Advances in Engineering Software 105 (2017) 30–47

Contents lists available at ScienceDirect

Advances in Engineering Software

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



Research paper

# Grasshopper Optimisation Algorithm: Theory and application

Shahrzad Saremi[a,b], Seyedali Mirjalili[a,b,*], Andrew Lewis[a]

[a] School of Information and Communication Technology, Griffith University, Nathan, Brisbane, QLD 4111, Australia
[b] Griffith College, Mt Gravatt, Brisbane, QLD 4122, Australia

ARTICLE INFO

*Article history:*
Received 29 October 2016
Accepted 10 January 2017
Available online 31 January 2017

*Keywords:*
Optimization
Optimization techniques
Heuristic algorithm
Metaheuristics
Constrained optimization
Benchmark
Algorithm

ABSTRACT

This paper proposes an optimisation algorithm called Grasshopper Optimisation Algorithm (GOA) and applies it to challenging problems in structural optimisation. The proposed algorithm mathematically models and mimics the behaviour of grasshopper swarms in nature for solving optimisation problems. The GOA algorithm is first benchmarked on a set of test problems including CEC2005 to test and verify its performance qualitatively and quantitatively. It is then employed to find the optimal shape for a 52-bar truss, 3-bar truss, and cantilever beam to demonstrate its applicability. The results show that the proposed algorithm is able to provide superior results compared to well-known and recent algorithms in the literature. The results of the real applications also prove the merits of GOA in solving real problems with unknown search spaces.



## 1. Introduction

The process of finding the best values for the variables of a particular problem to minimise or maximise an objective function is called optimisation. Optimisation problems exist in different fields of studies. To solve an optimisation problem, different steps need to be taken. Firstly, the parameters of the problem should be identified. Based on the nature of the parameters, problems may be classified as continuous or discrete. Secondly, the constraints that are applied to the parameters have to be recognised [1]. Constraints divide the optimisation problems into constrained and unconstrained. Thirdly, the objectives of the given problem should be investigated and considered. In this case, optimisation problems are classified into single-objective versus multi-objective problems [2]. Finally, based on the identified types of parameters, constraints, and number of objectives a suitable optimiser should be chosen and employed to solve the problem.

Mathematical optimisation mainly relies on gradient-based information of the involved functions in order to find the optimal solution. Although such techniques are still being used by different researchers, they have some disadvantages. Mathematical optimisation approaches suffer from local optima entrapment. This refers to an algorithm assuming a local solution is the global solution, thus failing to obtain the global optimum. They are also often ineffective for problems with unknown or computationally expensive derivation [3]. Another type of optimisation algorithm that alleviates these two drawbacks is stochastic optimisation [4].

Stochastic methods rely on random operators that allow them to avoid local optima. They all start optimisation process by creating one or a set of random solutions for a given problem. In contrast to mathematical optimisation techniques, they do not need to calculate the gradient of a solution, just evaluating the solutions using the objective function(s). Decisions as to how to improve the solutions are made based on the calculated objective values. Therefore, the problem is considered as a black box, which is a very useful mechanism when solving real problems with unknown search spaces. Due to these advantages, stochastic optimisation techniques have become very popular over the past two decades [5].

Among stochastic optimisation approaches, nature-inspired, population-based algorithms are the most popular [6]. Such techniques mimic natural problems-solving methods, often those used by creatures. Survival is the main goal for all creatures. To achieve this goal, they have been evolving and adapting in different ways. Therefore, it is wise to seek inspiration from nature as the best and oldest optimiser on the planet. Such algorithms are classified into two main groups: single-solutions-based and multi-solution-based. In the former class, a single random solution is generated and improved for a particular problem. In the latter class, however, multiple solutions are generated and enhanced for a given problem. Multi-solution-based algorithms are more popular than single-solution-based methods, as the literature shows [7].

Multi-solution-based algorithms intrinsically have higher local optima avoidance due to improving multiple solutions during optimisation. In this case, a trapped solution in a local optimum can be assisted by other solutions to jump out of the local

* Corresponding author.
*E-mail address:* seyedali.mirjalili@griffithuni.edu.au (S. Mirjalili).
*URL:* http://www.alimirjalili.com (S. Mirjalili)

http://dx.doi.org/10.1016/j.advengsoft.2017.01.004
0965-9978/

optimum. Multiple solutions explore a larger portion of the search space compared to single-solution-based algorithms, so the probability of finding the global optimum is high. Also, information about the search space can be exchanged between multiple solutions, which results in quick movement towards the optimum. Although multi-solution-based algorithms have several advantages, they require more function evaluations.

The most popular single-solution-based algorithms are hill climbing [8] and simulated annealing [9]. Both algorithms follow a similar idea, but the local optima avoidance of SA is higher due to the stochastic cooling factor. Other recent single-solution-based algorithms are Tabu Search (TS) [10,11], and Iterated Local Search (ILS) [12]. The most popular multi-solutions-based algorithms are Genetic Algorithms (GA) [13], Particle Swarm Optimisation (PSO) [14], Ant Colony Optimisation (ACO) [15], and Differential Evolution (DE) [16]. The GA algorithm was inspired by the Darwinian theory of evolution. In this algorithm, solutions are considered as individuals and the parameters of solutions take the place of their genes. Survival of the fittest individuals is the main inspiration of this algorithm where the best individuals tend to participate more in improving poor solutions. The PSO algorithm simulates the foraging of herds of birds or schools of fishes. In this algorithm the solutions are improved with respect to the best solutions obtained so far by each of the particles and the best solution found by the swarm. The ACO algorithm mimics the collective behaviour of ants in finding the shortest path from the nest to the source of foods. Finally, DE utilises simple formulae combining the parameters of existing solutions to improve the population of candidate solutions for a given problem.

The similarity of both classes of nature-inspired algorithms is the improvement of solutions until the satisfaction of an end criterion and the division of optimisation process into two phases: exploration versus exploitation [17]. Exploration refers to the tendency for an algorithm to have highly randomised behaviour so that the solutions are changed significantly. Large changes in the solutions cause greater exploration of the search space and consequently discovery of its promising regions. As an algorithm tends toward exploitation, however, solutions generally face changes on a smaller scale and tend to search locally. A proper balance of exploration and exploitation can result in finding the global optimum of a given optimisation problem.

The literature shows that there are many recent swarm intelligence optimisation techniques such as Dolphin Echolocation (DEL) [18,19], Firefly Algorithm (FA) [20,21], Bat Algorithm (BA) [22], and Grey Wolf Optimizer (GWO) [3]. DEL and BA mimic echolocation of dolphins in finding prey and bats navigating respectively. However, FA simulates the mating behaviour of fireflies in nature. Cuckoo Search (CS) [23,24] is another recent algorithm in this field, in which the reproductive processes of cuckoos are employed to propose an optimisation algorithm. The GWO is also a swarm-based technique that models the hunting mechanism of grey wolves.

There are also other algorithms with different inspiration in the literature. For instance, State of Matter Search (SMS) [25,26] uses the concepts of different phases in matter to optimise problems and the Flower Pollination Algorithm (FPA) [27] has been inspired by the survival and reproduction of flowers using pollination. There is a question here as to why we need more algorithms despite the many algorithms proposed so far.

The answer to this question is in the No Free Lunch (NFL) theorem [28] that logically has proven that there is no optimisation technique for solving all optimisation problems. In other words, algorithms in this field perform equally on average when considering *all* optimisation problems. This theorem, in part, has motivated the rapidly increasing number of algorithms proposed over the last decade and is one of the motivations of this paper as well. The next section proposes a new algorithm mimicking the behaviour of grasshopper swarms. There are a few works in the literature that have tried to simulate locust swarm [29–33]. The current study is an attempt to more comprehensively model grasshopper behaviours and propose an optimisation algorithm based on their social interaction.

