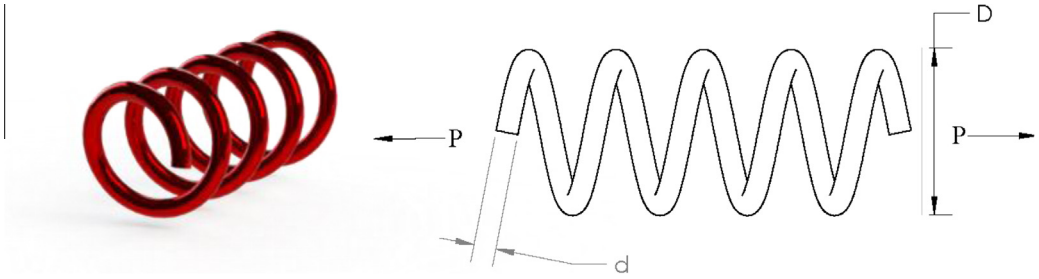

Fig. 15. Tension/compression spring design problem.

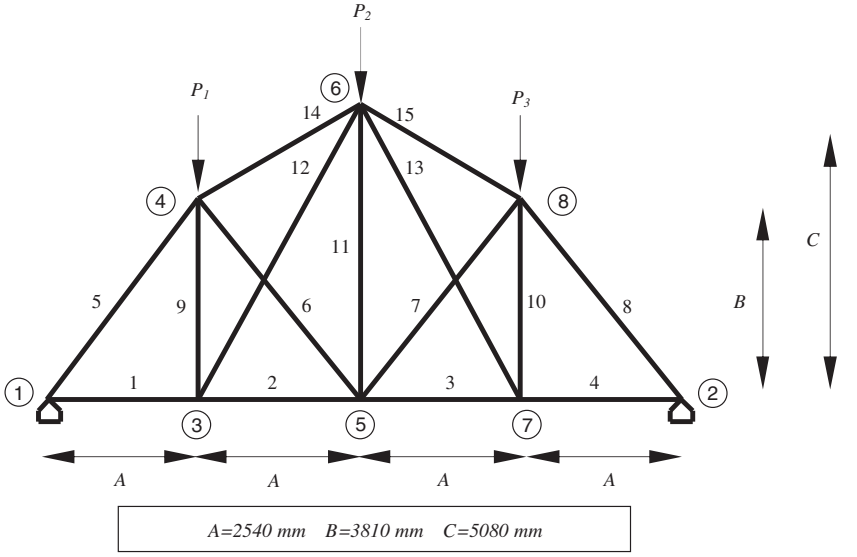

Fig. 16. Structure of a 15-bar truss.

This problem has been solved widely in the literature with considering $P_1 = 35$ kN, $P_2 = 35$ kN, $P_3 = 35$ kN as the loads [91,92]:

These three cases are solved using 30 search agents over 500 iterations and the results are presented in Table 17. Since this problem is a discrete problem, the search agents of MFO were simply rounded to the nearest integer number during optimization.

Table 17 shows that the MFO algorithm is able to find a similar structure compared to those of HPSO, SOS, and MBA. This is the best obtained optimum so far in the literature for this problem. Therefore, these results show that MFO is able to provide very competitive results in solving this problem as well.

5.9. 52-bar truss design

The last case study is another popular truss design problem. As may be seen in Fig. 17, there are 52 bars and 20 nodes of which four are fixed. The truss has 52 bars, which are classified in the following 12 groups:

- $Group1 : A_1,\ A_2,\ A_3,\ A_4$
- $Group2 : A_5,\ A_6,\ A_7,\ A_8,\ A_9,\ A_{10}$
- $Group3 : A_{11},\ A_{12},\ A_{13}$
- $Group4 : A_{14},\ A_{15},\ A_{16},\ A_{17}$
- $Group5 : A_{18},\ A_{19},\ A_{20},\ A_{21},\ A_{22},\ A_{23}$
- $Group6 : A_{24},\ A_{25},\ A_{26}$
- $Group7 : A_{27},\ A_{28},\ A_{29},\ A_{30}$
- $Group8 : A_{31},\ A_{32},\ A_{33},\ A_{34},\ A_{35},\ A_{36}$
- $Group9 : A_{37},\ A_{38},\ A_{39}$
- $Group10 : A_{40},\ A_{41},\ A_{42},\ A_{43}$
- $Group11 : A_{44},\ A_{45},\ A_{46},\ A_{47},\ A_{48},\ A_{49}$
- $Group12 : A_{50},\ A_{51},\ A_{52}$

