Engineering Applications of Artificial Intelligence 80 (2019) 20–34

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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

The Sailfish Optimizer: A novel nature-inspired metaheuristic algorithm for solving constrained engineering optimization problems☆

S. Shadravan [a], H.R. Naji [b,*], V.K. Bardsiri [a]

[a] Department of Computer Engineering, Kerman Branch, Islamic Azad University, Kerman, Iran
[b] Department of Computer Engineering, Graduate University of Advanced Technology, Kerman, Iran

ARTICLE INFO

Keywords:
Swarm optimization
Metaheuristic algorithm
Sailfish optimization algorithm
Constrained engineering optimization

ABSTRACT

Nature-inspired optimization algorithms, especially swarm based algorithms (SAs), solve many scientific and engineering problems due to their flexibility and simplicity. These algorithms are applicable for optimization problems without structural modifications. This work presents a novel nature-inspired metaheuristic optimization algorithm, called SailFish Optimizer (SFO), which is inspired by a group of hunting sailfish. This method consists of two tips of populations, sailfish population for intensification of the search around the best so far and sardines population for diversification of the search space. The SFO algorithm is evaluated on 20 well-known unimodal and multimodal mathematical functions to test different characteristics of the algorithm. In addition, SFO is compared with the six state-of-art metaheuristic algorithms in low and high dimensions. It also indicates competitive results for improvement of exploration and exploitation phases, avoidance of local optima, and high speed convergence especially on large-scale global optimization. The SFO algorithm outperforms the best algorithms in the literature on the majority of the test functions and it shows the statistically significant difference among other algorithms. Moreover, the SFO algorithm shows significantly great results for non-convex, non-separable and scalable test functions. Eventually, the promising results on five real world optimization problems indicate that the SFO is applicable for problem solving with constrained and unknown search spaces.

1. Introduction

In real-world, resources are always limited and the optimization of available resources is crucially important. Optimization can be used essentially in a variety of fields from engineering design to economics or holiday planning to Internet routing. Metaheuristic algorithms can provide appropriate technique to solve optimization problems through mathematical modeling of social political evolution. These algorithms are using methods that find solutions close to the optimum with an acceptable cost. Since random mechanisms play the key role in creating their structure, metaheuristic algorithms are known as imprecise methods for solving complex optimization problems.

In the last several decades, metaheuristic algorithms are applied to many applications. The main reason behind the success of metaheuristic algorithms is that they use commonly shared information among multiple agents. Also several factors may help these algorithms in creating the highest quality results such as self-organization, coevolution, and learning. All of the metaheuristics algorithms are not successful and a few techniques that have potential for solving real-world problems can be very efficient (Yang, 2018).

Recently, many metaheuristic algorithms have been proposed. Based on the source of inspiration, these algorithms is classified into four main categories: swarm based, physics based, human based and evolution based. Vast variety of metaheuristic algorithms are introduced in Swarm Intelligence (SI). Swarm based method imitates the population of homogeneous agents interacting with each other and with their environment. Colonies of ants, flocks of birds, and schools of fish are examples of such swarm intelligence. Physics based method is another category in this classification that mimics the physical rules in the world. The third category of metaheuristic algorithms includes human based techniques that are inspired by human behaviors and the laws that is existed among them. The last category is evolution based methods that are inspired by the laws of biological evolution such as selection, mutation, reproduction, and recombination.

In fact, the metaheuristic algorithms have a great ability for exploring and exploiting search experiences by using different methods. Of great importance hereby is that a metaheuristic algorithm will be successful if it can provide the dynamic balance between exploration and exploitation on a given optimization problem. In exploration phase, the movement of search agents should be randomized as much as

☆ No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.engappai.2019.01.001.

* Corresponding author.

E-mail addresses: shadravan@bardsiriau.ac.ir (S. Shadravan), h.naji@kgut.ac.ir (H.R. Naji), kvahid2@live.utm.my (V.K. Bardsiri).

https://doi.org/10.1016/j.engappai.2019.01.001
Received 10 July 2018; Received in revised form 2 January 2019; Accepted 3 January 2019
Available online xxxx
0952-1976/© 2019 Published by Elsevier Ltd.

possible. Whereas the exploitation phase investigates the promising area(s) in detail. Generally speaking, the main distinction between the existing metaheuristic algorithms is how to balance exploration and exploitation phases.

Some general aspects which may be affected by reputation of metaheuristic algorithms are as follow (see Fig. 1): First, trajectory methods vs. discontinuous methods. A main difference among these methods is that discontinuous method allows large jumps in the neighborhood graph, but trajectory method traces only one single trajectory on the neighborhood graph. Trajectory methods have the ability to avoid from local optimum by moving worse solutions. The Tabu Search algorithm (TS) (Glover, 1989) can be a good example for trajectory method. In this algorithm, the local searches (neighborhood) take a potential solution for a problem and check its immediate neighbors with hope to find an improved solution. Using this strategy, the TS algorithm can trace single trajectory to find an optimum point. Second, population based vs. single point search. A population of agents is used for population based search. This method is suitable way for improving the exploration phase. Certainly, how to manipulate a population affects the performance of the algorithm. On the contrary, one single agent is manipulated over the course of iteration in single point search during optimization. Since ant colony optimization algorithm (ACO) (Dorigo et al., 2006) uses a swarm of ants for recording the position of search agents for solving computational problems, it can be a good example for the population based strategy. Third, memory usage vs. memoryless methods. These are another methods in metaheuristic algorithms to influence the future search direction. The algorithms that use the memory functions to achieve this effectiveness are memory usage and the algorithms do not use the memory function are memoryless usage. One of the good examples of memory usage method is particle swarm optimization (PSO) (Kennedy and Eberhart, 1995) because this algorithm can keep the best local and global solutions in memory to calculate the future solution during the optimization. Fourth, multiple neighborhood vs. one neighborhood. Some algorithms use more than one neighborhood structure. They start the local search to escape from local optima with N neighborhood and the kick-move is applied to approach the global optimum in the search space. For example, mutation in genetic algorithm (Holland, 1992) acts as a kick-move and it can interpret like a jump between neighborhoods during local search. In Fig. 1, several examples for each aspect previously mentioned are shown.

Various algorithms are applied in different areas by many researchers. Numerous algorithms and their various applications could not offer a particular algorithm to solve all the optimization problems. Some algorithms are beneficial for several specific problems but are futile for another kind of problems (Wolpert and Macready, 1997). Therefore, researchers attempt to create new optimization techniques for solving a wider range of unsolved problems.

This paper describes a novel metaheuristic optimization algorithm (namely, SailFish Optimizer, SFO). The SFO algorithm mimics the sailfish group hunting that alternates their attacks on schooling sardine prey. To the knowledge of the present authors, there is no previous study on this subject in the optimization literature. The current study has several major differences with other methods that recently published. Firstly, SFO is employed two groups of prey and predator populations to simulate the group hunting behavior strategy. Secondly, the proposed algorithm uses the alternation of attacks to break down the collective defense of grouping prey. Thirdly, the prey movements can be updated over the search space, and hunter is allowed to catch the appropriate prey to become fitter than the past. The efficiency of the SFO algorithm is evaluated by solving 23 mathematical optimization functions. The rest of this paper is organized as follows:

Section 2 describes the sailfish optimization algorithm. In Section 3, experimental results of the test functions are presented and discussed. Several directions for future research concludes in Section 4.

2. Sailfish optimizer

The main inspiration of SFO algorithm will be described in this section. Then, the proposed algorithm and mathematical model are discussed in details.

2.1. Inspiration

One of the interesting example of social behavior in groups of arthropods, fishes, birds, and mammals is group hunting. In group hunting, predators do not need much effort to kill the prey compared with when hunting alone. In the simplest form of group hunting, predators attempt to kill the prey with little or no coordination of attack, while in the complex form of group hunting, predators use the specific roles to herd and catch the prey (Bailey et al., 2013).

One of the complex group hunting strategies is the alternation of attacks. This strategy provides the opportunity for the hunter to save the energy, while other predators are injuring the prey. One example of this kind of strategy is group hunting sailfish (*Istiophorus platypterus*) that alternate their attacks on schooling sardine prey (*Sardinella aurita*) (Domenici et al., 2014; Marras et al., 2015). Fig. 2 shown each behavioral state of sailfish's group hunting.

Sailfish is the fastest fish in the ocean that can reach maximum *speeds* of around 100 km/h. They hunt in groups, driving schools of smaller fish, such as sardines, toward the surface (Fig. 2a, b). The maneuverability and acceleration of the sardines during the attack are very challenging for sailfish (Fig. 2c). The sailfish either makes a slashing motion with its rostrum, injuring several sardines, or it taps a single sardine and destabilizing it (Fig. 2d). Because the sailfish has one of the highest accelerations ever recorded in an aquatic vertebrate, sardines cannot swim fast enough to avoid the tip of the sailfish's rostrum and they are unable to do anything in response to this group hunting. Given the observational behavior of sardines shown injured sardines will be separated from the prey school and could not move with the school, so they will quickly capture by the sailfish (Fig. 2e) (Herbert-Read James et al., 2016).

Most sailfish attacks do not lead to the death of sardines and only a few percent of sardines are directly captured. But with frequent attacks of sailfish, more and more sardines are hurt. This type of hunt is more typical of animals that hunt in packs, such as wolves. However, these groups of sailfish regularly break up and reform with new members.

A sailfish keeps its large dorsal fin and pelvic fins erect during an attack, probably to keep its body stable (Fig. 2f). Also, they change their body color with the normally bluish-silver lateral sides darkening to almost black just before beginning an attack. The reason for color changing is not very clear, but it should be some kind of communication between sailfish (Herbert-Read James et al., 2016). It is possible to avoid injuring by a compatriot, sailfish use the changes in their body to signal who goes first.

The main inspiration of the SFO algorithm is based on the attack-alternation strategy of sailfish's group hunting. In the next subsection the natural behaviors of sailfish and sardines are modeled mathematically and then an optimization algorithm based on this mathematical model is established.

2.2. SFO: The proposed algorithm

2.2.1. Initialization

The SFO is a population based metaheuristic algorithm. In this algorithm, it is assumed that the sailfish are candidate solutions and the problem's variables are the position of sailfish in the search space. Accordingly, the population over the solution space is randomly generated. The sailfish can search in one, two, three or hyper dimensional space with their variable position vectors. In an d-dimensional search space, the ith member at the kth searching bout has a current position $SF_{i,k} \in \mathcal{R}(i = 1, 2, \ldots, m)$. Matrix SF has been considered to save the

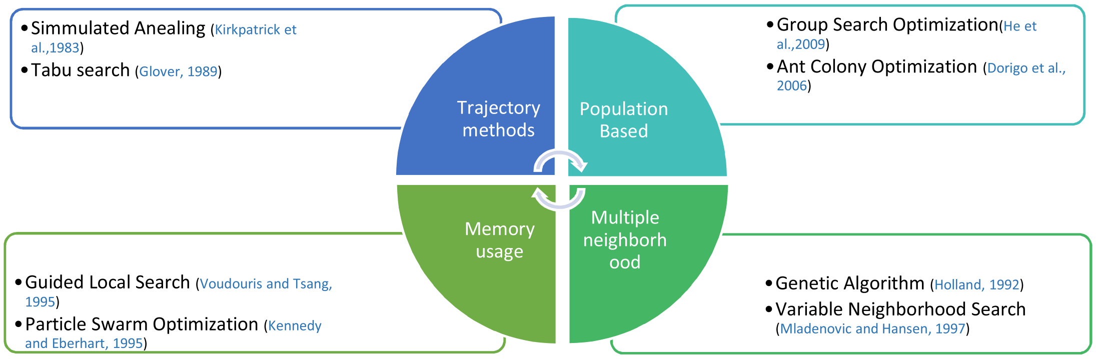
• Simmulated Anealing (Kirkpatrick et al.,1983)
• Tabu search (Glover, 1989)
Trajectory methods
Population Based
• Group Search Optimization(He et al.,2009)
• Ant Colony Optimization (Dorigo et al., 2006)
Memory usage
Multiple neighborhood
• Guided Local Search (Voudouris and Tsang, 1995)
• Particle Swarm Optimization (Kennedy and Eberhart, 1995)
• Genetic Algorithm (Holland, 1992)
• Variable Neighborhood Search (Mladenovic and Hansen, 1997)

Fig. 1. Four effective aspects on reputation of metaheuristic algorithms.