Due to their simplicity, gradient-free mechanism, high local optima avoidance, and considering problems as black boxes, nature-inspired algorithms have been applied widely in science and industry [34–36]. Therefore, we also investigate the application of the proposed algorithm in solving real problems. The rest of the paper is organised as follows:

The Grasshopper Optimisation Algorithm is proposed in Section 2. Section 3 presents and discusses the results on the optimisation test beds and inspects the behaviour of the proposed algorithm. Section 4 contains the application of the proposed method in the field of structural design optimisation. Finally, Section 5 concludes the work and suggests several directions for future studies.

## 2. Grasshopper Optimisation Algorithm (GOA)

Grasshopper are insects. They are considered a pest due to their damage to crop production and agriculture. The life cycle of grasshoppers is shown in Fig. 1. Although grasshoppers are usually seen individually in nature, they join in one of the largest swarm of all creatures [37]. The size of the swarm may be of continental scale and a nightmare for farmers. The unique aspect of the grasshopper swarm is that the swarming behaviour is found in both nymph and adulthood [38]. Millions of nymph grasshoppers jump and move like rolling cylinders. In their path, they eat almost all vegetation. After this behaviour, when they become adult, they form a swarm in the air. This is how grasshoppers migrate over large distances.

The main characteristic of the swarm in the larval phase is slow movement and small steps of the grasshoppers. In contrast, long-range and abrupt movement is the essential feature of the swarm in adulthood. Food source seeking is another important characteristic of the swarming of grasshoppers. As discussed in the introduction, nature-inspired algorithms logically divide the search process into two tendencies: exploration and exploitation. In exploration, the search agents are encouraged to move abruptly, while they tend to move locally during exploitation. These two functions, as well as target seeking, are performed by grasshoppers naturally. Therefore, if we find a way to mathematically model this behaviour, we can design a new nature-inspired algorithm.

The mathematical model employed to simulate the swarming behaviour of grasshoppers is presented as follows [39]:

$$X_i = S_i + G_i + A_i \tag{2.1}$$

where $X_i$ defines the position of the *i-th* grasshopper, $S_i$ is the social interaction, $G_i$ is the gravity force on the *i-th* grasshopper, and $A_i$ shows the wind advection. Note that to provide random behaviour the equation can be written as $X_i = r_1 S_i + r_2 G_i + r_3 A_i$ where $r_1$, $r_2$, and $r_3$ are random numbers in [0,1].

$$S_i = \sum_{\substack{j=1 \\ j \neq i}}^{N} s\left(d_{ij}\right) \widehat{d_{ij}} \tag{2.2}$$

where $d_{ij}$ is the distance between the *i-th* and the *j-th* grasshopper, calculated as $d_{ij} = |x_j - x_i|$, $s$ is a function to define the strength of social forces, as shown in Eg. (2.3), and $\widehat{d_{ij}} = \frac{x_j - x_i}{d_{ij}}$ is a unit vector from the *i*th grasshopper to the *j*th grasshopper.

The $s$ function, which defines the social forces, is calculated as follows:

$$s(r) = f e^{\frac{-r}{l}} - e^{-r} \tag{2.3}$$


**Fig. 1.** (a) Real grasshopper (b) Life cycle of grasshoppers (left image courtesy of Mehrdad Momeny).

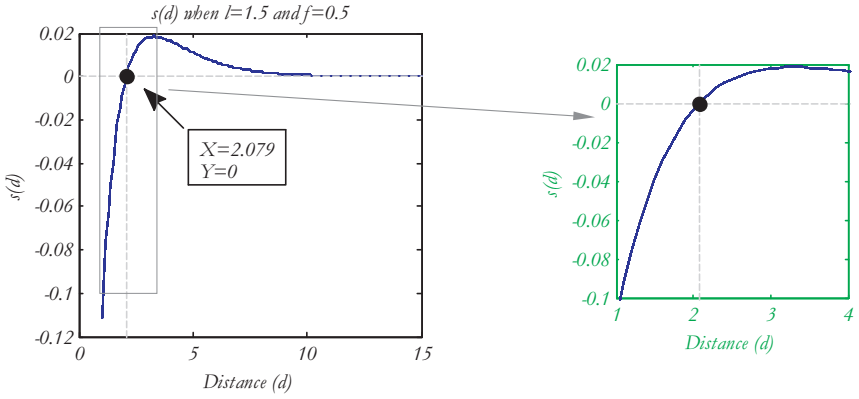


**Fig. 2.** (left) Function $s$ when $l$=1.5 and $f$=0.5 (right) range of function $s$ when $x$ is in [1,4].

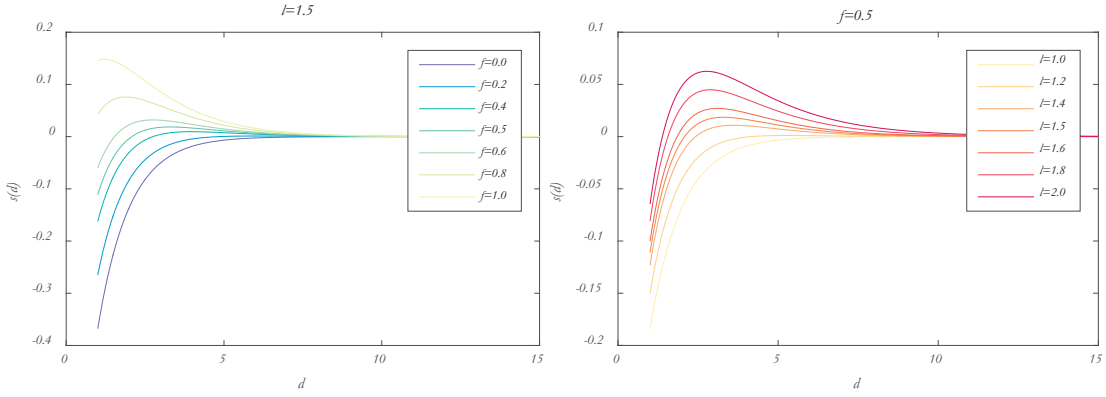


**Fig. 3.** Behaviour of the function $s$ when varying $l$ or $f$.

where $f$ indicates the intensity of attraction and $l$ is the attractive length scale.

The function $s$ is illustrated in Fig. 2 to show how it impacts on the social interaction (attraction and repulsion) of grasshoppers.

It may be seen in this figure that distances from 0 to 15 are considered. Repulsion occurs in the interval [0 2.079]. When a grasshopper is 2.079 units away from another grasshopper, there is neither attraction nor repulsion. This is called the comfort zone or comfortable distance. Fig. 2 also shows that the attraction increases from 2.079 unit of distance to nearly 4 and then gradually decreases. Changing the parameters $l$ and $f$ in Eq. (2.3) results in different social behaviours in artificial grasshoppers. To see the effects of these two parameters, the function $s$ is re-drawn in Fig. 3 varying $l$ and $f$ independently. This figure shows that the parameters $l$ and $f$ change comfort zone, attraction region, and repulsion region significantly. It should be noted that the attraction or repulsion regions are very small for some values ($l = 1.0$ or $f = 1.0$ for instance). From all these values we have chosen $l = 1.5$ and $f = 0.5$.