Therefore, this problem has 12 parameters to be optimized. Other assumptions for this problem are as follows:

- $\rho = 7860.0$ kg/m^3
- $E = 2.07e5$ MPa
- *Stress limitation* $= \pm 180$ MPa
- *Design variabe set are chosen from* Table 16
- $P_k = 100$ kN, $P_y = 200$ kN

Available cross-section areas of the AISC norm for this problem are available in Table 18. Again, 30 search agents are employed

Table 16
Comparison of results for tension/compression spring design problem.

Algorithm	Optimum variables			Optimum weight
	d	*D*	*N*	
MFO	0.051994457	0.36410932	10.868421862	0.0126669
GSA	0.050276	0.323680	13.525410	0.0127022
PSO [66]	0.051728	0.357644	11.244543	0.0126747
ES [81]	0.051989	0.363965	10.890522	0.0126810
GA [79]	0.051480	0.351661	11.632201	0.0127048
HS [78]	0.051154	0.349871	12.076432	0.0126706
DE [82]	0.051609	0.354714	11.410831	0.0126702
Mathematical optimization [89]	0.053396	0.399180	9.1854000	0.0127303
Constraint correction [88]	0.050000	0.315900	14.250000	0.0128334

Table 18
Available cross-section areas of the AISC norm (valid values for the parameters).

No.	in.^{2}	mm^2	No.	in.^{2}	mm^2
1	0.111	71.613	33	3.84	2477.414
2	0.141	90.968	34	3.87	2496.769
3	0.196	126.451	35	3.88	2503.221
4	0.25	161.29	36	4.18	2696.769
5	0.307	198.064	37	4.22	2722.575
6	0.391	252.258	38	4.49	2896.768
7	0.442	285.161	39	4.59	2961.284
8	0.563	363.225	40	4.8	3096.768
9	0.602	388.386	41	4.97	3206.445
10	0.766	494.193	42	5.12	3303.219
11	0.785	506.451	43	5.74	3703.218
12	0.994	641.289	44	7.22	4658.055
13	1	645.16	45	7.97	5141.925
14	1.228	792.256	46	8.53	5503.215
15	1.266	816.773	47	9.3	5999.988
16	1.457	939.998	48	10.85	6999.986
17	1.563	1008.385	49	11.5	7419.34
18	1.62	1045.159	50	13.5	8709.66
19	1.8	1161.288	51	13.9	8967.724
20	1.99	1283.868	52	14.2	9161.272
21	2.13	1374.191	53	15.5	9999.98
22	2.38	1535.481	54	16	10322.56
23	2.62	1690.319	55	16.9	10903.2
24	2.63	1696.771	56	18.8	12129.01
25	2.88	1858.061	57	19.9	12838.68
26	2.93	1890.319	58	22	14193.52
27	3.09	1993.544	59	22.9	14774.16
28	3.13	2019.351	60	24.5	15806.42
29	3.38	2180.641	61	26.5	17096.74
30	3.47	2238.705	62	28	18064.48
31	3.55	2290.318	63	30	19354.8
32	3.63	2341.931	64	33.5	21612.86

Table 17
Comparison of MFO optimization results with literature for the 15-bar truss design problem.