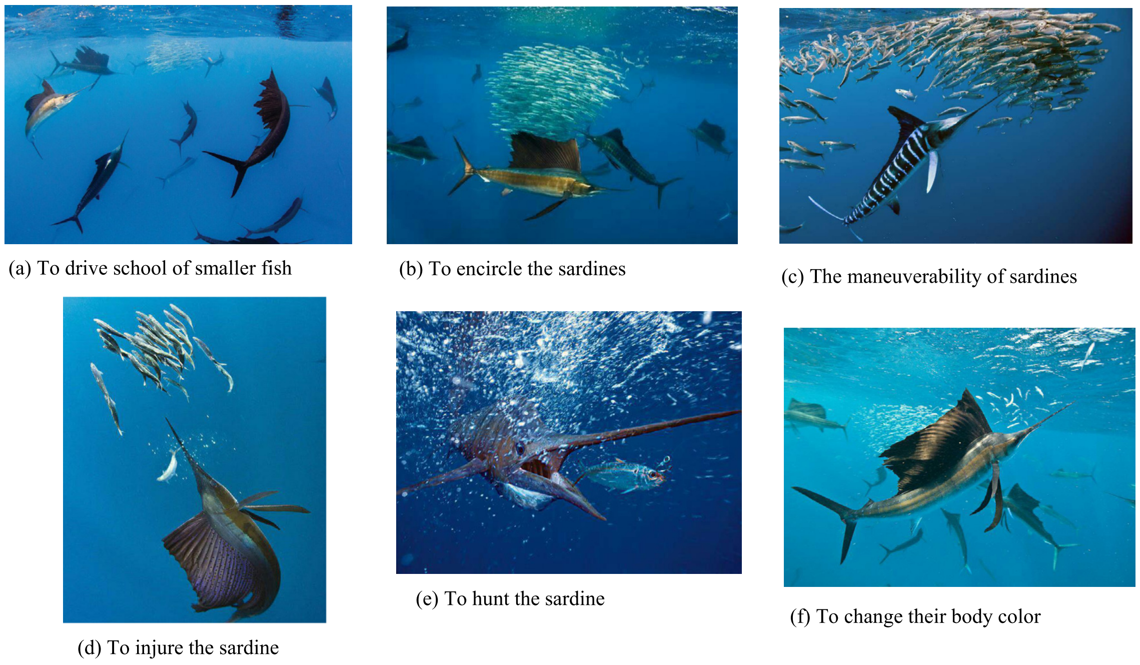
(a) To drive school of smaller fish
(b) To encircle the sardines
(c) The maneuverability of sardines
(d) To injure the sardine
(e) To hunt the sardine
(f) To change their body color

Fig. 2. Behavioral state of sailfish's group hunting.

position of all sailfish. So these positions show the variables of all solutions during optimization.

$$SF_{position} = \begin{bmatrix} SF_{1,1} & SF_{1,2} & \cdots & SF_{1,d} \\ SF_{2,1} & SF_{2,2} & \cdots & SF_{2,d} \\ \vdots & \vdots & \vdots & \vdots \\ SF_{m,1} & SF_{m,2} & \cdots & SF_{m,d} \end{bmatrix} \tag{1}$$

where m shows the number of sailfish and d indicates the number of variables and $SF_{i,j}$ shows the value of the jth dimension *of* ith sailfish. In addition, the fitness of each sailfish is computed by calculation the fitness function as follows:

$$fitness\ value\ of\ sailfish = f\,(sailfish) = f(SF_1, SF_2, \ldots, SF_m) \tag{2}$$

To evaluate each sailfish, the following matrix shows the fitness value for all solutions:

$$SF_{Fitness} = \begin{bmatrix} f(SF_{1,1} & SF_{1,2} & \cdots & SF_{1,d}) \\ f(SF_{2,1} & SF_{2,2} & \cdots & SF_{2,d}) \\ \vdots & \vdots & \vdots & \vdots \\ f(SF_{m,1} & SF_{m,2} & \cdots & SF_{m,d}) \end{bmatrix} = \begin{bmatrix} F_{SF_1} \\ F_{SF_2} \\ \vdots \\ F_{SF_m} \end{bmatrix} \tag{3}$$

where m is the number of sailfish, $SF_{i,j}$ is the value of the jth dimension *of* ith sailfish, f calculates the fitness function, and $SF_{Fitness}$ saves the fitness value that returns the value of fitness or objective function for each sailfish. The first row of $SF_{position}$ matrix is sent to the fitness function and the output indicates the fitness value of the corresponding sailfish in the $SF_{Fitness}$ matrix.

The school of sardines is another significant incorporator in SFO algorithm. It is assume that the group of sardines is also swimming in the search space. So the position of sardines and their fitness values are utilized as follows:

$$S_{position} = \begin{bmatrix} S_{1,1} & S_{1,2} & \cdots & S_{1,d} \\ S_{2,1} & S_{2,2} & \cdots & S_{2,d} \\ \vdots & \vdots & \vdots & \vdots \\ S_{n,1} & S_{n,2} & \cdots & S_{n,d} \end{bmatrix} \tag{4}$$

where n is the number of sardines and $S_{i,j}$ is the value of the jth dimension *of* ith sardine, $S_{position}$ matrix indicates the position of all sardines.

$$S_{Fitness} = \begin{bmatrix} f(S_{1,1} & S_{1,2} & \cdots & S_{1,d}) \\ f(S_{2,1} & S_{2,2} & \cdots & S_{2,d}) \\ \vdots & \vdots & \vdots & \vdots \\ f(S_{n,1} & S_{n,2} & \cdots & S_{n,d}) \end{bmatrix} = \begin{bmatrix} F_{S_1} \\ F_{S_2} \\ \vdots \\ F_{S_n} \end{bmatrix} \tag{5}$$

where n is the number of sardines, $S_{i,j}$ shows the value of the jth dimension *of* ith sardine, f is the objective function and $S_{Fitness}$ saves the fitness value of each sardine. It is notable that sailfish and sardines are corresponding factors to find the solutions. In this algorithm, the sailfish are the main factor that is scattered in the search space and sardines can cooperate to find the best position in this area. In fact, sardine can be eaten by sailfish when searching the search space and sailfish updates its position in case of finding a better solution that obtained so far.

2.2.2. *Elitism*

Occasionally good solutions can be lost when updating the position of search agents and these positions may be weaker than the old positions unless the elitist selection is employed. Elitism involves copying the unchanged fittest solution(s) into the next generation. In SFO algorithm also, the best position of sailfish is saved in each iteration and considered as an elite. The elite sailfish is the fittest sailfish that is obtained so far and it should be able to affect the maneuverability and acceleration of sardines during the attack. In addition, as previously mentioned, sardines will be injured by slashing motion with sailfish's rostrum during group hunting. Therefore, the position of injured sardine in each iteration is also saved and this sardine will be selected as the best target for collaborative hunting by the sailfish. The position of elite sailfish and injured sardine that have the highest fitness at ith iteration are called $X^i_{elite\_SF}$ and $X^i_{injured\_S}$ respectively. These positions can have a dramatic impact on performance of SFO and they prevent the waste of time for re-discovering previously discarded solutions.

2.2.3. *Attack-alternation strategy*

In fact, sailfish mostly attack the prey school when none of their compatriots is attacking. In the other words, sailfish can promote the rate of success in hunting with the temporally coordinated attack (Herbert-Read James et al., 2016). Sailfish chase and herd their prey. The herding behavior of sailfish adjust their position according to the location of the other hunters around the prey school without direct coordination between them. The SFO algorithm demonstrates sailfish's attack-alternation strategy while hunting in groups. It is observed in Fig. 3 that search agents provide the exploration phase and it consists of searching a large section of the search space to find the promising solutions that are yet to be refined. As shown in Fig. 3, sailfish do not attack just from up to down or from right to left and vice versa. They can attack in all directions and within a shrinking circle. Consequently, sailfish update their position within a sphere around the best solution.

In the SFO algorithm, at the ith iteration, the new position of sailfish $X^i_{new\_SF}$ updates as follows:

$$X^i_{new\_SF} = X^i_{elite\_SF} - \lambda_i \times \left(rand\,(0,1) \times \left(\frac{X^i_{elite_{SF}} + X^i_{injured_S}}{2} \right) - X^i_{old_{SF}} \right) \tag{6}$$

where $X^i_{elite\_SF}$ is the position of elite sailfish formed until now, $X^i_{injured\_S}$ determines the best position of injured sardine formed so far, $X^i_{old\_SF}$ is the current position of sailfish, $rand(0,1)$ is a random number between 0 and 1, and λ_i is a coefficient at the ith iteration that generated as follows:

$$\lambda_i = 2 \times rand\,(0,1) \times PD - PD \tag{7}$$

where PD is prey density that shows the number of prey at each iteration. Because the number of prey will decrease during group hunting by sailfish, the PD parameter is a significant parameter for updating the position of sailfish around the prey school. The adaptive formula for this parameter is as follows:

$$PD = 1 - \left(\frac{N_{SF}}{N_{SF} + N_S} \right) \tag{8}$$

where N_{SF} and N_S are respectively the number of sailfish and the number of sardines in each cycle of the algorithm. In addition, due to the initial number of sardines is mostly larger than sailfish, N_{SF} is defined by $N_S \times PP$ Where PP represents the percentage of sardine population that form the initial sailfish population.

Fig. 3 demonstrates the rationale behind Eq. (6) for a 2D problem. According to the average distance between position of the current best sailfish (Elite sailfish) and the current best sardine (Injured sardine), the position of sailfish can be updated over course of iteration. With this strategy, the promising area of search space will be saved. Also, sailfish can achieve different places around the school by adjusting the value of λ. According to Eq. (7), the fluctuation range of λ is between -1 and 1 but it depends on the number of prey. In other words, with increasing PD the amount of λ will be close to -1 or 1 with respect to $rand(0,1)$ in Eq. (7).

The λ parameter will be tended to 1 when $rand\,(0,1) > 0.5$, while it tend to -1 when $rand\,(0,1) < 0.5$ and it will be zero if $rand\,(0,1) = 0.5$. Fluctuation of λ and update position of sailfish can mathematically model divergence of sailfish from each other and convergence of them around the prey schools. This emphasize exploration and search the solution globally.

By doing so, the SFO algorithm assists any sailfish to update its position around the prey school with two way. In the first way, sailfish have the alternative attack to prey school with respect to *elite* sailfish and *injured* sardines such as sailfish 2 and 3 in Fig. 3. In the second way, sailfish occupies empty space around the prey school and simulate encircling the prey like sailfish 1 in Fig. 3. In both ways, sailfish are going to injure more sardines in first stage of hunting and it leads to higher capture success rate at later stages of collaborative hunting.

2.2.4. *Hunting and catching prey*

At the beginning of the group hunting, the complete slaughter of sardines are rarely observed. In 95% of cases, sardines' scales will be removed when the sailfish's bills hits the sardines' bodies (Herbert-Read James et al., 2016). This causes a large number of sardines in the schools having pronounced injuries on their bodies. In (Herbert-Read James et al., 2016), researchers found the positive correlation between the capture success rate and the amount of injuries in the prey school. At the beginning of the hunt, sailfish have more energy to catch prey and also sardines are not more tired and injured. Therefore sardines sustain high escape speed and they have a great ability to maneuver. Gradually, the power of sailfish's attack will be decremented over time of hunting. Because of intense and frequent attacks, the energy stores in prey will also reduce and may have the reduction of ability to detect directional information about the location of sailfish, which it affects the school's escape maneuvers. Eventually, sardines will hit by sailfish's bill, break off from the shoal and will capture quickly.