A conceptual model of the interactions between grasshoppers and the comfort zone using the function $s$ is illustrated in Fig. 4. It may be noted that, in simplified form, this social interaction was the motivating force in some earlier locust swarming models [32].

Although the function $s$ is able to divide the space between two grasshoppers into repulsion region, comfort zone, and attraction region, this function returns values close to zero with distances greater than 10 as Figs. 2 and 3 show. Therefore, this function is not able to apply strong forces between grasshoppers with large distances between them. To resolve this issue, we have mapped

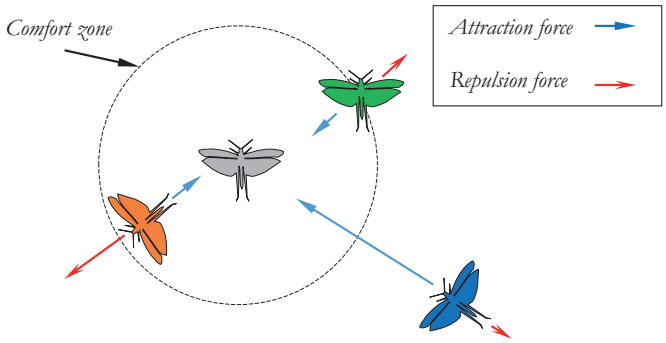


**Fig. 4.** Primitive corrective patterns between individuals in a swarm of grasshoppers.

the distance of grasshoppers in the interval of [1,4]. The shape of the function $s$ in this interval is shown in Fig. 2 (right).

The $G$ component in Eq. (2.1) is calculated as follows:

$$G_i = -g\widehat{e_g} \tag{2.4}$$

where $g$ is the gravitational constant and $\widehat{e_g}$ shows a unity vector towards the centre of earth.

The $A$ component in Eq. (2.1) is calculated as follows:

$$A_i = u\widehat{e_w} \tag{2.5}$$

where $u$ is a constant drift and $\widehat{e_w}$ is a unity vector in the direction of wind.

Nymph grasshoppers have no wings, so their movements are highly correlated with wind direction.

Substituting S, G, and A in Eq.(2.1), this equation can be expanded as follows:

$$X_i = \sum_{\substack{j=1 \\ j\neq i}}^{N} s\left(\left|x_j - x_i\right|\right)\frac{x_j - x_i}{d_{ij}} - g\widehat{e_g} + u\widehat{e_w} \tag{2.6}$$

where $s(r) = fe^{\frac{-r}{l}} - e^{-r}$ and $N$ is the number of grasshoppers.

Since nymph grasshoppers land on the ground, their position should not go below a threshold. However, we will not utilise this equation in the swarm simulation and optimisation algorithm because it prevents the algorithm from exploring and exploiting the search space around a solution. In fact, the model utilised for the swarm is in free space. Therefore, Eq. (2.6) is used and can simulate the interaction between grasshoppers in a swarm. The behaviour of two swarms in 2D and 3D space using this equation is illustrated in Figs. 5 and 6. In these two figures, 20 artificial grasshoppers are required to move over 10 units of time.

Fig. 5 shows how Eq. (2.6) brings the initial random population closer until they form a united, regulated swarm. After 10 units of time, all the grasshoppers reach the comfort zone and do not move anymore. The same behaviour is observed in a 3D space in Fig. 6. This shows that the mathematical model is able to simulate a swarm of grasshoppers in 2D, 3D, and hyper dimensional spaces.

However, this mathematical model cannot be used directly to solve optimisation problems, mainly because the grasshoppers quickly reach the comfort zone and the swarm does not converge to a specified point. A modified version of this equation is proposed as follows to solve optimisation problems:

$$X_i^d = c\left(\sum_{\substack{j=1 \\ j\neq i}}^{N} c\frac{ub_d - lb_d}{2} s\left(\left|x_j^d - x_i^d\right|\right)\frac{x_j - x_i}{d_{ij}}\right) + \widehat{T_d} \tag{2.7}$$

where $ub_d$ is the upper bound in the $D$th dimension, $lb_d$ is the lower bound in the $D$th dimension $s(r) = fe^{\frac{-r}{l}} - e^{-r}$, $\widehat{T_d}$ is the value of the $D$th dimension in the target (best solution found so far), and $c$ is a decreasing coefficient to shrink the comfort zone, repulsion zone, and attraction zone. Note that $S$ is almost similar to the $S$ component in Eq. (2.1). However, we do not consider gravity (no $G$ component) and assume that the wind direction ($A$ component) is always towards a target ($\widehat{T_d}$).

Eq. (2.7) shows that the next position of a grasshopper is defined based on its current position, the position of the target, and the position of all other grasshoppers. Note that the first component of this equation considers the location of the current grasshopper with respect to other grasshoppers. In fact, we have considered the status of all grasshoppers to define the location of search agents around the target. This is different to PSO as the most well-regarded swarm intelligence technique in the literature. In PSO, there are two vectors for each particle: position and velocity vector. However, there is only one position vector for every search agent in GOA. The other main difference between these two algorithms is that PSO updates the position of particles with respect to current position, personal best, and global best. However, GOA updates the position of a search agent based on its current position, global best, and the position of all other search agents. This means that in PSO none of the other particles contribute to updating the position of a particle, whereas GOA requires all search agents to get involved in defining the next position of each search agent.

It is also worth mentioning here that the adaptive parameter $c$ has been used twice in Eq. (2.7) for the following reasons:

- The first $c$ from the left is very similar to the inertial weight ($w$) in PSO. It reduces the movements of grasshoppers around the target. In other words, this parameter balances exploration and exploitation of the entire swarm around the target.
- The second $c$ decreases the attraction zone, comfort zone, and repulsion zone between grasshoppers. Considering the component $c\frac{ub_d - ld_d}{2}s(|x_j - x_i|)$ in the Eq. (2.7), $c\frac{ub_d - ld_d}{2}$ linearly decreases the space that the grasshoppers should explore and exploit. The component $s(|x_j - x_i|)$ indicates if a grasshopper should be repelled from (explore) or attracted to (exploitation) the target.

It should be noted that the inner $c$ contributes to the reduction of repulsion/attraction forces between *grasshoppers* proportional to the number of iterations, while the outer $c$ reduces the search coverage around the *target* as the iteration count increases.

In summary, the first term of Eq. (2.7), the sum, considers the position of other grasshoppers and implements the interaction of grasshoppers in nature. The second term, $\widehat{T_d}$, simulates their tendency to move towards the source of food. Also, the parameter $c$ simulates the deceleration of grasshoppers approaching the source of food and eventually consuming it. To provide more random behaviour, and as an alternative, both terms in Eq. (2.7) can be multiplied with random values. Also, individual terms can be multiplied with random values to provide random behaviour in either interaction of grasshoppers or tendency towards the food source.