Variables (mm^2)	GA [30]	PSO [31]	PSOPC [31]	HPSO [31]	MBA [92]	SOS [41]	MFO
A1	308.6	185.9	113.2	113.2	113.2	113.2	113.2
A2	174.9	113.2	113.2	113.2	113.2	113.2	113.2
A3	338.2	143.2	113.2	113.2	113.2	113.2	113.2
A4	143.2	113.2	113.2	113.2	113.2	113.2	113.2
A5	736.7	736.7	736.7	736.7	736.7	736.7	736.7
A6	185.9	143.2	113.2	113.2	113.2	113.2	113.2
A7	265.9	113.2	113.2	113.2	113.2	113.2	113.2
A8	507.6	736.7	736.7	736.7	736.7	736.7	736.7
A9	143.2	113.2	113.2	113.2	113.2	113.2	113.2
A10	507.6	113.2	113.2	113.2	113.2	113.2	113.2
A11	279.1	113.2	113.2	113.2	113.2	113.2	113.2
A12	174.9	113.2	113.2	113.2	113.2	113.2	113.2
A13	297.1	113.2	185.9	113.2	113.2	113.2	113.2
A14	235.9	334.3	334.3	334.3	334.3	334.3	334.3
A15	265.9	334.3	334.3	334.3	334.3	334.3	334.3
Optimal weight (kg)	142.117	108.84	108.96	105.735	105.735	105.735	105.735

over 500 iterations for solving this problem. Similarly to 15-bar truss design, the search agents of MFO were rounded to the nearest integer number during optimization since this problem is discrete

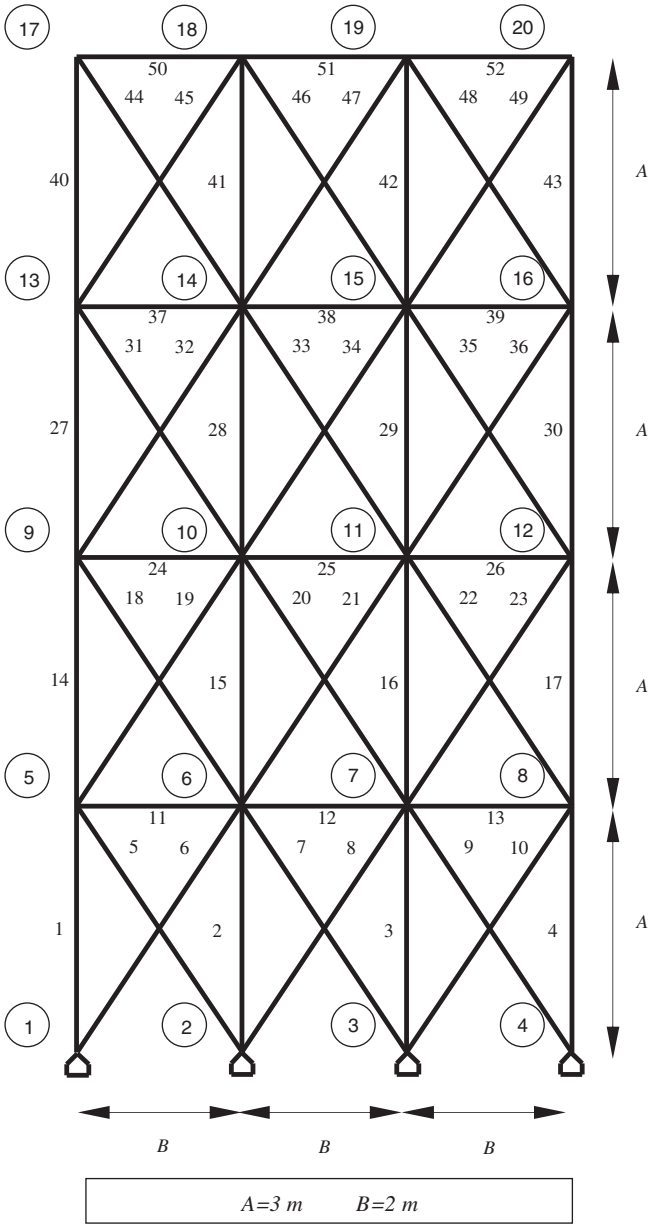

Fig. 17. Structure of a 52-bar truss.

as well. The results are presented and compared to several algorithms in the literature in Table 19.

As per the results in Table 19, the best optimal weight obtained by MFO is 1902.605, which is identical to the optimal weights found by SOS and MBA. It is evident from the results that MFO, SOS, and MBA significantly outperformed PSO, PSOPC, HPSO, and DHPSACO.