For mimicking this process, each sardine is obliged to update its position with respect to the current best position of sailfish and power of its attack at each iteration. In the SFO algorithm, at the ith iteration the new position of sardine $X^i_{new\_S}$ may be given as:

$$X^i_{new\_S} = r \times \left(X^i_{elite\_SF} - X^i_{old_S} + AP \right) \tag{9}$$

where $X^i_{elite\_SF}$ is the best position of elite sailfish formed so far, $X^i_{old\_S}$ is the current position of sardine, r is a random numbers between 0 and 1 and AP shows the amount of sailfish's *Attack Power* at the each iteration that is generated as follows:

$$AP = A \times (1 - (2 \times Itr \times \varepsilon)) \tag{10}$$

where A and ε are coefficients for decreasing the value of power attack linearly from A to 0. To see the effects of using Eqs. (9) and (10), some of the possible locations of sardines after slashing the prey school are illustrated in Fig. 4. When a sailfish attacks, sardines escape to different places immediately. As it can be seen in Fig. 4, sardines update their position for confusing the predator and reducing the risk of being found according to the amount of r and AP parameters. The random value r can regulate the dispersion rate of prey around the predator in the space search (Fig. 4a). Eq. (9) is one of the main components of the proposed algorithm because it indicates how the sardines update their positions around the sailfish after each attack. This equation allows a sardine to escape from the best sailfish (Elite) which assists to exploit information from their respective neighbors and it can be considered as a robust local

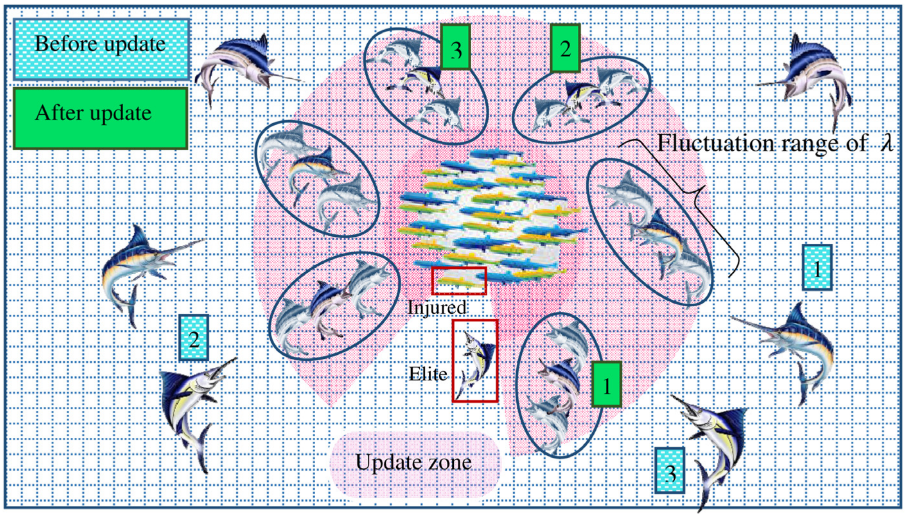

Fig. 3. Swimming sailfish around the prey school in the search space.

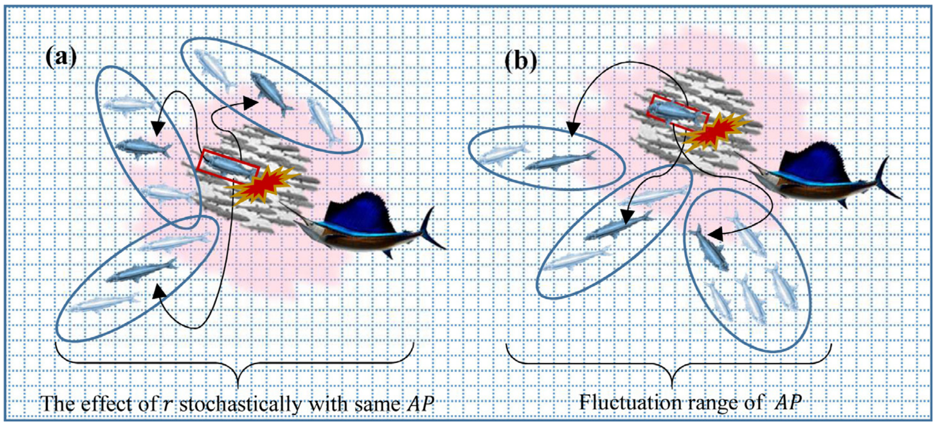
(a)
(b)
The effect of r stochastically with same AP
Fluctuation range of AP

Fig. 4. Slashing the prey school by sailfish and update the position of sardines in the search space.

search. Therefore, the exploration and exploitation of the search space can be balanced with this method.

In addition, the number of sardines that update their positions and the amount of their displacement depends on the power of sailfish attack (AP) (Fig. 4b). As mentioned above the power of sailfish's attack will reduce over time of hunting. In fact, reducing the intensity of sailfish's attack power can adaptively assist the convergence of the search agents. Using AP parameter, the number of sardines that update their position (α) and the number of variables of them (β) can be calculated as follows:

$$\alpha = N_S \times AP \tag{11}$$

$$\beta = d_i \times AP \tag{12}$$

where d_i is the number of variables at ith iteration and N_S is the number of sardines in each cycle of the algorithm. According to AP parameter, if the intensity of sailfish's tap is low ($AP < 0.5$) just α sardines with β variables of sardine will be updated. However, if the intensity of sailfish's tap is high ($AP \geq 0.5$) the position of all sardines will be updated. Basically, AP and r parameters assist SFO to show a more random behavior throughout optimization and they are very helpful for local optima stagnation during all iterations.

In the final stage of hunting, the injured sardine that is break off from the shoal will be captured quickly. In the proposed algorithm, it is assumed that catching prey occurs when the sardine becomes fitter than its corresponding sailfish. In this situation, the position of sailfish substitutes with the latest position of the hunted sardine to increase the chance of hunting new prey. The equation is proposed as follows:

$$X^i_{SF} = X^i_s \quad if \quad f\left(S_i\right) < f\left(SF_i\right) \tag{13}$$

where X^i_S shows the current position of sardine at ith iteration and X^i_{SF} shows the current position of sailfish at ith iteration.

In the SFO algorithm, the position of sailfish and sardines are generated randomly. In every iteration, the position of each sailfish will be updated with respect to an injured sardine and the elite sailfish. Also, the updating position of sardine is then accomplished by selected sardine and elite sailfish depending on the power of sailfish attack. When the process of updating the position of sailfish and sardines finished, they will be evaluated by the objective function. If any of the sardines become fitter than any other sailfish, the sailfish updates its position to this corresponding sardine. Also, the position of elite sailfish and the injured sardine will be updated in each cycle of the algorithm. Eventually, the hunted sardine will be removed from its population. These steps are updated iterative until the end of criterion is satisfied. The pseudocode of the SFO is summarized in Table 1

Some significant points to show the ability of SFO for approximating the global optima point of optimization problems may be noted as follow:

- Random selection of sailfish and sardines guarantee exploration of the search space.
- Since the SFO algorithm uses a sardine population the local optima stagnation become very low.

Table 1
Pseudocode for the SFO algorithm.

```
Initialize the population of sailfish and sardine randomly
Initialize parameters (A=4, $\varepsilon = 0.001$).
Compute the fitness of sailfish and sardines.
Find the best sailfish and sardine and assume that they are as elite
sailfish and injured sardine respectively.
 While the termination conditions is not satisfied
   for each sailfish
     Calculate $\lambda_i$ using Eq. (7).
     Update the position of sailfish using Eqs. (6).
   end for
   Calculate AttackPower using Eq. (10).
     If AttackPower < 0.5
       Calculate $\alpha$ using Eq. (11).
       Calculate $\beta$ using Eq. (12).
       Select a set of sardine base on the value of $\alpha$ and $\beta$
       Update the position of selected sardine by Eqs. (9).
     else
       Update the position of all sardine by the Eqs. (9).
     end if
   Calculate the fitness of all sardine
     If there is a better solution in sardine population
       Replace a sailfish with injured Sardine using Eq. (13).
       Remove the hunted sardine from population
       Update the best sailfish and best sardine
     end if
  end while
Return best sailfish
```

- The proposed encircling strategy provides a hyper-sphere neighborhood around the solutions that guarantee the exploration phase for the search space.
- Maneuverability of sardines around the best agent promotes exploitation when the iteration counter increases.
- Updating the position of sardine causes the diverse movement behaviors for sardines after slashing the prey school that creates the diversity of position explored around the sailfish.
- The gradual decrement of AP parameter provides the dynamic balance between exploration and exploitation.
- The convergence of the SFO algorithm will be guaranteed when the intensity of sailfish's attack power is adaptively decreased at each iteration.
- Since the sailfish relocate to the position of the fitter sardine, the promising areas of search space will be saved during optimization.

The following section investigates the effectiveness of SFO practically.

3. Result and discussion

To evaluate the performance of the SFO algorithm, a set of mathematical functions are selected. Tables 2, 3, and 4 describe unimodal, multi-modal and fixed dimension multimodal benchmark functions where *Dim* is dimension of the function, *Range* indicates the boundary of the function's search space, and f_{min} is the optimum (Suganthan et al., 2005). Also the experiments were carried out on a machine with the following configuration. A CPU with an Intel(R) Core(TM) i7-2630 CPU @ 2.00 GHz and 6.00 GB RAM. All the programs were written and executed in MATLAB 2016. The operating system was Microsoft Windows 10.

The unimodal functions (F1–F6) have one global optimum and no local optima, so they are suitable for benchmarking the exploitation. The SFO outperforms all other competitive algorithms in F2, F3, F4 and F6. It may be due to the proposed exploitation operators previously discussed. In contrary, multi-modal functions (F7–F11) have a massive number of local optima and are suitable to evaluate local optima avoidance and exploration ability of an algorithm. This set of benchmarks are most difficult functions to optimize since the number of local minima increases as the function dimension increases. Eventually, fixed dimension multimodal functions (F12–F20) have also more than one optimum but the number of local minima is less than the second category and their dimension is low (*Dim* < 6).

Table 2
Unimodal benchmark functions.

Function	Dim	Range	$f_{\min}$
$f_1(x) = \sum_{i=1}^{n} x_i^2$	30	[−100,100]	0
$f_2(x) = \sum_{i=1}^{n} \left(\sum_{j-1}^{i} x_j\right)^2$	30	[−100,100]	0
$f_3(x) = \max_i \left\{\lvert x_i \rvert, 1 \le i \le n\right\}$	30	[−100,100]	0
$f_4(x) = \sum_{i=1}^{n-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(x_i - 1\right)^2\right]$	30	[−30,30]	0
$f_5(x) = \sum_{i=1}^{n} \left(\left[x_i + 0.5\right]\right)^2$	30	[−100,100]	0
$f_6(x) = \sum_{i=1}^{n} i x_i^4 + random[0,1)$	30	[−1.28,1.28]	0

For verification of the performance of the SFO, several well-known algorithms are employed: Grew Wolf Optimizer (GWO) (Mirjalili et al., 2014), Satin Bowerbird Optimizer (SBO) (Moosavi and Khatibi Bardsiri, 2017), Ant Lion Optimizer ALO (Mirjalili, 2015a,b), Genetic algorithms (GA) (Holland, 1992), Particle swarm optimization (PSO) (Kennedy and Eberhart, 1995) and Salp Swarm Algorithm (Mirjalili et al., 2017) (SSA). In order to generate meaningful statistical results, each algorithm is executed on test functions for 30 times. Every test function is solved with 30 candidate solutions on the 500 iterations to provide a fair comparison. Also, it should be noted that the number of sailfish or any other search agents can be selected experimentally. The large number of artificial sailfish determine the higher probability of the global optima. But, it is obvious that 30 agents are a reasonable number for solving optimization problems. To adjust some of the control parameters of each algorithm, the values in the latest version (source code) are employed to ensure the best performance as follows:

1. SFO: Parameter values of A and ε are considered, 4 and 0.001, respectively. Maximum iteration = 500 and initial population = 30.
2. GWO: Maximum iteration = 500 and initial population = 30.
3. PSO: $c_1 = 2$, $c_2 = 2$, $w = 1$. Maximum iteration = 500 and initial population = 30.
4. ALO: Maximum iteration = 500 and initial population = 30.
5. SBO: Parameter values of a and z are considered, 0.94 and 0.02, respectively. The mutation probability is 0.05. Maximum iteration = 500 and initial population = 30.
6. SSA: Maximum iteration = 500 and initial population = 30.
7. GA: The single point crossover with a crossover probability of 0.9, mutation probability of 0.01. Maximum iteration = 500 and initial population = 30.