The proposed mathematical formulations are able to explore and exploit the search space. However, there should be a mechanism to require the search agents to tune the level of exploration to exploitation. In nature, grasshoppers first move and search for foods locally because in larvae they have no wing. They then move freely in air and explore a much larger scale region. In stochastic optimisation algorithms, however, exploration comes first due to the need for finding promising regions of the search space. After finding promising regions, exploitation obliges search agents to

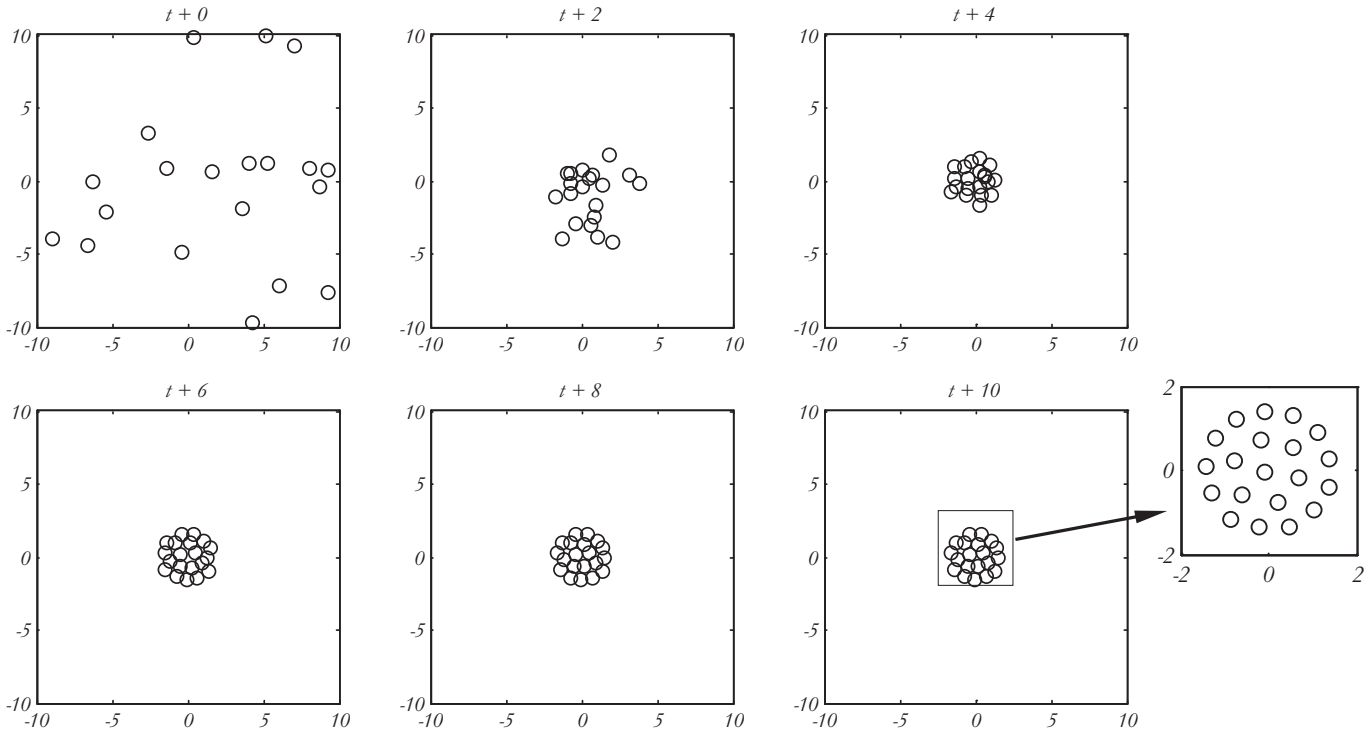


**Fig. 5.** Behaviour of swarm in a 2D space.

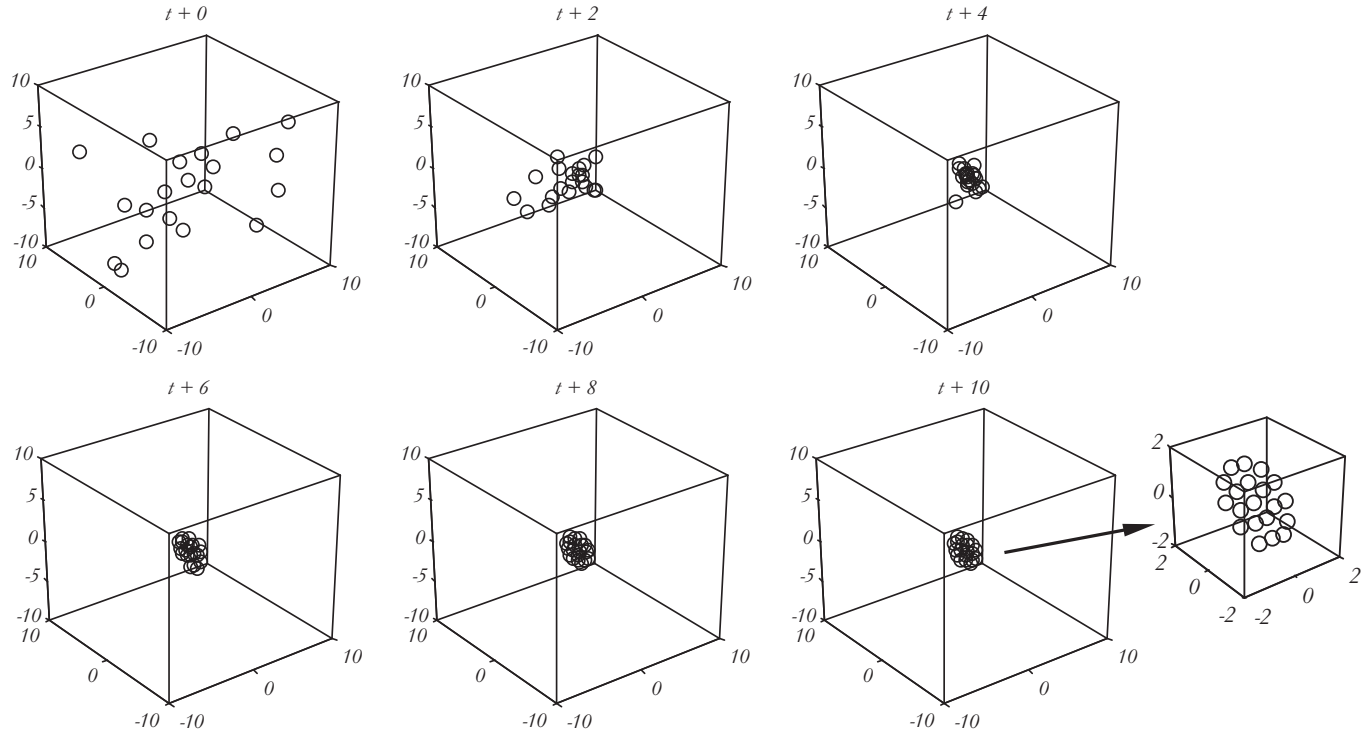


**Fig. 6.** Behaviour of swarm in a 3D space.

search locally to find an accurate approximation of the global optimum.

For balancing exploration and exploitation, the parameter $c$ is required to be decreased proportional to the number of iteration. This mechanism promotes exploitation as the iteration count increases. The coefficient $c$ reduces the comfort zone proportional to the number of iterations and is calculated as follows:

$$c = cmax - l\frac{cmax - cmin}{L} \tag{2.8}$$

where $cmax$ is the maximum value, $cmin$ is the minimum value, $l$ indicates the current iteration, and $L$ is the maximum number of iterations. In this work, we use 1 and 0.00001 for $cmax$ and $cmin$ respectively.