As a summary, the results of this section show that MFO outperforms other algorithms in the majority of real case studies. Since the search space of these problems is unknown, these results are strong evidences for the applicability of MFO in solving real problems. Due to the constrained nature of the case studies, in addition, it can be stated that the MFO algorithm is able to optimize search spaces with infeasible regions as well. This is due to the update mechanism of moths, in which they are required to update their positions with respect to the best recent feasible flames. Therefore, this approach promotes exploration of promising feasible regions and is the main reason of the superiority of the MFO algorithm.

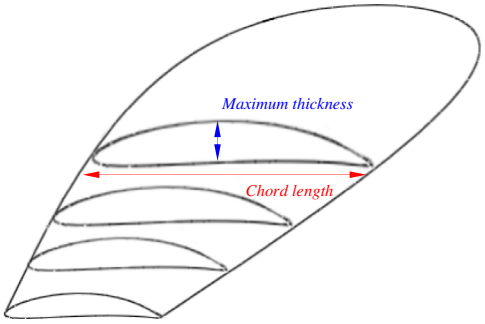

Fig. 19. Airfoils MFOng the blade define the shape of the propeller [95].

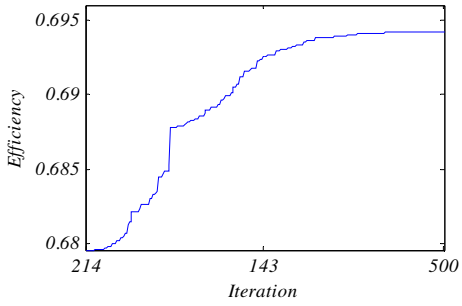

Fig. 20. Convergence of the MFO algorithm when solving the propeller design problem.

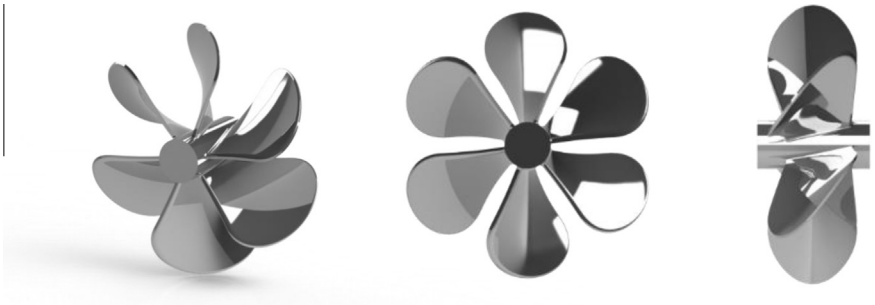

Fig. 18. Fixed pitch ship propeller with 4 blades and 2 m diameter.

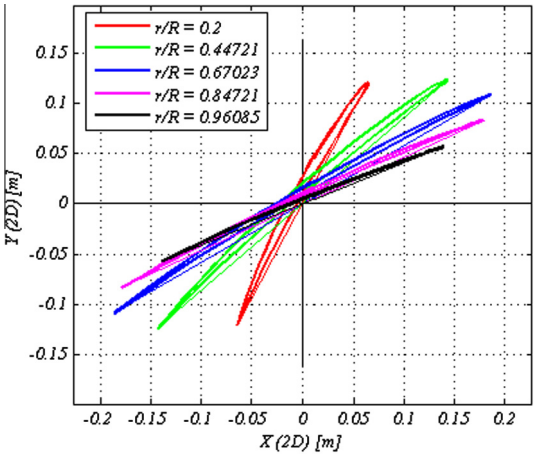

Fig. 21. 2D airfoils of the obtained optimal design using MFO.

To further demonstrate the performance of the proposed MFO algorithm, next section is devoted to the real application of MFO in the field of Computational Fluid Dynamics (CFD) problems.

6. Marine propeller design using MFO

In this section the shape of a marine propeller is optimized by MFO. The selected propeller is a ship propeller with the diameter of two meters. The shape of the propeller is illustrated in Fig. 18.

Generally speaking, the efficiency of a marine propeller is a critical performance metric because of the density of water. The ultimate goal when designing a marine propeller is to convert the rotation power of motor to thrust with the least loss. Note that there is always 1–5% intrinsic loss due to swirl for marine propellers. The efficiency of marine propellers is calculated as follows [94]:

$$\eta = \frac{V_a}{2\pi n D} \times \frac{K_T(x)}{K_Q(x)} \tag{6.1}$$

where V is axial velocity, D is the diameter of the propeller, n is the rotation speed of the propeller, K_T indicates the thrust coefficient, and K_Q shows that torque coefficient.