Since metaheuristic algorithms are stochastic optimization techniques, mean and standard deviation are calculated for evaluating the performance of algorithms. Because of the stochastic nature of the metaheuristic algorithms, t-test analysis also is employed. A t-test uses the statistical examination to analyze two sets of populations and it can indicate that the result have statistically significant difference or not. This test returns a p-value parameter that should be less than 0.05 to prove the statistical difference.

3.1. Assessment of exploitation phase

The test functions that have only one global optimum are unimodal (F1–F6). These functions are suitable for evaluating the exploitation phase of metaheuristic algorithms. Table 6 shows that the SFO is very competitive compared with other metaheuristic algorithms. As it can be seen in the convergence curve in Fig. 5, the SFO algorithm can converge rapidly toward the optimum for unimodal functions. This indicates

Table 3
Multimodal benchmark functions.

Function	Dim	Range	$f_{\min}$
$F_7(x) = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	30	[−500,500]	$-418 \times Dim$
$F_8(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$	30	[−5.12,5.12]	0
$F_9(x) = -20\,exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$	30	[−32,32]	0
$F_{10}(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	30	[−600,600]	0
$F_{11}(x) = 0.1\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{n}(x_i - 1)^2\left[1 + \sin^2(3\pi x_1 + 1) +\right] + (x_n - 1)^2\left[1 + \sin(2\pi x_n)\right]\right\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	30	[−50,50]	0

Table 4
Fixed dimension multimodal benchmark functions.

Function	Dim	Range	$f_{\min}$
$f_{12}(x) = \sum_{i=1}^{11}\left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2$	4	[−5,5]	0.00030
$f_{13}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1 x_2 - 4x_2^2 + 4x_2^4$	2	[−5,5]	−1.0316
$f_{14}(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$	2	[−5,5]	0.398
$f_{15}(x) = \left[1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\right] \times \left[30 + (2x_1 - 3x_2)^2 \times (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\right]$	2	[−2,2]	3
$f_{16}(x) = -\sum_{i=1}^{4} c_i\,exp\left(-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2\right)$	3	[0,1]	−3.86
$f_{17}(x) = -\sum_{i=1}^{4} c_i\,exp\left(-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\right)$	6	[0,1]	−3.32
$f_{18}(x) = -\sum_{i=1}^{5}\left[(X - a_i)(X - a_i)^r + c_i\right]^{-1}$	4	[0,10]	−10.1532
$f_{19}(x) = -\sum_{i=1}^{7}\left[(X - a_i)(X - a_i)^r + c_i\right]^{-1}$	4	[0,10]	−10.4028
$f_{20}(x) = -\sum_{i=1}^{10}\left[(X - a_i)(X - a_i)^r + c_i\right]^{-1}$	4	[0,10]	−10.5363

that the SFO has been accurately exploited during optimization. The results in the Table 6 present the SFO is the most efficient optimizer for functions F2, F3, F4, and F6 and it can hence provide very good exploitation than the other algorithms. In addition, the p-values in Table 5 are much less than 0.05 and it shows the superiority of SFO in the statistical difference. Table 6 also presents the CPU time in seconds for each unimodal algorithms.

3.2. Assessment of exploration phase

The multimodal functions contain many local minima points and these points will be increased according to the number of variables. Therefore, these functions are very suitable for evaluating the capability of an algorithm in the exploration phase. The CPU time in seconds for each multimodal functions are shown in Table 6. The results presented in Table 6 for multimodal functions indicate that SFO has significantly better performance on F8, F10, and F11. Also, the p-values in Table 5 report that the SFO algorithm shows significantly better results for most multimodal functions. In addition, for fixed-dimension multimodal functions, it can be seen from Table 6 that the SFO algorithm has outperformed the other comparative algorithms for functions F12, F18, F19, and F20 and it also has the same result with other algorithms for F13, F14, F15 and F16.

The proposed algorithm is the most effective or the second best algorithm in the most of test functions. For instance, the minima of no algorithm for F18, F19, and F20 is equal to f_{min} in Table 4 (i.e. −10.1532, −10.4028 and −10.5363). However, the SFO algorithm can minimize these functions exactly equal to f_{min} in Table 4 due to a very good exploration capability.

Moreover, The SFO has great ability to avoid local optimum with using sardines hunted that are fitter than search agents (sailfish) for leading this algorithm toward the global optimum. The convergence curves of the algorithms on several multimodal test functions are illustrated in Figs. 6 and 7. As seen in these figures, the SFO shows the fastest convergence on multimodal functions. These results indicate that the SFO applies the good balance between exploration and exploitation phases to find the global optimum. Moreover, the statistical results of the algorithms on fixed-dimension multimodal functions are presented in Table 5. As is shown in this table, for F15, the SFO algorithm versus other algorithms provide a p-value greater than 0.05, which means that the superiority of the other algorithms is not statistically significant. In other words, SFO and other algorithms perform very similar and can be considered as the best algorithms for solving F15.

3.3. Performance of SFO on test functions

In general, test function is defined as an artificial problem and can be used to evaluate the behavior of an algorithm in diverse and difficult situations. They may include single global minimum, long narrow valleys, null-space effects, flat surfaces and single or multiple global minima in the presence of many local minima. The objective functions in these problems could be characterized with several parameters such as scalability, convex or non-convex, separable or non-separable, etc. Test functions can easily manipulate and modify to test the algorithms in diverse scenarios. In fact, test functions can be useful to test new algorithms in an unbiased way.

As previously mentioned, the performance of the SFO is compared on the 20 test function with various characteristics. After investigate this compression and review the characteristics of test functions according to Jamil and Yang (2013), the results appear that the SFO algorithm has great ability for solving the non-convex *optimization problem in the short time*. In *non-convex optimization problem*, any function may have multiple feasible regions and multiple locally optimal points within each region. This problem can take a lot of time to identify whether the algorithm finds the global optima or sticks in the local optima. Since a non-convex optimization problem maintains the properties a non-linear programming problem, the SFO algorithm is also suitable for this type of problems. In addition, the SFO is the efficient optimizer for non-separable functions. In these functions, all the parameters or variables of objective functions are dependent and they show inter-relation among themselves. Therefore the non-separable functions cannot divide into sub-objective function and are not relatively easy to solve and optimize. However, the SFO algorithm has outperformed the other comparative algorithms for most non-separable functions in Table 6. Another important property of test functions is scalability. This ability responds well when the dimension of the search space increases. In previous

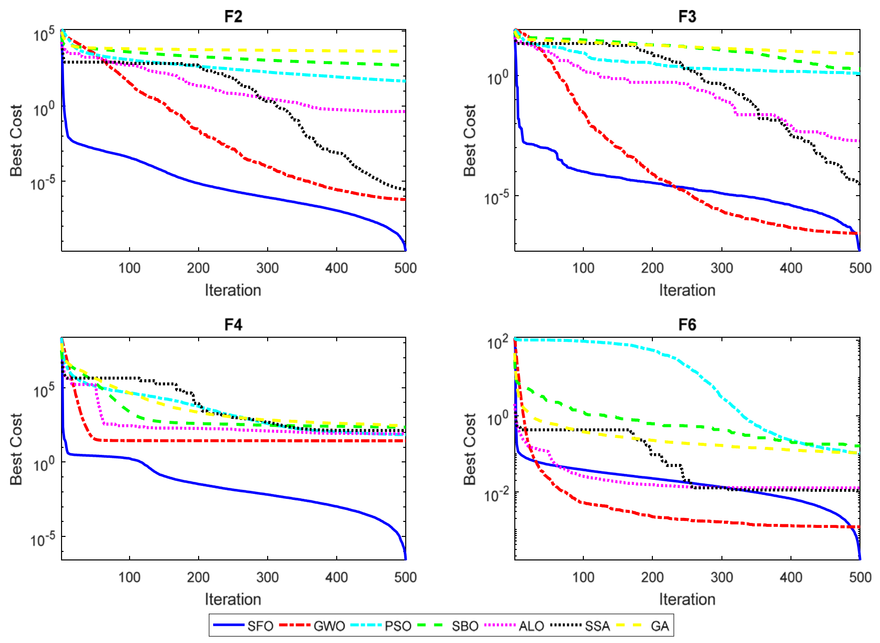

Fig. 5. Convergence curve of algorithms on four unimodal functions.

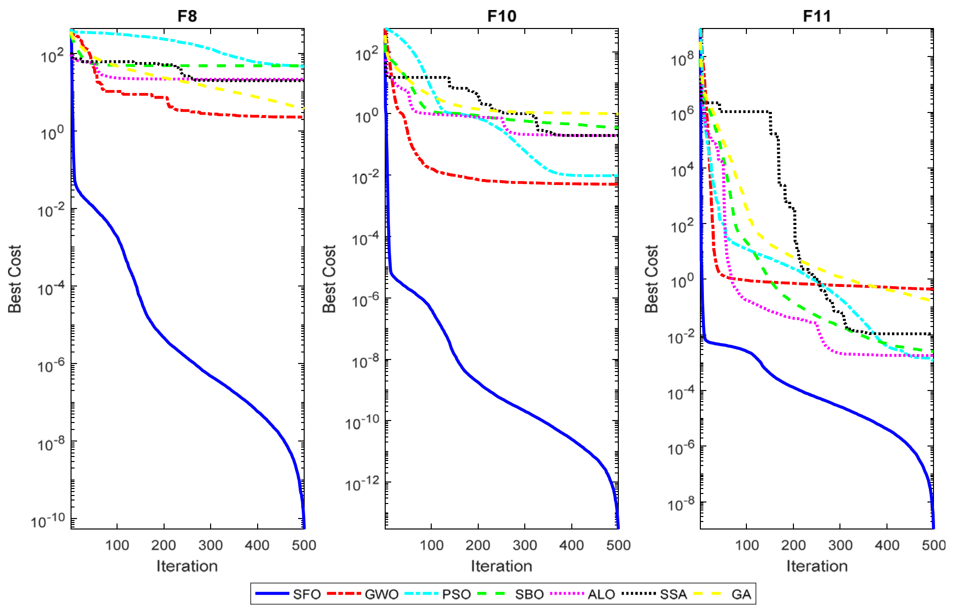

Fig. 6. Convergence curve of algorithms on three of the multimodal functions.

two subsection, several multidimensional scalable test functions are evaluated and the results shows that the SFO algorithm has great ability to optimize scalable problems as well.

3.4. Analysis of the SFO algorithm

In this section, the convergence behavior of SFO and the performance of the proposed algorithm in terms of exploration and exploitation is discussed. To confirm these items, five parameters are employed as follows:

- Search history.
- Route of the first sailfish in its first dimension.
- Sardines hunted.
- Fitness history.
- Convergence curve.