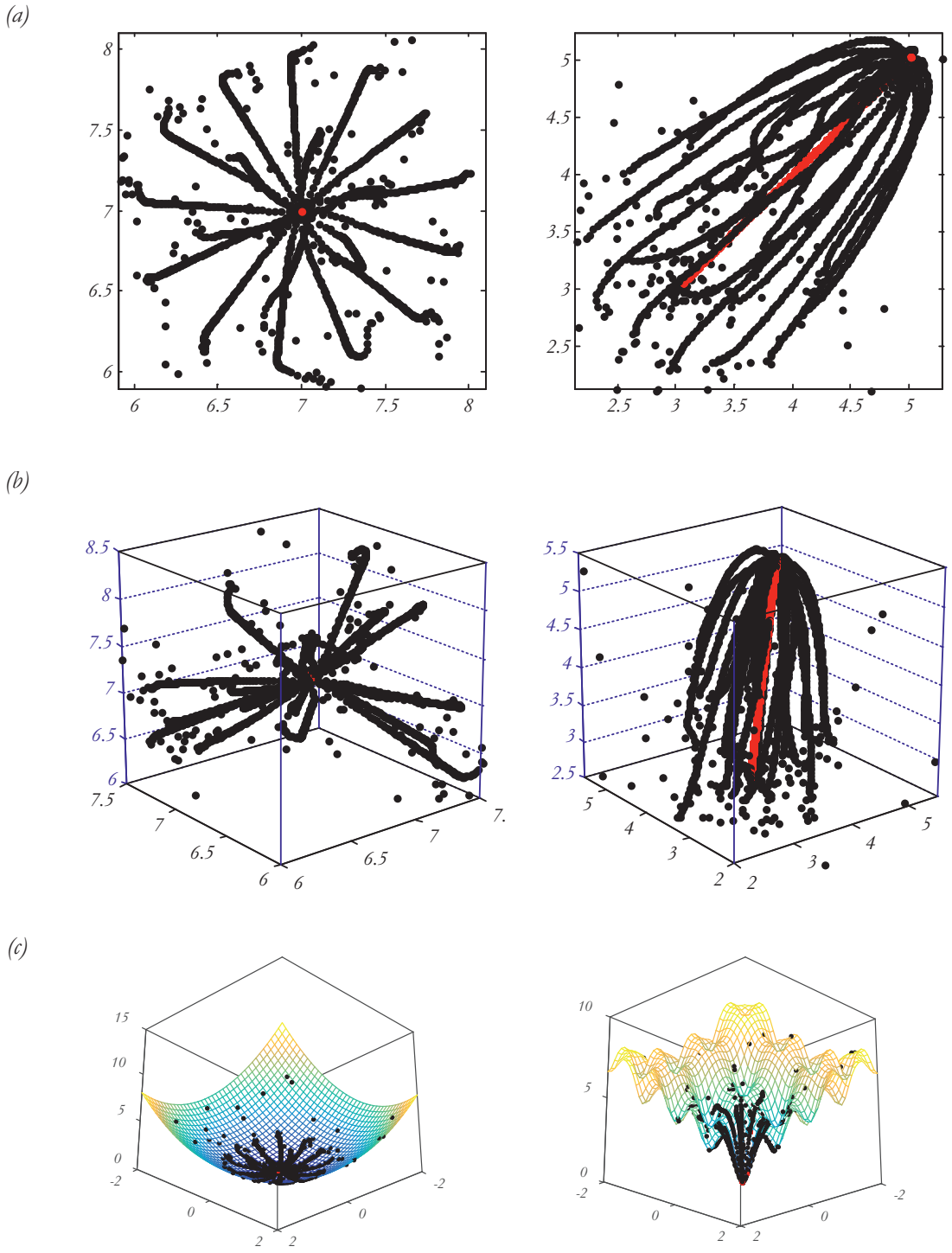


**Fig. 7.** (a) Behaviour of grasshoppers around a stationary and mobile target in 2D space and (b) 3D space (c) Behaviour of grasshoppers on a unimodal test function and a multi-modal test function.

The effect of this parameter on the movement and convergence of grasshoppers is illustrated in Fig. 7. The sub-figures illustrate the position history of grasshoppers over 100 iterations. We have performed the experiment on both stationary and mobile targets to see how the swarm moves towards and chases them. This figure shows that the swarm converges gradually towards a stationary target in both 2D and 3D spaces. This behaviour is due to reducing the comfort zone by the factor *c*. Fig. 7 also shows that the swarm properly chases a mobile target as well. This is due to the last component of Eq. (2.6) ($\widehat{T}_d$), in which grasshoppers are attracted towards the target. The interesting pattern is the gradual convergence of grasshoppers towards the target over the course of iteration, which is again due to decreasing the factor *c*. These behaviours will assist the GOA algorithm not to converge towards the target too quickly and consequently not to become trapped in local optima. In the last steps of optimisation, however, grasshoppers will converge towards the target as much as possible, which is essential in exploitation.

The above discussions show that the mathematical model proposed requires grasshoppers to move towards a target gradually

```
Initialize the swarm Xi (i = 1, 2, ..., n)
Initialize cmax, cmin, and maximum number of iterations
Calculate the fitness of each search agent
T=the best search agent
while (l < Max number of iterations)
    Update c using Eq. (2.8)
    for each search agent
            Normalize the distances between grasshoppers in [1,4]
            Update the position of the current search agent by the equation (2.7)
            Bring the current search agent back if it goes outside the boundaries
    end for
    Update T if there is a better solution
    l=l+1
end while
Return T
```

**Fig. 8.** Pseudo codes of the GOA algorithm.

over the course of iterations. In a real search space, however, there is no target because we do not know exactly where the global optimum, the main target, is. Therefore, we have to find a target for grasshoppers in each step of optimisation. In GOA, it is assumed that the fittest grasshopper (the one with the best objective value) during optimisation is the target. This will assist GOA to save the most promising target in the search space in each iteration and requires grasshoppers to move towards it. This is done with the hope of finding a better and more accurate target as the best approximation for the real global optimum in the search space. Fig. 7 includes two test functions and shows that the proposed model and target updating mechanism are effective in problems with unknown optimum as well.

The pseudo code of the GOA algorithm is shown in Fig. 8. The GOA starts optimisation by creating a set of random solutions. The search agents update their positions based on Eq. (2.7). The position of the best target obtained so far is updated in each iteration. In addition, the factor $c$ is calculated using Eq. (2.8) and the distances between grasshoppers are normalised in [1,4] in each iteration. Position updating is performed iteratively until the satisfaction of an end criterion. The position and fitness of the best target is finally returned as the best approximation for the global optimum.

Although the above simulations and discussions demonstrate the effectiveness of the GOA algorithm in finding the global optimum in a search space, the performance of the proposed algorithm is investigated by employing a set of mathematical functions and three challenging real problems in the next sections. Note that the source codes of the GOA algorithm can be found at http://www.alimirjalili.com/Projects.html and http://au.mathworks.com/matlabcentral/profile/authors/2943818-seyedali-mirjalili.

## 3. Results

This section first presents the test bed problems and performance metrics that are used to benchmark the performance of the proposed GOA algorithm. The experimental results are then provided and analysed in detail.

### 3.1. Experimental setup

In the field of stochastic optimisation, it is common to employ a set of mathematical test functions with known optima. Thus, the performance of different algorithms can be measured quantitatively. However, the characteristics of the test functions should be diverse to be able to draw a mature conclusion. In this work, three sets of test functions with different features are employed to confidently benchmark the performance of the proposed algorithm. The test functions are unimodal, multimodal, and composite [40–43]. The mathematical formulation of these test functions are available in the appendix.

As shown in Fig. 9, a unimodal test function has no local solutions and there is only one global optimum. The entire search space favours the global optima, so the convergence speed and exploitation of an algorithm can be benchmarked. Fig. 9 also shows that multi-modal and composite test functions have many local optima which make them highly suitable for benchmarking the performance of an algorithm in terms of local optima avoidance and exploration. Composite tests functions are usually more challenging than the multi-modal test functions and better mimic real search spaces. Therefore, the potential performance of an algorithm solving real problems may be inferred from such benchmarks.