K_T is calculated as follows:

$$K_T(x) = \frac{T}{\rho n^2 D^2} \tag{6.2}$$

where ρ shows the fluid density, T is the thrust, n indicates the rotation speed of the propeller, and D is the diameter length.

In order to mathematically model the shape of the blades, Bézier curves can be chosen. In this method, a set of controlling points define the shape of the airfoils along the blades. Another method of designing a propeller is to select and define the type and shape of airfoils along the blades. Due to the simplicity, the second method is utilized in this work. In the employed propeller, the blade's arifoils are determined by NACA $a = 0.8$ meanline and NACA 65A010 thickness sections. The shape of airfoils across the blade define the final shape of the propeller.

As shown in Fig. 19, the blades are divided into 10 cross sections, and each cross section has two parameters: chord length and thickness. Therefore, there are 20 parameters for this problem.

After all, the problem of propeller design is formulated as a maximization problem as follows:

$$\textit{Suppose}: \quad \vec{X}_i = (x_0^i, x_{0.1R}^i, \ldots, x_R^i), \quad i = 1,2,3,4,5,6,7,8,9,10 \tag{6.3}$$

$$\textit{Maximize}: \quad \eta(x) \tag{6.4}$$

$$\textit{Subject to}: \quad \textit{wake friction and thrust deduction} \tag{6.5}$$

$$\textit{Parameter range}: \quad 0 < x_1 - x_{10} \leqslant 1 \tag{6.6}$$

The CFD problems are usually very challenging with dominated infeasible regions. Therefore, this problem is a very hard test bed for the proposed MFO algorithm. It is also worth mentioning here that propeller design is an expensive problem because each function evaluation takes around 2–4 min. Note that a freeware called Openprop is utilized as the simulator for calculating efficiency [96]. The constant parameters of the propeller during optimization are as follows:

- Ship speed: 5 m/s (9.7192 knots).
- Rotation speed: 170 RPM.
- Diameter: 2 m.
- Number of blades: 6.
- Thrust: 40,000 N.
- Torque: 16183.1936 Nm.
- Power: 288099.0115 W.
- Density of water: 999.97 kg/m^3.

Sixty moths are employed over 500 iterations to solve this problem. The best obtained design parameters are presented in Table 20.

The best efficiency obtained by the MFO algorithm was 0.6942. Other characteristics of the obtained optimal design are presented in Table 21.

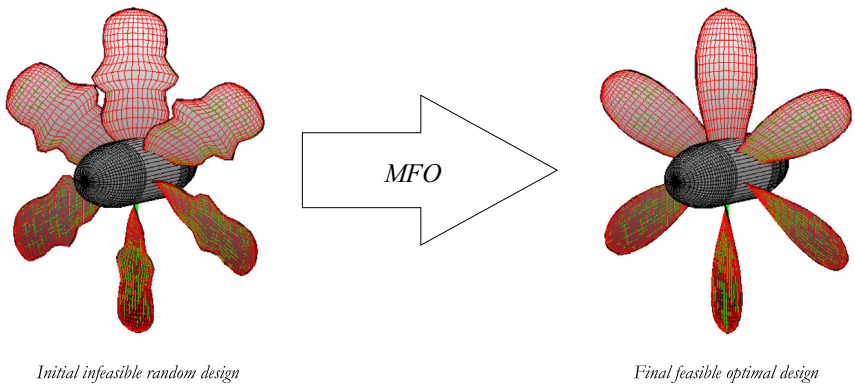

Fig. 22. Improved design from initial infeasible design to feasible optimal design.

Table 19
Comparison of MFO optimization results with literature for the 52-bar truss design problem.