The experiments are re-done with 2 variables and 4 sailfish over 100 iterations. The results are shown in Fig. 8. As it can be seen in this figure, the second column shows the history of search agent's positions over the course of iterations (blue spots). These spots show that the sailfish explore promising areas of the search space and they also exploit around the global optima very accurately. The approximating of global optima can effectively confirm with these observations.

The third column indicates the changes of the position of the first sailfish in the first dimension. This parameter tracks the position of sailfish and it assists to observe the moving of candidate solutions. These observations demonstrate the abrupt changes for the implementation of exploration and gradual changes for the implementation of exploitation. According to Van den Bergh and Engelbrecht (2006), this behavior guarantees the convergence of the population based algorithm to one point in the search space.

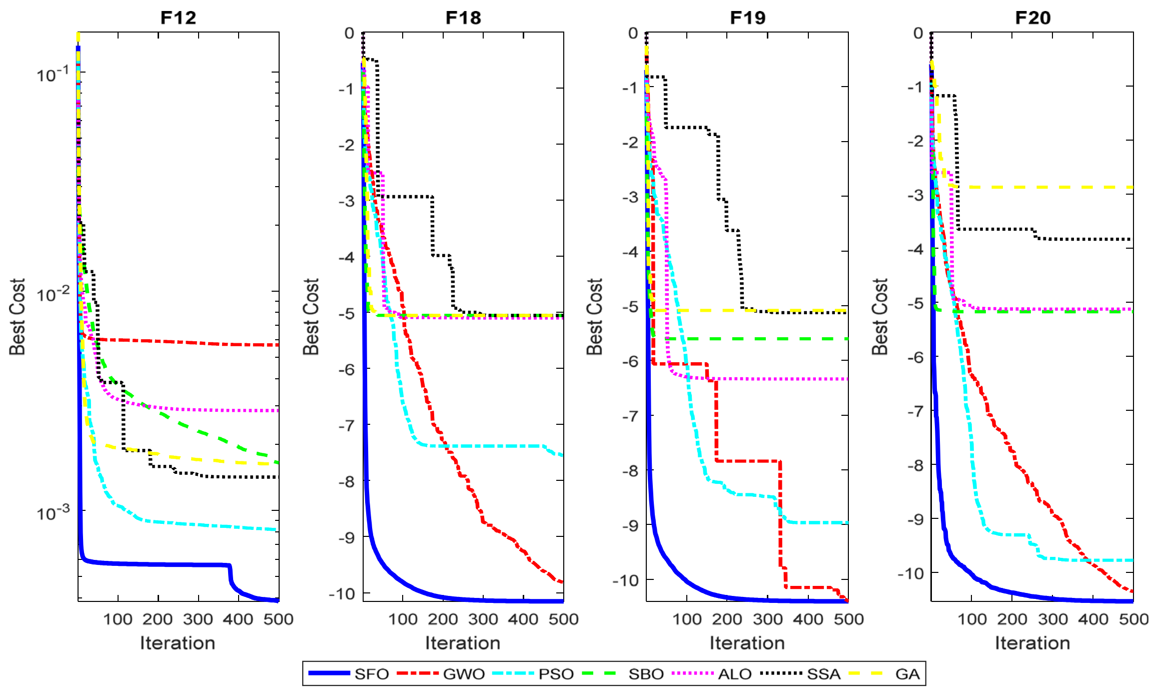

Fig. 7. Convergence curve of algorithms on four of the fixed dimension multimodal functions.

Table 5
P-value of the t test analysis over unimodal and multimodal benchmark functions.

Function	SFO versus GWO	SFO versus SBO	SFO versus ALO	SFO versus PSO	SFO versus SSA	SFO versus GA
F1	3.1e−3	1.97e−323	7.05e−121	5.17e−243	3.62e−173	0
F2	9.38e−149	0	1.49e−270	0	3.34e−137	0
F3	2.40e−30	7.70e−215	1.49e−134	1.01e−207	6.71e−74	3.01e−215
F4	8.68e−233	2.69e−255	1.06e−254	3.18e−248	6.29e−249	9.80e−246
F5	9.94e−26	4.32e−12	3.00e−17	3.10e−17	8.96e−21	5.26e−32
F6	1.37e−32	1.34e−89	3.65e−69	5.82e−90	3.24e−61	7.17e−117
F7	7.79e−08	1.11e−06	5.54e−12	0.99	1.40e−14	4.40e−23
F8	7.45e−320	0	0	0	0	0
F9	4.22e−05	9.03e−199	1.47e−184	5.25e−175	3.13e−188	2.13e−202
F10	0	0	0	0	0	0
F11	1.67e−246	3.13e−185	3.13e−174	3.36e−188	1.18e−167	1.37e−217
F12	7.25e−23	7.75e−20	6.03e−17	0.20	8.79e−16	1.45e−16
F13	4.88e−35	4.88e−35	4.88e−35	4.88e−35	6.40e−36	8.59e−131
F14	1.08e−05	1.08e−05	7.90e−36	7.90e−36	1.03e−37	3.77e−06
F15	0.16	0.61	0.16	0.16	0.08	0.99
F16	4.78e−2	1.68e−40	2.40e−2	2.40e−2	1.68e−2	3.94e−68
F17	6.63e−04	5.5e−3	2.28e−04	9.9e−3	0.99	0.05
F18	1.18e−156	6.62e−179	3.28e−176	3.03e−168	5.35e−194	1.66e−176
F19	5.24e−131	5.30e−172	5.33e−165	6.53e−158	2.44e−144	1.72e−174
F20	5.97e−138	3.92e−159	1.17e−165	7.39e−131	2.93e−146	1.30e−167

The fourth column in Fig. 8 shows the number of sardines that are hunted during optimization. As previously mentioned, when the sardine becomes fitter than its corresponding sailfish, catching the prey will be occurred. The gradual increment in the number of sardines hunted assists search agents to improve the overall fitness and it balances between exploration and exploitation phases.

The fifth column in Fig. 8 illustrates the average fitness of all sailfish and sardines, respectively. The average fitness of sailfish shows the decrement of the fluctuations over the course of iteration on all of the test functions (red line). Also, the blue line shows the fluctuations of the cost function of sardines that assists the sailfish in getting the fitter points. With comparing third and fifth columns in Fig. 8, it is obvious that when the movement of search agent becomes very gradual, the average of its fitness will be degraded in each cycle of the algorithm. As it can be seen in Fig. 8, these curves show the sharp acceleration for the average fitness and they demonstrate that the improvement of solutions become faster over the course of iteration. In the other word, the sailfish swim around the prey school, decrease the number of sardines adaptively in the local search and tend to converge when the counter of iteration increases.

The last column in Fig. 8 illustrates the convergence curves during optimization. The reduction of fitness also indicates the accuracy of the approximation of global optima. In addition, it can be seen in these curves that search agents explore the promising areas during initial iterations and exploit these regions in the final iteration.

4. Optimization of large-scale problems using SFO

To further prove the scalability of SFO algorithm, which is important for applicability of the algorithm to real-world problems, the 300-dimensional versions of the unimodal and multimodal benchmark functions (F1–F11) are employed in this subsection. The population of 30 search agents is used to solve these test problems over 500 iterations. As shown in Table 7, the SFO outperforms all other algorithms except GA for the test function F7. Unlike other algorithms, SFO has achieved these strong results without increasing the number of population and

Table 6
Comparison of optimization results for the benchmark functions.

	Function		SFO	GWO	SBO	ALO	PSO	SSA	GA
Unimodal functions	F1	Mean	6.55e−13	**1.84e−27**	9.57e−2	1.55e−3	1.63e−4	2.75e−07	2.6846
		Std	1.11e−12	4.43e−27	3.74e−2	1.33e−3	2.70e−4	6.97e−07	1.0141
		CPU	0.67 s	0.71 s	6.79 s	14.21 s	0.21 s	0.54 s	1.38 s
	F2	Mean	**2.18e−10**	8.44e−06	865.87	3321.87	86.96	1519.48	4566.03
		Std	6.04e−10	1.83e−05	363.26	1111.49	39.89	1037.73	1620.94
		CPU	4.09 s	1.75 s	7.48 s	14.06 s	1.04 s	1.37 s	2.58 s
	F3	Mean	**3.29e−08**	8.67e−07	1.87	16.86	1.06	12.31	8.21
		Std	2.47e−08	9.65e−07	1.09	5.23	0.23	3.87	2.30
		CPU	0.63	0.56 s	7.06 s	13.94 s	0.24 s	0.51 s	1.44 s
	F4	Mean	**3.86e−07**	27.21	162.56	355.68	92.71	147.98	270.83
		Std	5.38e−07	0.70	189.20	409.01	47.68	355.63	125.55
		CPU	0.56 s	0.57 s	6.62 s	13.81 s	0.31 s	0.56 s	1.53 s
	F5	Mean	0.24	0.72	0.09	0.25	2.90e−4	**9.26e−10**	1.98
		Std	0.07	0.38	0.04	0.08	8.57e−4	3.62e−10	0.71
		CPU	0.67 s	0.58 s	6.65 s	13.93 s	0.22 s	0.51 s	1.47 s
	F6	Mean	**1.1e−4**	1.83e−3	0.16	0.25	0.16	1.28e−2	0.10
		Std	8.86e−05	1.25e−3	3.78e−2	8.24e−2	5.39e−2	1.07e−2	2.96e−2
		CPU	0.89 s	0.72 s	6.65 s	13.79 s	0.34 s	0.65 s	1.85 s
Multimodal functions	F7	Mean	−4662.72	−5947.55	−5752.83	−5668.22	−4550.51	−2789.44	**−10 863.70**
		Std	861.40	838.94	806.62	638.38	1112.60	296.90	313.15
		CPU	0.92 s	0.59 s	6.84 s	14.51 s	0.37 s	0.60 s	1.90 s
	F8	Mean	**6.49e−11**	2.14	49.45	79.63	59.00	16.75	3.73
		Std	2.34e−10	3.75	10.32	23.40	14.80	7.06	2.15
		CPU	0.80 s	0.51 s	7.23 s	4.37 s	0.28 s	0.56 s	1.53 s
	F9	Mean	2.18e−07	**1.05e−13**	1.36	4.48	0.21	0.49	0.50
		Std	3.82e−07	2.23e−14	3.73	3.02	0.49	0.83	0.21
		CPU	0.99 s	0.52 s	7.01 s	13.92 s	0.37 s	0.53 s	1.55 s
	F10	Mean	**2.31e−14**	4.4e−3	0.44	6.72e−2	7e−3	0.19	0.98
		Std	6.46e−14	8.7e−3	0.21	3.82e−2	7.1e−3	0.11	0.07
		CPU	1.02 s	0.57 s	7.24 s	4.88 s	0.36 s	0.68 s	2.29 s
	F11	Mean	**1.96e−09**	0.68	5.26e−3	28.53	6.67e−2	2.16e−3	0.16
		Std	4.52e−09	0.28	3.22e−3	19.69	1.28e−2	5.2e−3	0.07
		CPU	1.76 s	1.04 s	6.99 s	14.00 s	0.79 s	0.99 s	5.11 s
	F12	Mean	**6.9e−4**	4.40e−3	3.56e−3	2.94e−3	8.9e−4	1.59e−3	1.62e−3
		Std	6.5e−4	8.12e−3	4.38e−3	5.93e−3	1.5e−4	3.55e−3	9.1e−4
		CPU	0.72 s	0.36 s	1.36 s	6.04 s	0.20 s	0.55 s	1.36 s
	F13	Mean	**−1.03**	**−1.03**	**−1.03**	**−1.03**	**−1.03**	**−1.03**	**−1.03**
		Std	4.55e−07	2.25e−08	3.34e−06	5.97e−14	6.71e−16	1.435e−14	2.97e−07
		CPU	0.51 s	0.28 s	0.97 s	3.37 s	0.10 s	0.48 s	1.34 s
	F14	Mean	**0.40**	**0.40**	**0.40**	**0.40**	**0.40**	**0.40**	**0.40**
		Std	1.06e−05	0.00029	7.269e−05	6.33e−14	0	2.45e−14	2.04e−09
		CPU	0.49 s	0.29 s	0.95 s	3.38 s	0.11 s	0.41 s	1.19 s
	F15	Mean	**3.00**	**3.00**	4.15	**3.00**	**3.00**	**3.00**	**3.00**
		Std	6.85	4.58e−05	5.07	3.69e−17	1.98e−15	4.19e−13	4.89e−06
		CPU	0.44 s	0.29 s	0.87 s	3.54 s	0.08 s	0.37 s	1.19 s
	F16	Mean	**−3.86**	**−3.86**	−3.81	**−3.86**	**−3.86**	**−3.86**	−3.41
		Std	2.33e−3	3.01e−3	0.17	3.61e−13	2.62e−15	2.47e−11	0.59
		CPU	1.04 s	0.45 s	1.36 s	4.95 s	0.29 s	0.60 s	1.42 s
	F17	Mean	−3.23	**−3.29**	−3.27	**−3.29**	−3.28	−3.23	−3.26
		Std	0.09	8.29e−2	5.92e−2	5.15e−2	6e−2	6.2e−2	6.03e−2
		CPU	1.08 s	0.45 s	1.94 s	8.83 s	0.28 s	0.54 s	1.79 s
	F18	Mean	**−10.15**	−9.23	−4.74	−5.79	−7.83	−6.81	−5.68
		Std	1.06e−06	2.14	3.35	2.86	3.18	3.50	2.70
		CPU	1.04 s	0.60 s	1.72 s	6.27 s	0.42 s	0.82 s	1.61 s
	F19	Mean	**−10.40**	−10.22	−5.77	−7.75	−8.89	−9.24	−4.76
		Std	2.23e−05	0.98	3.62	3.37	2.84	2.67	2.21
		CPU	1.33 s	0.83 s	1.97 s	6.77 s	0.56 s	0.89 s	1.71 s
	F20	Mean	**−10.53**	−10.08	−8.11	−6.46	−10.28	−8.50	−5.77
		Std	5.61e−06	1.75	3.53	3.50	1.41	3.22	3.57
		CPU	1.72 s	0.96 s	2.04 s	6.88 s	0.67 s	1.18 s	1.98 s