For solving the test functions, 30 search agents and 500 iterations were employed. Each of the test functions was solved 30 times to generate the statistical results. Different performance indicators were utilised to quantitatively compare the algorithms: average and standard deviation of the best solutions obtained in the last iterations. Obviously, the lower the value of average and standard deviation, the greater the ability of an algorithm in avoiding local solutions and determining the global optimum. Qualitative results, including convergence curves, trajectory of grasshoppers, search history, and average fitness of population have been illustrated and analysed in the following subsection.

For verification of results, seven algorithms were employed from the literature including well-known and recent ones: PSO, SMS [25,26], BA [22], FPA [27], CS [23,24], FA [20,21], GA, DE, and Gravitational Search Algorithm (GSA) [44]. The initial controlling parameters of all algorithms are shown in Table 1.

### 3.2. Qualitative results and discussion

The first experiment was performed on the 2D version of some of the test functions using only 5 artificial grasshoppers. The main objective for this experiment was to observe the behaviour of the GOA qualitatively. Five diagrams have been drawn for each of the test functions in Fig. 10 in addition to the shape of test functions. These diagrams are:

- Search history: this diagram shows the location history of the artificial grasshoppers during optimisation.
- Attraction/repulsion rates: this diagram shows the number of times that all artificial grasshoppers attracted or repelled each other during optimisation.
- Trajectory of the first grasshopper in the first dimension: this diagram shows the value of the first variable of the first grasshopper in each iteration.
- Average fitness: this diagram indicates the average objective value of all grasshoppers in each iteration.
- Convergence curve: this diagram shows the objective value of the best solutions obtained so far (target) in each iteration.

As per the results in Fig. 10, grasshoppers tend to explore the promising regions of the search space and cluster around the global optima eventually. This pattern can be observed in unimodal, multimodal, and composite test functions. These results show that the GOA algorithm beneficially balances exploration and exploitation to drive the grasshoppers towards the global optimum. The rates of attraction and repulsion in the pie charts show that the grasshoppers interact differently on the test functions. They seem to attract each other more often on the unimodal test functions. This can clearly be observed in the pie charts for F1 and F4. This behaviour is fruitful because unimodal functions do not have local optima, so grasshopper can better determine the global optimum by moving towards the best solution obtained so far.

Another interesting pattern in the pie charts is the high repulsion rate between grasshoppers when solving multi-modal and composite test functions. This can be observed in the pie charts

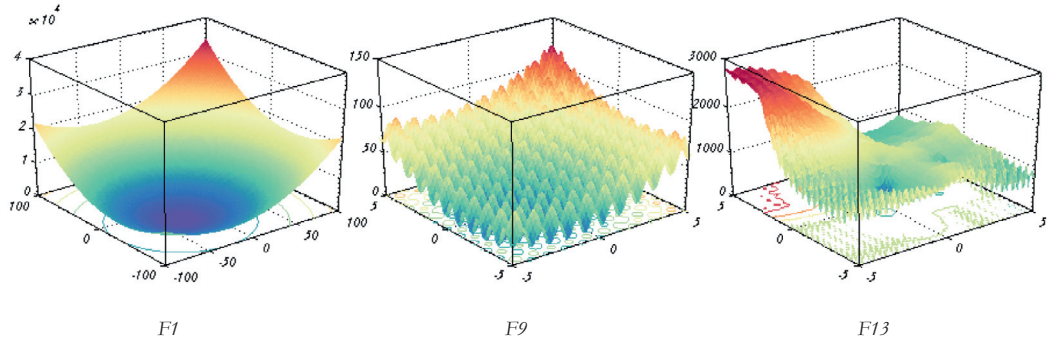


**Fig. 9.** An example of unimodal, multi-modal, and composite test functions.

**Table 1**
Initial values for the controlling parameters of algorithms.

| Algorithm | Parameter | Value |
|---|---|---|
| PSO | Topology | Fully connected |
| | Cognitive and social constants | 1.5, 1.5 |
| | Inertial weight | Linearly decreases from 0.6 to 0.3 |
| GA | Type | Real coded |
| | Selection | Roulette wheel |
| | Crossover | Single point (probability = 1) |
| | Mutation | Uniform (probability = 0.01) |
| DE | Crossover probability | 0.9 |
| | Differential weight | 0.5 |
| GSA | Rnorm, Rpower, alpha, and $G_0$ | 2, 1, 20, 100 |
| BA | Loudness ($A$), pulse rate ($r$) | 0.5, 0.5 |
| | Frequency min and max | 0, 2 |
| FPA | probability switch($p$) | 0.4 |
| SMS | Beta | [0.9, 0.5, 0.1] |
| | Alpha | [0.3, 0.05, 0] |
| | H | [0.9, 0.2, 0] |
| | Phases | [0.5, 0.1,−0.1] |
| FA | Alpha, beta, and gamma | 0.2, 1, 1 |

of F9, F14 and F18. This is due to the fact that repulsion is a key mechanism to avoid local solutions in the GOA algorithm and these results show that this algorithm prevents grasshoppers from local optima stagnation by high repulsion rates. It is worth mentioning here that the results show that the high repulsion rate does not negatively impact on the convergence. This is due to the adaptive parameter of GOA, which shrinks the repulsion area proportional to the number of iterations. Therefore, grasshoppers avoid local valleys in the initial steps of iteration and cluster around the global optimum in the final stages of optimisation. For the test functions with both unimodal and multi modal regions (F10 for instance), Fig. 10 shows that the repulsion rate is lower. These results again demonstrate that GOA efficiently balances exploration and exploitation to approximate the global optimum.

The trajectory curves in Fig. 10 show that the grasshoppers exhibit large, abrupt changes in the initial steps of optimisation. This is due to the high repulsion rate which causes exploration of the search space by GOA. It also can be seen that the fluctuation decreased gradually during optimisation, which is due to the adaptive comfort zone and attraction forces between the grasshoppers. This guarantees that the proposed GOA algorithm explores and exploits the search space and converges towards a point eventually. To confirm that this behaviour results in improving the fitness of grasshoppers, average fitness of grasshoppers and convergence curves are provided in Fig. 10. The curves clearly show descending behaviour on all of the test functions. This proves that GOA enhances the initial random population on the test functions and desirably improves the accuracy of the approximated optimum over the course of iterations.

### 3.3. Quantitative results and discussion

The above discussed results qualitatively demonstrated that the GOA is able to solve optimisation problems. However, the test functions were of 2 variables and qualitative results cannot tell us how much better this algorithm is compared to current ones. In order to show the merits of GOA quantitatively, this subsection solved the test functions with 30 dimensions and presents/discusses the results quantitatively. The experimental results are provided in Tables 2, 3, and 4 for unimodal, multi-modal, and composite test functions. Note that the results are normalised between 0 and 1 for all the test functions due to the different domain/range of test functions. This assist us in conveniently comparing the results on different test functions as well.

As per the results in Table 2, the GOA algorithm shows the best results when solving unimodal test functions. The results of this algorithm are substantially better in more than half of the unimodal test functions, showing the high performance of this algorithm. Unimodal test functions have only one global optimum, so the results clearly show that the GOA algorithm benefits from high exploitation ability.

The results in Table 3 are consistent with those in Table 2, in which the GOA algorithm tends to significantly outperform others in both of the performance metrics. The results of this algorithm are again remarkably superior in the majority of multi-modal test functions. Since the multi-modal test functions have a significant number of local solutions, these results quantitatively show the effectiveness of the proposed algorithm in avoiding local solutions during optimisation.