Variables (mm^2)	PSO [91]	PSOPC [91]	HPSO [91]	DHPSACO [93]	MBA [92]	SOS [41]	MFO
A1–A4	4658.055	5999.988	4658.055	4658.055	4658.055	4658.055	4658.055
A5–A10	1374.19	1008.38	1161.288	1161.288	1161.288	1161.288	1161.288
A11–A13	1858.06	2696.77	363.225	494.193	494.193	494.193	494.193
A14–A17	3206.44	3206.44	3303.219	3303.219	3303.219	3303.219	3303.219
A18–A23	1283.87	1161.29	940	1008.385	940	940	940
A24–A26	252.26	729.03	494.193	285.161	494.193	494.193	494.193
A27–A30	3303.22	2238.71	2238.705	2290.318	2238.705	2238.705	2238.705
A31–A36	1045.16	1008.38	1008.385	1008.385	1008.385	1008.385	1008.385
A37–A39	126.45	494.19	388.386	388.386	494.193	494.193	494.193
A40–A43	2341.93	1283.87	1283.868	1283.868	1283.868	1283.868	1283.868
A44–A49	1008.38	1161.29	1161.288	1161.288	1161.288	1161.288	1161.288
A50–A52	1045.16	494.19	792.256	506.451	494.193	494.193	494.193
Optimal weight (kg)	2230.16	2146.63	1905.495	1904.83	1902.605	1902.605	1902.605

Table 20
Obtained best design parameters using MFO.

Chord1	Thickness1	Chord2	Thickness2	Chord3	Thickness3	Chord4	Thickness4	Chord5	Thickness5
0.13	0.14	0.160036	0.175114	0.184335	0.185283	0.174	0.144001	0.11	0.0008
Chord6	Thickness6	Chord7	Thickness7	Chord8	Thickness8	Chord9	Thickness9	Chord10	Thickness10
0.03998	0.031706	0.018	0.016645	0.013051	0.010043	0.007164	0.006201	0.003	1.00E-05

Table 21
Performance of the obtained optimal design.

Name	Value
J	0.88235
K_T	0.30382
K_Q	0.06146
Effy	0.6942
AdEffy	0.82919
C_T	0.99374
C_Q	0.40205
C_P	1.4315

The convergence of MFO when solving this problem is illustrated in Fig. 20. Note that the convergence between iterations 214 to 500 is just shown since there was no feasible solutions during iterations 1–213.

The 2D images of the obtained optimal airfoils along five cross sections of the blade are illustrated in Fig. 21.

To see how the MFO algorithm found the optimal shape for the propeller, Fig. 22 illustrates the initial infeasible random design created in the first iteration and the final feasible optimal design obtained in the last iteration. It may be observed that how the proposed algorithm effectively and efficiently found a smooth shape for the blades to maximize the overall efficiency of the propeller.

As discussed above, the propeller design is a CFD problem with dominated infeasible regions. The results of this section highly demonstrate the applicability of the proposed algorithm in solving challenging real problems with unknown and constrained search spaces. Therefore, it can be stated that the MFO algorithm has merits in solving similar challenging problems.

7. Conclusion

In this paper the transverse orientation of moths was modelled to propose a new stochastic population-based algorithm. In fact, the spiral convergence toward artificial lights was the main inspiration of the MFO algorithm. The algorithm was equipped with several operators to explore and exploit the search spaces. In order to benchmark the performance of MFO, three phases of test were conducted: test functions, classical engineering problems, and a CFD problem. In addition the results were compared to a wide range of algorithms for verification. In the first test phase, 19 test functions were employed to benchmark the performance of MFO from different perspectives. It was observed that the MFO algorithm is able show high and competitive exploration in multi-modal functions and exploitation in unimodal functions. Moreover, the results of the composite test functions prove the MFO balances exploration and exploitation properly. The first test phase also considered the observation and investigation of MFO's convergence.

In the second test phase, seven classical engineering test problem were employed to further investigate the effectiveness of MFO in practice. The problems were welded beam design, gear train design, three-bar truss design, pressure vessel design, cantilever design, I-beam design, and tension/compression spring , 15-bar truss, and 52-bar truss design problems. The results proved that the MFO algorithm can also be effective in solving challenging problems with unknown search spaces. The results of this algorithm were compared to a variety of algorithms in the literature. The second test phase also considered the constrained and discrete problems to observe the performance of the MFO algorithm in solving problems with different characteristics. Eventually, the last test phase demonstrated the application of the MFO algorithm in the field of propeller design. The employed problem was a highly constrained and expensive problem. However, the MFO algorithm easily managed to optimize the structure of the employed propeller and improve its efficiency.