maximum iteration compared with comparison of low-dimensional ($n \leq 30$) benchmark problems. For the 11 tested high-dimensional problems, the SFO algorithm converged to near-optimal solutions and it can be scaled up to handle most of the 300-dimensional cases. Therefor these results of SFO in Table 7 can be good evidence for solving large-scale problems as well.

5. Engineering optimization problem

It has proved that no single optimization algorithm has the best performance for all problems with different structures (Wolpert and Macready, 1997). Some of the algorithms can solve some problems

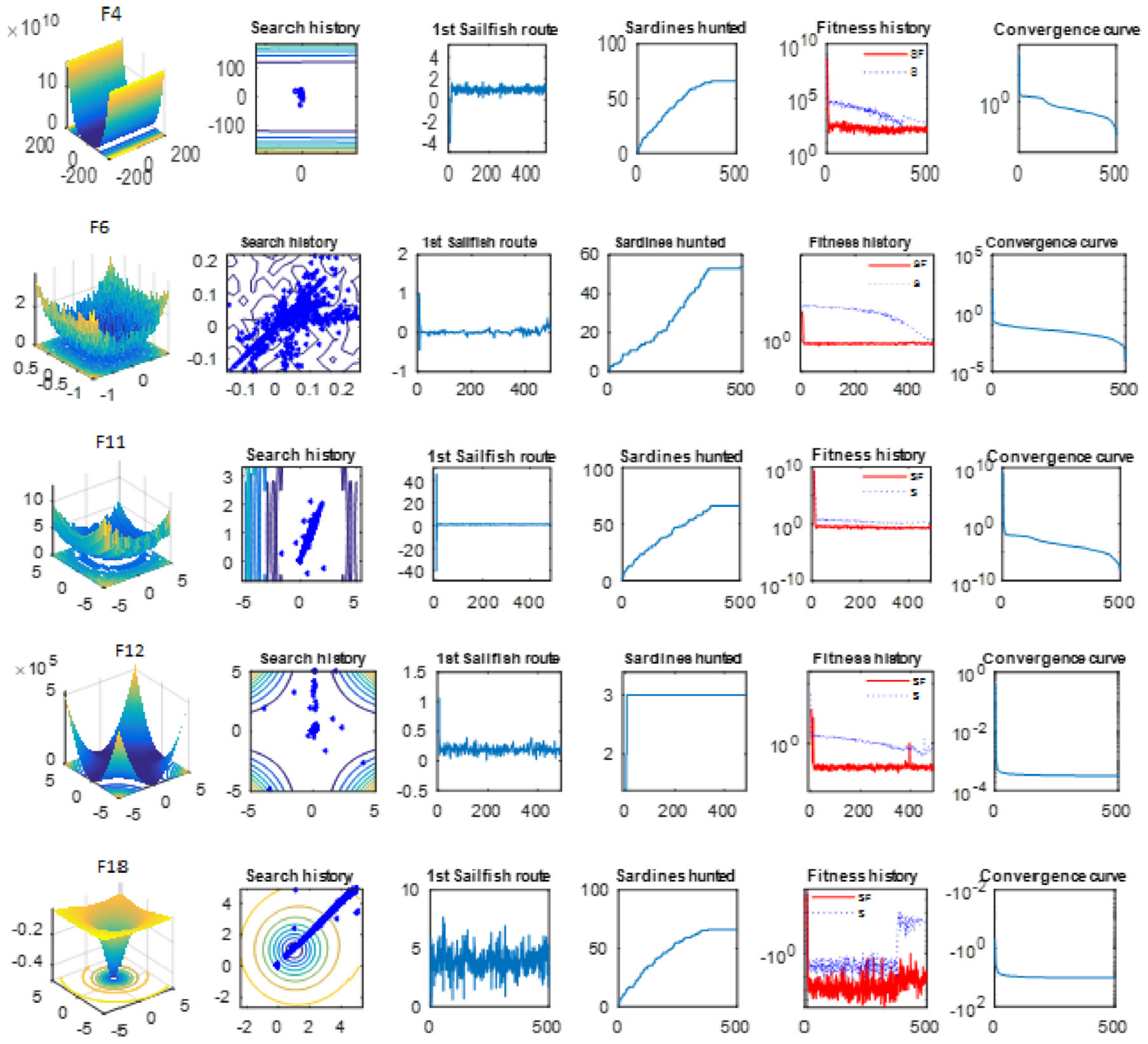

Fig. 8. Search history, trajectory in first dimension, sardines hunted, fitness history, and convergence rate.

Table 7
Result of unimodal and multimodal benchmark functions (300-dimensional).

	Function		SFO	GWO	SBO	ALO	PSO	SSA	GA
Unimodal functions	F1	Mean	**2.49e−14**	1.27e−05	1715.45	84 946.66	1197.40	41 571.17	22 997.81
		Std	2.93e−14	4.35e−06	246.61	20 293.84	99.19	4099.30	3109.23
	F2	Mean	**2.91e−09**	79 573.07	232 560.37	739 745.37	196 201.26	518 751.31	640 003.72
		Std	8.81e−09	27 639.86	25 943.26	249 990.98	72 078.06	241 130.46	196 898.51
	F3	Mean	**6.56e−09**	46.60	38.90	44.28	23.52	37.24	41.70
		Std	5.22e−09	6.96	1.96	3.42	1.38	3.42	2.33
	F4	Mean	**1.72e−06**	297.74	221 533.08	53 231 068.11	3 855 906.63	12 871 662.36	6 020 315.41
		Std	2.71e−06	0.41	58 269.60	22 978 698.04	679 351.70	1 702 878.25	1 753 119.52
	F5	Mean	**0.33**	49.04	1691.51	92 703.27	1257.88	40 213.24	21 951.73
		Std	0.17	1.59	193.57	17 114.14	142.54	4164.26	2401.37
	F6	Mean	**1.80e−4**	2.74e−2	5.41	190.17	19 255.45	61.76	32.56
		Std	1.80e−4	8.39e−3	0.77	57.22	911.35	11.13	7.54
Multimodal functions	F7	Mean	−40 172.20	−40 997.98	−52 321.63	−55 857.90	−18 264.96	−45 316.05	**−56 239.80**
		Std	22 281.76	3044.38	7340.83	9208.07	6320.54	3554.65	3717.84
	F8	Mean	**1.68e−12**	38.20	920.50	1902.26	3318.14	1553.12	1168.30
		Std	6.25e−12	15.21	54.20	167.70	235.16	84.10	60.63
	F9	Mean	**3.90e−08**	2.05e−4	17.25	15.64	8.74	13.73	11.07
		Std	4.08e−08	3.97e−05	0.35	0.67	0.32	0.49	0.56
	F10	Mean	**1.18e−16**	1.44e−2	17.35	766.64	22.04	368.05	200.69
		Std	2.49e−16	2.47e−2	2.88	183.24	6.30	26.38	20.12
	F11	Mean	**1.41e−10**	27.33	8761.14	67 568 040.51	160 608.71	8 829 053.13	3 625 915.34
		Std	2.60e−10	0.80	5329.49	40 934 415.75	78 961.95	3 390 630.44	2 296 342.17

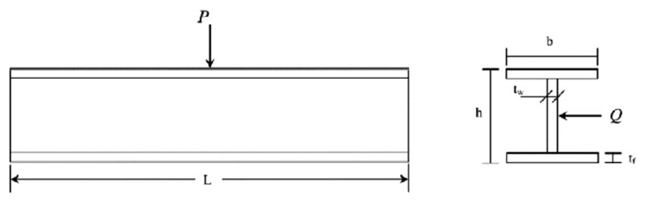

Fig. 9. I-beam design problem.

Table 8
Comparison results for I-beam design problem.

Algorithm	Optimal values for variables				$f(x)$
	b	h	t_w	t_f	
SFO	50	80	1.7637	5.0000	0.00662584
MFO	50	80	1.7647	5.0000	0.0066259
ARSM	37.05	80	1.71	2.31	0.0157
IARSM	48.42	79.99	0.90	2.40	0.131
CS	50	80	0.9	2.3216	0.0130747
SOS	50	80	0.9	2.3217	0.0130741

quickly and well but these algorithms have not the satisfactory performances to solve another problems. In this subsection, solving five engineering design problems such as I-beam design problem, Welded beam design problem, Gear train design problem, Three-bar truss design problem, and circular antenna array design problem are considered to evaluate the ability of SFO algorithm for optimization without biased conclusion.

5.1. I-beam design problem

This problem is one of structural optimization problem that the objective function is to minimize vertical deflection. I-beams are usually made of structural steel and are used in construction and civil engineering. There are several structural parameters such as length, height, and two thicknesses for this problem. As shown in Fig. 9, designing of an I-shaped beam are illustrated. Formulation and constraint of this problem shows as follows:

consider $x = [x_1, x_2, x_3, x_4] = [b, h, t_w, t_f]$

$$\text{Min. } f(x) = \frac{5000}{\frac{t_w(h-2t_f)^3}{12} + \frac{bt_f^3}{6} + 2bt_f\left(\frac{h-t_f}{2}\right)^2}$$

S.t.

$$g(x) = 2bt_w + t_w\left(h - 2t_f\right) \le 0,$$

$$10 \le x_1 \le 50, 10 \le x_2 \le 80, 0.9 \le x_3 \le 5, 0.9 \le x_4 \le 5 \tag{14}$$

Table 8 shows the experimental results on I-beam design problem. As can be seen in this table, SFO are compared with moth-flame optimization algorithm (MFO) (Mirjalili, 2015a,b), adaptive response surface method (ARSM) (Wang, 2003), Improved ARSM (IARSM) (Wang, 2003), Cuckoo search (CS) (Gandomi et al., 2013), and Symbiotic organisms search (SOS) (Cheng and Prayogo, 2014) in the literature. As the results show, SFO produces promising results in comparison with the other algorithms and has a good ability for achieving minimal vertical deflection in this problem.