The results of the algorithms on composite test functions are presented in Table 4. These results show that the GOA algorithm provides very competitive results compared to other algorithms. Composite test functions are even more challenging than the multi-modal ones and require a proper balance between exploration and exploitation. Therefore, it can be stated that the GOA is able to balance exploration and exploitation properly for solving such challenging problems.

Comparing algorithms based on average and standard deviation over 30 independent runs does not compare each of the runs. Therefore, it is still possible that the superiority occurs by chance despite its low probability in 30 runs. In order to compare the results of each run and decide on the significance of the results, the Wilcoxon statistical test was performed at 5% significance level and the p-values are reported in Table 5. For the statistical test, the best algorithm in each test function is chosen and compared with other algorithms independently. For example, if the best algorithm is GOA, a pairwise comparison is done between GOA/PSO, GOA/GSA, GOA/BA, and so on. Note that since the best algorithm cannot be compared with itself, N/A has been

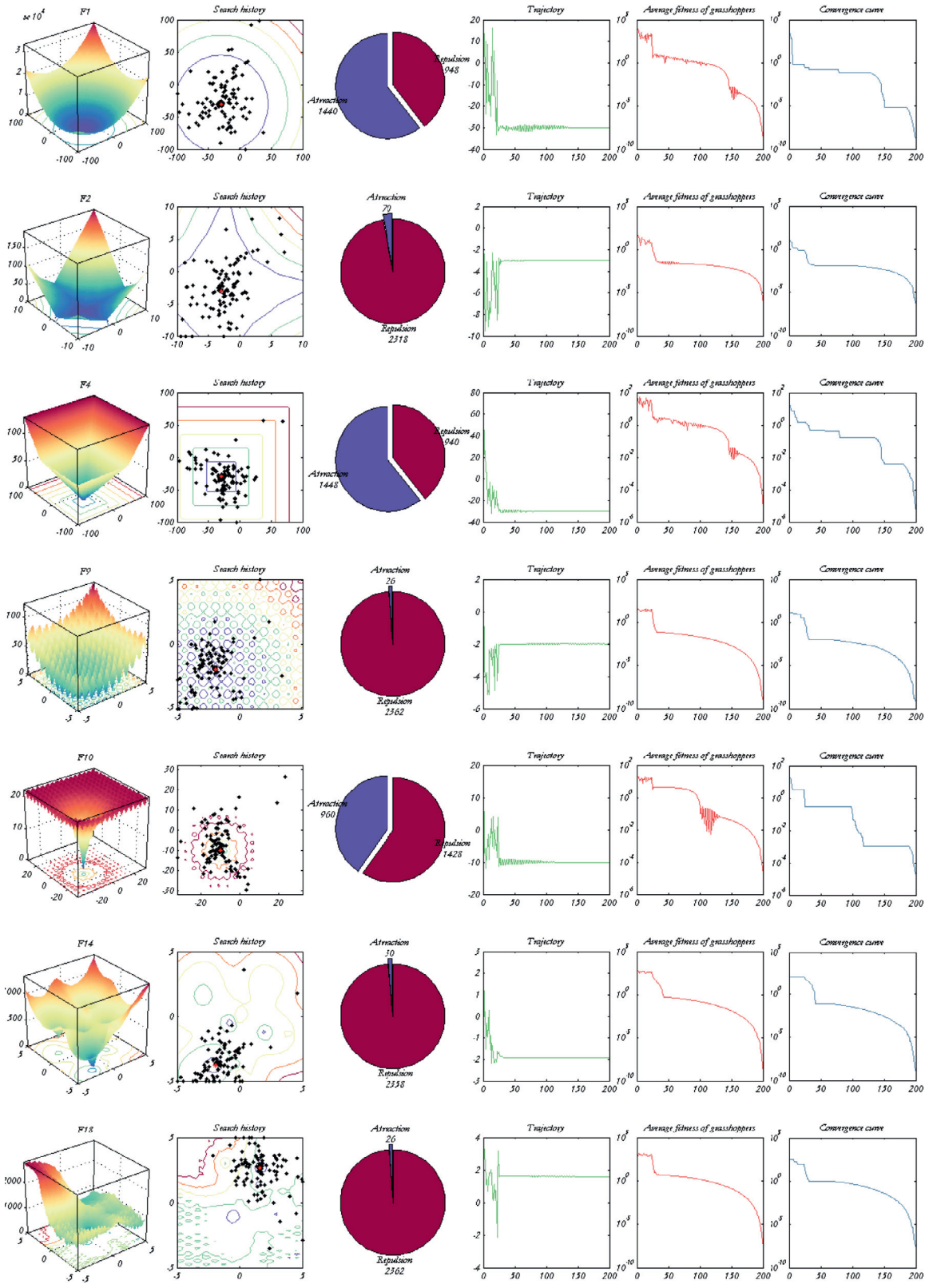


**Fig. 10.** Behaviour of GOA on the 2D benchmark problems.

written for the best algorithm in each function which stands for Not Applicable.

As per the results in this table, p-values are mostly less than 0.05 for the GOA, which demonstrates that the superiority of this algorithm is statistically significant. For the F3 function the results show the FPA is not significantly superior to GOA. Overall, these results show that GOA is able to outperform other algorithms in the literature. According to the NFL theorem, therefore, it has the potential to solve problems (of the types tested) that cannot be solved efficiently by other algorithms.

**Table 2**
Results of unimodal benchmark functions.

| F | GOA | | PSO | | GSA | | BA | |
|---|---|---|---|---|---|---|---|---|
| | *Ave* | *std* | *ave* | *std* | *ave* | *std* | *ave* | *std* |
| F1 | 0.0000 | 0.0000 | 0.2391 | 0.5622 | 0.0002 | 0.0012 | 0.9882 | 1.0000 |
| F2 | 0.0020 | 0.0010 | 0.0097 | 0.0013 | 0.0000 | 0.0000 | 1.0000 | 1.0000 |
| F3 | 0.0010 | 0.0203 | 0.2613 | 0.3547 | 0.0328 | 0.0395 | 1.0000 | 1.0000 |
| F4 | 0.0000 | 0.0000 | 0.4767 | 0.4730 | 0.3244 | 0.5119 | 0.9148 | 1.0000 |
| F5 | 0.0000 | 0.0000 | 0.0386 | 0.0944 | 0.0000 | 0.0000 | 1.0000 | 1.0000 |
| F6 | 0.0000 | 0.0000 | 0.7786 | 0.4808 | 0.3825 | 0.2231 | 1.0000 | 1.0000 |
| F7 | 0.0000 | 0.0000 | 0.1349 | 0.1648 | 0.0226 | 0.0763 | 1.0000 | 1.0000 |

| F | FPA | | SMS | | FA | | GA | |
|---|---|---|---|---|---|---|---|---|
| | *ave* | *std* | *ave* | *std* | *ave* | *std* | *ave* | *std* |
| F1 | 0.0329 | 0.0784 | 1.0000 | 0.4478 | 0.1581 | 0.0748 | 0.4121 | 0.5202 |
| F2 | 0.0131 | 0.0007 | 0.0157 | 0.0008 | 0.0076 | 0.0001 | 0.0100 | 0.0003 |
| F3 | 0.0000 | 0.0000 | 0.3486 | 0.0651 | 0.0617 | 0.0160 | 0.2022 | 0.0710 |
| F4 | 0.3219 | 0.4215 | 1.0000 | 0.7232 | 0.3796 | 0.2116 | 0.7245 | 0.2384 |
| F5 | 0.0060 | 0.0345 | 0.3901 | 0.4781 | 0.0068 | 0.0068 | 0.0746 | 0.0931 |
| F6 | 0.0088 | 0.0189 | 0.7025 | 0.5891 | 0.0414 | 0.0190 | 0.1933 | 0.1932 |
| F7 | 0.0386 | 0.0459 | 0.0036 | 0.0022 | 0.0590 | 0.0194 | 0.4416 | 0.2028 |