According to this comprehensive comparative study, the following conclusion remarks can be made:

- Procedure of updating positions allows obtaining neighbouring solutions around the flames, a mechanism for mostly promoting exploitation.
- Adaptive convergence constant (r) towards the flame causes accelerated exploitation around the flames over the course of iterations.
- Local optima avoidance is high since MFO employs a population of moths to perform optimization.
- Assigning each moth a flame increases exploration of the search space and decreases the probability of local optima stagnation.

- Decreasing the number of flames balances exploration and exploitation of the search space.
- Considering recent best solution obtained so far as the flames save the promising solutions as the guides for moths.
- The best solutions are saved so they never get lost.
- The convergence of the MFO algorithm is guaranteed because the moths always tend to update their positions with respect to flames who are the most promising solutions obtained so far over the course of iterations.
- The MFO algorithm is able to solve real challenging problems with unknown and constrained search spaces.
- According to the NFL theorem, there is no optimization algorithm for solving all optimization problems. Since the MFO algorithm was able to outperform other algorithms on the majority of test cases in this study, it can be considered as an alternate optimizer for solving optimization problems among the current famous algorithms.

For future works, several research directions can be recommended. Firstly, the effect of different spirals in improving the performance of the MFO is worth researching. Secondly, the binary version of the MFO algorithm can be another interesting future work. Last but not least, the proposal of specific operators to solve multi-objective algorithms using MFO is recommended.

Appendix A

I – Welded beam design problem

$$\begin{aligned}
\text{Consider}\quad & \vec{x} = [x_1\ x_2\ x_3\ x_4] = [h\ l\ t\ b],\\
\text{Minimize}\quad & f(\vec{x}) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14.0 + x_2),\\
\text{Subject to}\quad & g_1(\vec{x}) = \tau(\vec{x}) - \tau_{\max} \leqslant 0,\\
& g_2(\vec{x}) = \sigma(\vec{x}) - \sigma_{\max} \leqslant 0,\\
& g_3(\vec{x}) = x_1 - x_4 \leqslant 0,\\
& g_4(\vec{x}) = 1.10471x_1^2 + 0.04811x_3x_4(14.0 + x_2) - 5.0 \leqslant 0,\\
& g_5(\vec{x}) = 0.125 - x_1 \leqslant 0,\\
& g_6(\vec{x}) = \delta(\vec{x}) - \delta_{\max} \leqslant 0,\\
& g_7(\vec{x}) = P - P_c(\vec{x}) \leqslant 0,\\
\text{Variable range}\quad & 0.1 \leqslant x_1 \leqslant 2,\\
& 0.1 \leqslant x_2 \leqslant 10,\\
& 0.1 \leqslant x_3 \leqslant 10,\\
& 0.1 \leqslant x_4 \leqslant 2,\\
\text{where}\quad & \tau(\vec{x}) = \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2},\\
& \tau' = \frac{P}{\sqrt{2}x_1x_2},\ \tau'' = \frac{MR}{J},\ M = P\left(L + \frac{x_2}{2}\right),\\
& R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2},\\
& J = 2\left\{\sqrt{2}x_1x_2\left[\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\},\\
& \sigma(\vec{x}) = \frac{6PL}{x_4x_3^2},\ \delta(\vec{x}) = \frac{6PL^3}{Ex_3^2x_4},\\
& P_c(\vec{x}) = \frac{4.013E\sqrt{\frac{x_3^2x_4^6}{36}}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right),\\
& P = 6000\,\text{lb},\ L = 14\,\text{in.},\ E = 30 \times 1^6\,\text{psi},\ G = 12 \times 10^6\,\text{psi},\\
& \tau_{\max} = 13{,}600\,\text{psi},\ \sigma_{\max} = 30{,}000\,\text{psi},\ \delta_{\max} = 0.25\,\text{in.}
\end{aligned}$$