5.2. Welded beam design problem

As illustrated in Fig. 11 a rigid member welded onto a beam and a load is applied to the end of the member. The total cost of production is equal to the labor costs plus the cost of the weld and beam material. Thus, the objective of this problem is to minimize the fabrication cost of a welded beam. There are four decision variable such as thickness of

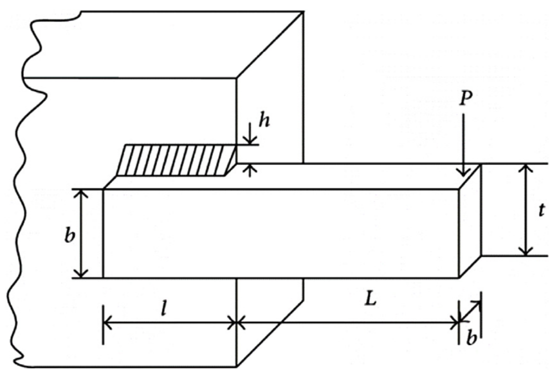

Fig. 10. Design parameters of the welded beam design problem.

weld (h), length of attached part of bar (l), the height of the bar (t), and thickness of the bar (b) as shown in Fig. 10. Also the constraints of this problem are based on shear stress (s), bending stress in the beam (r), bucking load on the mathematical formulation is as follows:

$$\text{Min. } f(x) = 1.10471x_1^2x_2 + 0.04811x_3x_4$$

S.t.

$$\begin{aligned} &g_1(x) = \tau(x) - \tau_{max} \le 0 \\ &g_2(x) = \sigma(x) - \sigma_{max} \le 0 \\ &g_3(x) = x_1 - x_4 \le 0 \\ &g_4(x) = 1.10471x_1^2 + 0.04811x_3x_4\left(14.0 + x_2\right) - 5.0 \le 0 \\ &g_5(x) = 0.125 - x_1 \le 0 \\ &g_6(x) = \delta(x) - \delta_{max} \le 0 \\ &g_7(x) = P - P_c(x) \le 0 \end{aligned} \tag{15}$$

where

$$\tau(x) = \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2}$$

$$\tau' = \frac{P}{\sqrt{2}x_1x_2}, \tau'' = \frac{MR}{J}, M = P(L + \frac{x_2}{2})$$

$$R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2}, J = 2\left\{\sqrt{2}x_1x_2\left[\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\}$$

$$\sigma(x) = \frac{6PL}{x_4x_3^2}, \delta(x) = \frac{6PL^3}{Ex_4x_3^2}$$

$$P_c(x) = \frac{4.013E\sqrt{\frac{x_3^2x_4^6}{36}}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right)$$

$P = 6000$ lb, $L = 14$ in, $E = 30 \times 10^6$ psi, $G = 12 \times 10^6$ psi

$\tau_{max} = 13{,}600$ psi, $\sigma_{max} = 30{,}000$ psi, $\delta_{max} = 0.25$ in

$0.1 \le x_1 \le 2,\ 0.1 \le x_2 \le 10,$

$0.1 \le x_3 \le 10,\ 0.1 \le x_4 \le 2$

Welded beam is solved by the SFO algorithm and compared to Coello (2002), GA (Carlos and Coello, 2000), GSA (Rashedi et al., 2009), Siddall (1972), Richardson's random method, Simplex method, Davidon–Fletcher–Powell (Ragsdell and Phillips, 1976). Table 9 shows the SFO is able to find best solution and corresponding constraint value for this problem. In this compression, SFO can outperform all algorithms and outperforms by CPSO (He and Wang, 2007).

5.3. Gear train design problem

The design of gear train is a kind of mixed problems which have to determine various types of design variables such as continuous,

Table 9
Comparison results for welded beam design problem.

Algorithm	Optimal values for variables				$f(x)$
	h	l	t	b	
SFO	0.2038	3.6630	9.0506	0.2064	1.73231
CPSO	0.2023	3.5442	9.0482	0.2057	1.73148
Coello	0.2088	3.42050	8.997500	0.2100	1.74831
GA	0.1829	4.0483	9.3666	0.2059	1.82420
GSA	0.1821	3.85697	10.0000	0.202376	1.87995
Siddall	0.2444	6.2189	8.2915	0.2444	2.38154
David	0.2434	6.2552	8.2915	0.2444	2.38411
Ragsdell	0.2455	6.1960	8.2730	0.2455	2.38594
Simplex	0.2792	5.6256	7.7512	0.2796	2.53073
Random	0.4575	4.7313	5.0853	0.6600	4.11856

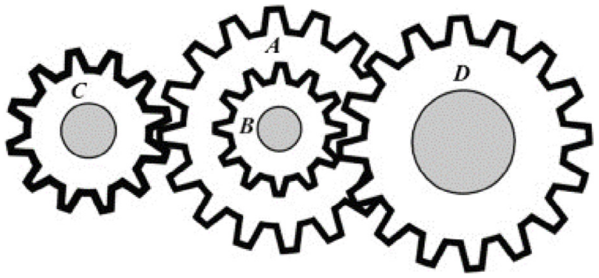

Fig. 11. Design parameters of the gear train design problem.

Table 10
Comparison results for gear train design problem.

Algorithm	Optimal values for variables				$f(x)$
	n_a	n_b	n_c	n_d	
SFO	49	16	19	43	2.700857148886513e−12
CSA	49	19	16	43	2.70085714889e−12
MBA	43	16	19	49	2.700857e−12
MFO	43	19	16	49	2.7009e−12
GA	49	16	19	43	2.7019e−012
CS	19	16	43	49	2.701e−12
Kannan and Kramer	33	15	13	41	2.1469e−08
Sandgren	18	22	45	60	5.712e−06

discrete, and integer variables. This problem simply stated is: given a fix input drive and a number of fixed output drive spindles, how can the spindles be driven by the input using the minimum number of connecting gear in the train. Therefore, the objective of gear train design problem is to minimize the cost of the gear ratio of the gear train in field mechanical engineering problem as shown in Fig. 11. The parameters in this problem are the number of teeth of the gears. n_a, n_b, n_c and n_d are the variables in gear train problem and this problem is formulated as follow:

$$\text{Min. } f(x) = \left(\frac{1}{6.931} - \frac{x_3 x_2}{x_1 x_4}\right)^2 \tag{16}$$

S.t.

$12 \le x_1, x_2, x_3, x_4 \le 60$

Table 10 compares the performance of the SFO with Mine blast algorithm (MBA) (Sadollah et al., 2013), Crow search algorithm (CSA) (Askarzadeh, 2016), Moth-flame optimization algorithm (MFO) (Mirjalili, 2015a,b), genetic algorithms (GA) (Deb and Goyal, 1996), Cuckoo search algorithm (CS) (Gandomi et al., 2013), Kannan and Kramer (1994) and Sandgren (1990) in the literature. As seen in these results, SFO has the same result with MBA and CSA and outperforms other algorithms. Since the gear train is a discrete problem, these results prove that SFO has a good ability for solving discrete problems as well.

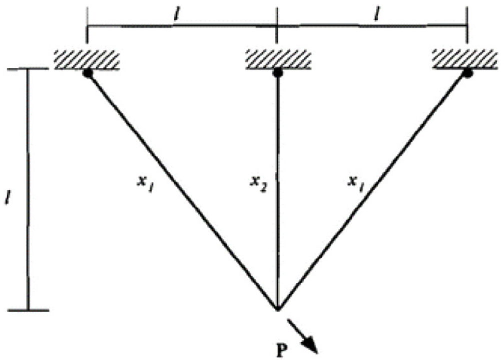

Fig. 12. Schematic of three-bar truss design problem.

Table 11
Comparison results for three-bar truss problem.

Algorithm	Optimal values for variables		$f(x)$
	x_1	x_2	
SFO	0.7884562	0.40886831	263.89592128
PSO-DE	0.7886751	0.4082482	263.8958433
MBA	0.7885650	0.4085597	263.8958522
MFO	0.78824477	0.40946690	263.89597968
CS	0.78867	0.40902	263.9716
Ray and Saini	0.795	0.395	264.3

5.4. Three-bar truss design problem

This problem is one of optimization problem in the field of civil engineering. Due to This problem difficult constrained search space, this problem has been mostly utilized to evaluate the optimization power of proposed algorithms. Fig. 12 illustrate the schematic of three-bar truss design problem. The objective of this problem is to find the optimal values of the cross-sectional areas of the bars, which minimize the weight of the structure, subject to maximize stress constraints on the three bars. In this problem, there is a nonlinear fitness function with three nonlinear inequality constraints and there are two continuous variables for decision. The formulation of this problem is as follows:

$$\text{Min.} f(x) = \left(2\sqrt{2}x_1 + x_2\right) * l \tag{17}$$

S.t.

$$g_1(x) = \frac{\sqrt{2}x_1 + x_2}{\sqrt{2x_1^2 + 2x_1x_2}} P - \sigma \le 0$$

$$g_2(x) = \frac{x_2}{\sqrt{2x_1^2 + 2x_1x_2}} P - \sigma \le 0$$

$$g_1(x) = \frac{1}{\sqrt{2x_2 + x_1}} P - \sigma \le 0$$

$0 \le x_1, x_2 \le 1$

Table 11 presents the best weights obtained and the optimal values for the decision variables by SFO and several algorithms. As shown in Table 11, SFO compared to PSO-DE (Liu et al., 2010), MBA (Sadollah et al., 2013), MFO (Mirjalili, 2015a,b), Ray and Saini (2001) and CS (Gandomi et al., 2013) algorithms in the literature. The results show that SFO outperforms three of the algorithms and outperforms by PSO-DE and MBA.

5.5. Circular antenna array design problem

Circular antenna array design problem is one example of the CEC 2011 competition which consider of real world optimization problems (Das and Suganthan, 2010). This problem has various applications in sonar, radar, mobile and commercial satellite communication systems

Table 12
Comparison results for Circular antenna array design problem.

Algorithm	Optimal values for variables												$f(x)$
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	
SFO	0.0097	0.0097	0.0091	0.0038	0.0068	0.0093	1.7028	−1.7708	1.7503	−1.7595	−1.7081	1.7660	−13.916
SBO	1.4918	1.5010	0.2718	0.6113	0.6145	0.5705	−15.234	5.3373	−23.373	38.0860	6.8295	−11.864	−11.227
WCA	1	0.5498	0.3167	0.3563	0.2387	0.8040	0.2	0.9715	0.2	0.7601	0.9987	0.2001	−10.370
GA	1.2378	0.5828	0.4172	0.3880	0.4023	0.5573	−23.724	−7.9299	−40.292	0.1474	53.6568	5.9731	−11.942

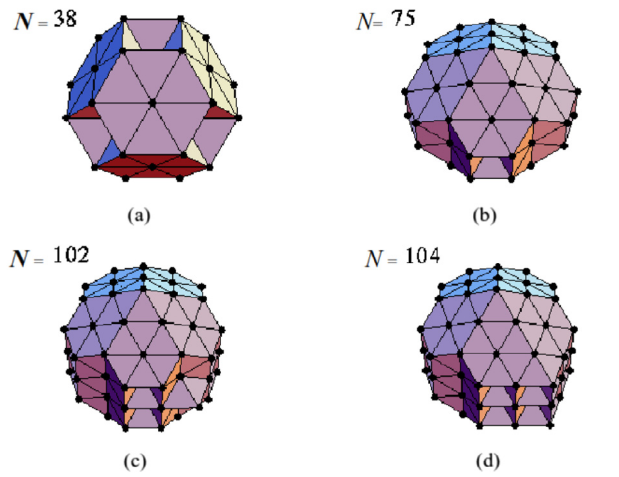

Fig. 13. (a), (b), (c), and (d) are the shapes for 38, 75, 102 and 104 atoms.