II – Gear train design problem

$$\begin{aligned}
\text{Consider}\quad & \vec{x} = [x_1\ x_2\ x_3\ x_4] = [n_A\ n_B\ n_C\ n_D],\\
\text{Minimize}\quad & f(\vec{x}) = \left(\frac{1}{6.931} - \frac{x_3x_2}{x_1x_4}\right)^2,\\
\text{Subject to}\quad & 12 \leqslant x_1,\ x_2,\ x_3,\ x_4 \leqslant 60.
\end{aligned}$$

III – Three-bar truss design problem

$$\begin{aligned}
\text{Consider}\quad & \vec{x} = [x_1\ x_2] = [A_1\ A_2],\\
\text{Minimize}\quad & f(\vec{x}) = \left(2\sqrt{2}x_1 + x_2\right) * l,\\
\text{Subject to}\quad & g_1(\vec{x}) = \frac{\sqrt{2}x_1 + x_2}{\sqrt{2}x_1^2 + 2x_1x_2}P - \sigma \leqslant 0,\\
& g_2(\vec{x}) = \frac{x_2}{\sqrt{2}x_1^2 + 2x_1x_2}P - \sigma \leqslant 0,\\
& g_3(\vec{x}) = \frac{1}{\sqrt{2}x_2 + x_1}P - \sigma \leqslant 0,\\
\text{Variable range}\quad & 0 \leqslant x_1,\ x_2 \leqslant 1,\\
\text{where}\quad & l = 100\,\text{cm},\ P = 2\,\text{KN/cm}^2,\quad \sigma = 2\,\text{KN/cm}^2.
\end{aligned}$$

IV – Pressure vessel design problem

$$\begin{aligned}
\text{Consider}\quad & \vec{x} = [x_1\ x_2\ x_3\ x_4] = [T_s\ T_h\ R\ L],\\
\text{Minimize}\quad & f(\vec{x}) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4\\
& \qquad + 19.84x_1^2x_3,\\
\text{Subject to}\quad & g_1(\vec{x}) = -x_1 + 0.0193x_3 \leqslant 0,\\
& g_2(\vec{x}) = -x_3 + 0.00954x_3 \leqslant 0,\\
& g_3(\vec{x}) = -\pi x_3^2x_4 - \frac{4}{3}\pi x_3^3 + 1{,}296{,}000 \leqslant 0,\\
& g_4(\vec{x}) = x_4 - 240 \leqslant 0,\\
\text{Variable range}\quad & 0 \leqslant x_1 \leqslant 99,\\
& 0 \leqslant x_2 \leqslant 99,\\
& 10 \leqslant x_3 \leqslant 200,\\
& 10 \leqslant x_4 \leqslant 200.
\end{aligned}$$

V – Cantilever design

$$\begin{aligned}
\text{Consider}\quad & \vec{x} = [x_1\ x_2\ x_3\ x_4\ x_5]\\
\text{Minimize}\quad & f(\vec{x}) = 0.6224(x_1 + x_2 + x_3 + x_4 + x_5),\\
\text{Subject to}\quad & g(\vec{x}) = \frac{61}{x_1^3} + \frac{27}{x_2^3} + \frac{19}{x_3^3} + \frac{7}{x_4^3} + \frac{1}{x_5^3} - 1 \leqslant 0,\\
\text{Variable range}\quad & 0.01 \leqslant x_1,\ x_2,\ x_3,\ x_4,\ x_5 \leqslant 100.
\end{aligned}$$

VI – I-beam design

$$\begin{aligned}
\text{Consider}\quad & \vec{x} = [x_1\ x_2\ x_3\ x_4\ x_5] = [b\ h\ t_w\ t_f],\\
\text{Minimize}\quad & f(\vec{x}) = \frac{5000}{\frac{t_w(h - 2t_f)^3}{12} + \frac{bt_f^3}{6} + 2bt_f\left(\frac{h - t_f}{2}\right)^2},\\
\text{Subject to}\quad & g(\vec{x}) = 2bt_w + t_w(h - 2t_f) \leqslant 0,\\
\text{Variable range}\quad & 10 \leqslant x_1 \leqslant 50,\\
& 10 \leqslant x_2 \leqslant 80,\\
& 0.9 \leqslant x_3 \leqslant 5,\\
& 0.9 \leqslant x_4 \leqslant 5.
\end{aligned}$$