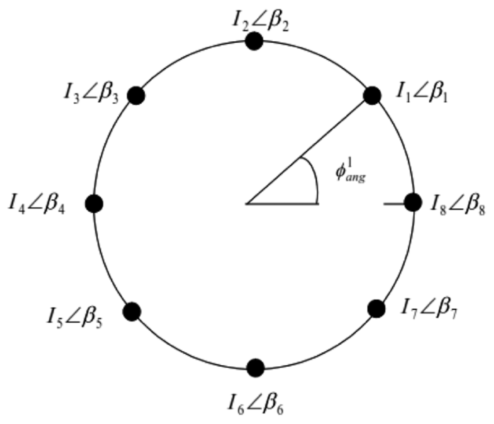

Fig. 14. Geometry of circular antenna array.

(Dessouky et al., 2006a; Gurel and Ergul, 2008; Dessouky et al., 2006b). Fig. 13 is shown N antenna elements spaced on a circle in the x-y plane and the antenna elements constitute a circular antenna array, as it is illustrated in the Fig. 14. The array factor for the circular array is formulated as follows:

$$AF(\varphi) = \sum_{n=1}^{N} I_n exp\left[jk_r\left(cos\left(\varphi - \varphi^n_{ang}\right) - cos\left(\varphi_0 - \varphi^n_{ang}\right)\right) + \beta_n\right] \quad (18)$$

$$\varphi^n_{ang} = \frac{2\pi(n-1)}{N},\ k_r = Nd,$$

where φ^n_{ang} is the angular position and k_r is the wave-number, d is the angular spacing between elements, φ_0 is the direction of maximum, φ is the angle of incidence of the plane wave, I_0 is the current excitation and β is the phase excitation of the nth element. In the circular antenna array design problem, the current excitations of the antenna elements vary and try to suppress side-lobes, minimize beamwidth and achieve null control at desired directions. The objective function is taken as,

$$OF = \frac{|AR(\varphi_{sll}, I, \beta, \varphi_0)|}{|AR(\varphi_{max}, I, \beta, \varphi_0)|} + \frac{1}{DIR\left(\varphi_0, I, \beta\right) + |\varphi_0 - \varphi_{des}| + \sum_{k=1}^{num}|AR(\varphi_{sll}, I, \beta, \varphi_0)|} \quad (19)$$

As illustrated in Fig. 14, for considering a symmetrical excitation of the circular antenna array the relations given below will hold,

$$I_{\frac{n}{2}+1}\angle\beta_{\frac{n}{2}+1} = conj(I_1\angle\beta_1),$$

$$I_{\frac{n}{2}+2}\angle\beta_{\frac{n}{2}+2} = conj\left(I_1\angle\beta_1\right), \ldots$$

$$I_n\angle\beta_n = conj(I_{\frac{n}{2}}\angle\beta_{\frac{n}{2}}) \quad (20)$$

This problem mimics another characteristic of real world optimization problems and is different with other employed problems in this section. To provide the instantiation of the design problem for this competition, the number of elements in circular array is 12, x_I is any string within bounds, null is equal to [50,120] in radians (no null control), phi_desired is 180° and distance parameter is equal to 0.5. As shown in Table 12, SFO compared to SBO (Moosavi and Khatibi Bardsiri, 2017), WCA (Eskandar and Bahreininejad, 2012) and GA (Holland, 1992) algorithms in the literature. The results show that SFO outperforms these algorithms and the results in Table 12 evidence that the SFO algorithm has ability to solve these types of problems efficiently as well.

6. Conclusion

This paper proposed a novel swarm intelligence optimization algorithm and is inspired by the group hunting behavior of sailfish. The proposed algorithm that is called SFO (SailFish Optimizer) simulated the search for prey, attack-alternation strategy, and hunting and catching prey. The SFO algorithm evaluated with 24 mathematical benchmark functions for analyzing exploration and exploitation phases, local optima avoidance, and convergence behavior of the proposed algorithm. The superior results reported that the proposed algorithm achieved high exploration, exploitation and convergence speed on the majority of test functions. In addition, since SFO employed two populations of prey and predator, the local optima stagnation become very low. Based on the amount of maneuverability of sardines and every dimensions of problem, the SFO algorithm provided the diverse movement behaviors for sardines inside the encircled area and guarantees the variety of position explored around the hunter. However, this movement is decreased adaptively and leads the SFO algorithm to convergence during the optimization. According to the severity of the attack by sailfish, when the iteration counter increases, the SFO algorithm defines the boundaries of maneuverability of sardines which promote the exploitation phase during optimization. Also since the position of the best sailfish is saved in each iteration and considered as the elite, the position of sardines will be updated toward the best solution according to amount of sailfish's attack power. With this strategy, the promising area of search space will be saved at each course of iteration. And the last point is to reduce the number of sardines after hunting that provides the dynamic balance between exploration and exploitation phases in the search space. The SFO algorithm also showed significant results for non-convex, non-separable and scalable test functions. Eventually, the results of the proposed algorithm were compared with several state-of-the-art metaheuristic algorithms in literature and it was found to be enough competitive and superior over conventional techniques.

The proposed SFO may contribute to several research directions for future works. The first direction can be applying this method to solve various optimization problems in different domains. Another direction can be developing a version with the parallelization process to demonstrate more functionality and efficiency of the SFO algorithm. In addition, to solve multi objective problems with this algorithm can be considered as a good contribution.

Conflict of interest

The authors declare that there is no conflict of interest.

References

Askarzadeh, A., 2016. A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm. Comput. Struct. 169, 1–12.

Bailey, I., Myatt, J.P., Wilson, A.M., 2013. Group hunting within the carnivora: physiological, cognitive and environmental influences on strategy and cooperation. Behav. Ecol. Sociobiol. 67, 1–17.

Van den Bergh, F., Engelbrecht, A., 2006. A study of particle swarm optimization particle trajectories. Inform. Sci. 176, 937–971.

Carlos, A., Coello, C., 2000. Constraint-handling using an evolutionary multiobjective optimization technique. Civ. Eng. Syst. 17, 319–346.

Cheng, M.-Y., Prayogo, D., 2014. Symbiotic organisms search: a new metaheuristic optimization algorithm. Comput. Struct. 139, 98–112.

Coello, C.A., 2002. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Comput. Methods Appl. Mech. Engrg. 191, 1245–1287.

Das, Swagatam, Suganthan, P.N., 2010. Problem Definitions and Evaluation Criteria for CEC 2011 Competition on Testing Evolutionary Algorithms on Real World Optimization Problems, Technical Report.

Deb, K., Goyal, M., 1996. A combined genetic adaptive search (GeneAS) for engineering design. Comput. Sci. Inform. 26, 30–45.

Dessouky, M., Sharshar, H., Albagory, Y., 2006a. A novel tapered beamforming window for uniform concentric circular arrays. J. Electromagn. Waves Appl. 20 (14), 2077–2089.

Dessouky, M.I.H., Sharshar, A., Albagory, Y.A., 2006b. Efficient sidelobe reduction technique for small-sized concentric circular arrays. Prog. Electromagn. Res. PIER 65, 187–200.

Domenici, P., et al., 2014. How sailfish use their bills to capture schooling prey. Proc. R. Soc. B 281. http://dx.doi.org/10.1098/rspb.2014.0444.

Dorigo, M., Birattari, M., Stutzle, T., 2006. Ant colony optimization. IEEE Comput. Intell. 1, 28–39.

Eskandar, Sadollah, Bahreininejad, Hamdi, 2012. Water cycle algorithm - a novel metaheuristic optimization method for solving constrained engineering optimization problems. Comput. Struct. 110–111, 151–166.

Gandomi, A.H., Yang, X.-S., Alavi, A.H., 2013. Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Eng. Comput. 29, 17–35.

Glover, F., 1989. Tabu search – Part I. ORSA J. Comput. 1, 190–206.

Gurel, L., Ergul, O., 2008. Design and simulation of circular arrays of trapezoidal-tooth logperiodic antennas via genetic optimization. Prog. Electromagn. Res. PIER 85, 243–260.

He, Q., Wang, L., 2007. An effective co-evolutionary particle swarm optimization for constrained engineering design problems. Eng. Appl. Artif. Intell. 20, 89–99.

Herbert-Read James, E., et al., 2016. Proto-cooperation: group hunting sailfish improve hunting success by alternating attacks on grouping prey. R. Soc. http://dx.doi.org/10.1098/rspb.2016.1671.

Holland, J.H., 1992. Genetic algorithms. Sci. Am. 267, 66–72.

Jamil, M., Yang, X., 2013. A literature survey of benchmark functions for global optimization problems. Int. J. Math. Model. Numer. Optim. 4 (2), 150–194.

Kannan, B., Kramer, S.N., 1994. An augmented Lagrange multiplier based method for mixed integer discrete continuous optimization and its applications to mechanical design. J. Mech. Des. 116, 405–411.

Kennedy, J., Eberhart, R.C., 1995. Particle swarm optimization. In: Proc. IEEE International Conference on Neural Networks. Piscataway, NJ, pp. 1942–1948.

Liu, H., Cai, Z., Wang, Y., 2010. Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization. Appl. Soft Comput. 10, 629–640.

Marras, S., et al., 2015. Not so fast: swimming behavior of sailfish during predator –prey interactions using high-speed video and accelerometry. Integr. Comput. Biol. 55, 719–727. http://dx.doi.org/10.1093/icb/icv017.

Mirjalili, S., 2015a. Moth-flame optimization algorithm: A novel nature-inspired heuristic Paradigm. Knowl.-Based Syst. 89, 228–249.

Mirjalili, S., 2015b. The ant lion optimizer. Adv. Eng. Softw. 83, 80–98.

Mirjalili, S., Mirjalili, S.M., Lewis, A., 2014. Grey wolf optimizer. Adv. Eng. Softw. 69, 46–61.

Mirjalili, S., et al., 2017. Salp Swarm Algorithm: A bio-inspired optimizer for engineering design problems. Adv. Eng. Softw. 1–29.

Moosavi, S., Khatibi Bardsiri, V., 2017. Satin bowerbird optimizer: A new optimization algorithm to optimize ANFIS for software development effort estimation. Eng. Appl. Artif. Intell. 60, 1–15.

Ragsdell, K., Phillips, D., 1976. Optimal design of a class of welded structures using geometric programming. ASME J. Eng. Ind. 98, 1021–1025.

Rashedi, E., Nezamabadi-Pour, H., Saryazdi, S., 2009. GSA: a gravitational search algorithm. Inform. Sci. 179, 2232–2248.

Ray, T., Saini, P., 2001. Engineering design optimization using a swarm with an intelligent information sharing among individuals. Eng. Optmiz. 33 (6), 735–748.

Sadollah, A., Bahreininejad, A., Eskandar, H., Hamdi, M., 2013. Mine blast algorithm: a new population based algorithm for solving constrained engineering optimization problems. Appl. Soft Comput. 13, 2592–2612.

Sandgren, E., 1990. Nonlinear integer and discrete programming in mechanical design optimization. J. Mech. Des. 112 (2), 223–229.

Siddall, J.N., 1972. Analytical Decision-Making in Engineering Design. Prentice-Hall, Englewood Cliffs, NJ.

Suganthan, P.N., Hansen, N., Liang, J.J., Deb, K., Chen, Y.-P., Auger, A., et al., 2005. Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization, KanGAL report 2005005.

Wang, G.G., 2003. Adaptive response surface method using inherited latin hypercube design points. J. Mech. Des. 125, 210–220.

Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. IEEE Trans. Evolut. Comput. 1, 67–82.

Yang, X.S., 2018. Nature-Inspired Algorithms and Applied Optimization, first ed. Springer International Publishing.