**Table 3**
Results of multimodal benchmark functions.

| F | GOA | | PSO | | GSA | | BA | |
|---|---|---|---|---|---|---|---|---|
| | *ave* | *std* | *ave* | *std* | *ave* | *std* | *ave* | *std* |
| F8 | 1.0000 | 0.0002 | 0.7425 | 0.0016 | 0.8473 | 0.0020 | 0.0148 | 1.0000 |
| F9 | 0.0000 | 0.0007 | 0.6520 | 1.0000 | 0.1361 | 0.2722 | 0.7022 | 0.7517 |
| F10 | 0.0975 | 1.0000 | 0.6140 | 0.2426 | 0.0000 | 0.0000 | 0.9665 | 0.1155 |
| F11 | 0.0000 | 0.0000 | 0.8184 | 0.3512 | 1.0000 | 0.5790 | 0.9912 | 1.0000 |
| F12 | 0.0000 | 0.0007 | 0.4689 | 0.8147 | 0.0577 | 0.4246 | 0.6892 | 0.9635 |
| F13 | 0.0000 | 0.0000 | 0.0973 | 0.1647 | 0.1603 | 0.0890 | 1.0000 | 1.0000 |

| F | FPA | | SMS | | FA | | GA | |
|---|---|---|---|---|---|---|---|---|
| | *ave* | *std* | *ave* | *std* | *ave* | *std* | *ave* | *std* |
| F8 | 0.0381 | 0.0006 | 0.5613 | 0.0049 | 0.6140 | 0.0000 | 0.0000 | 0.0006 |
| F9 | 0.6568 | 0.4179 | 0.8628 | 0.2633 | 0.8377 | 0.0329 | 1.0000 | 0.0000 |
| F10 | 0.7170 | 0.3848 | 1.0000 | 0.0666 | 0.7078 | 0.0410 | 0.8628 | 0.1415 |
| F11 | 0.0124 | 0.0058 | 0.6746 | 0.7789 | 0.0548 | 0.0070 | 0.1941 | 0.2865 |
| F12 | 0.0237 | 0.1907 | 0.1140 | 0.0000 | 0.2442 | 1.0000 | 1.0000 | 0.4363 |
| F13 | 0.3766 | 0.1566 | 0.9609 | 0.2394 | 0.1119 | 0.1510 | 0.4446 | 0.0798 |

**Table 4**
Results of composite benchmark functions.

| F | GOA | | PSO | | GSA | | BA | |
|---|---|---|---|---|---|---|---|---|
| | *Ave* | *std* | *ave* | *std* | *ave* | *std* | *ave* | *std* |
| F14 | 0.0000 | 0.3386 | 0.6083 | 1.0000 | 0.0840 | 0.2977 | 1.0000 | 0.5714 |
| F15 | 0.4892 | 0.7182 | 0.4236 | 0.7929 | 0.0672 | 0.5226 | 1.0000 | 1.0000 |
| F16 | 0.0000 | 0.0000 | 0.4651 | 0.6805 | 0.4799 | 0.8414 | 1.0000 | 1.0000 |
| F17 | 0.8169 | 1.0000 | 0.3241 | 0.2970 | 0.0000 | 0.6439 | 1.0000 | 0.6905 |
| F18 | 0.0000 | 0.0064 | 0.3122 | 1.0000 | 0.0581 | 0.2503 | 1.0000 | 0.8953 |
| F19 | 0.7863 | 0.9355 | 1.0000 | 0.0000 | 0.9391 | 0.3063 | 0.9097 | 0.4190 |

| F | FPA | | SMS | | FA | | GA | |
|---|---|---|---|---|---|---|---|---|
| | *ave* | *std* | *ave* | *std* | *ave* | *std* | *ave* | *std* |
| F14 | 0.0008 | 0.0570 | 0.5604 | 0.4830 | 0.5035 | 0.6008 | 0.3799 | 0.0000 |
| F15 | 0.0000 | 0.4822 | 0.5097 | 0.5559 | 0.5730 | 0.9765 | 0.1338 | 0.0000 |
| F16 | 0.3381 | 0.0759 | 0.8914 | 0.5077 | 0.4921 | 0.0922 | 0.6820 | 0.1783 |
| F17 | 0.1395 | 0.0348 | 0.6594 | 0.0383 | 0.3264 | 0.4911 | 0.3660 | 0.0000 |
| F18 | 0.3249 | 0.9194 | 0.3144 | 0.4097 | 0.3160 | 0.3885 | 0.1347 | 0.0000 |
| F19 | 0.0000 | 0.0702 | 0.4257 | 0.8595 | 0.7068 | 1.0000 | 0.0211 | 0.1720 |

To further show the effectiveness of the proposed GOA algorithm, we have solved more challenging test functions and compared the results with the most popular algorithms in the literature. The test functions are 25 taken from the CEC2005 special session [45]. These test functions are the most challenging test functions in the literature and can be found in the appendix. The results are compared to PSO, GA, DE, GSA, BA, FPA, and FA as the most well-known and recent algorithms in the literature. The results are again normalised in [0,1] and presented in Tables 6 and 7.

Inspecting the results in Table 6, it is evident that the proposed GOA algorithm outperforms other algorithms on the majority of the CEC2005 test functions. The p-values in Table 7 show that the superiority of GOA is statistically significant. Comparison

**Table 5**
P-values obtained from the Wilcoxon ranksum test.

| TP | GOA | PSO | GSA | BA | FPA | SMS | FA | GA |
|---|---|---|---|---|---|---|---|---|
| F1 | N/A | 0.000183 | N/A | 0.000183 | 0.000183 | 0.000183 | 0.000183 | 0.000183 |
| F2 | 0.002827 | 0.000183 | N/A | 0.000183 | 0.000183 | 0.000183 | 0.000183 | 0.000183 |
| F3 | 0.140465 | 0.000183 | 0.000183 | 0.000183 | N/A | 0.000183 | 0.000183 | 0.000183 |
| F4 | N/A | 0.000183 | 0.000183 | 0.000183 | 0.000183 | 0.000183 | 0.000183 | 0.000183 |
| F5 | N/A | 0.000183 | 0.241322 | 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 | N/A | 0.000183 | 0.00033 | 0.000183 | 0.000183 | 0.000583 | 0.000183 | 0.000183 |
| F8 | 0.000183 | 0.000183 | 0.000183 | N/A | 0.000183 | 0.000183 | 0.000183 | 0.000183 |
| F9 | N/A | 0.000183 | 0.01133 | 0.000183 | 0.000183 | 0.000183 | 0.000183 | 0.000183 |
| F10 | 0.000183 | 0.000183 | N/A | 0.000183 | 0.000183 | 0.000183 | 0.000183 | 0.000183 |
| F11 | N/A | 0.000183 | 0.000183 | 0.000183 | 0.001315 | 0.000183 | 0.000183 | 0.000183 |
| F12 | N/A | 0.000183 | 0.000183 | 0.000183 | 0.000183 | 6.39e-5 | 0.000183 | 0.000183 |
| F13 | N/A | 0.000183 | 0.000183 | 0.000183 | 0.000183 | 6.39e-5 | 0.000183 | 0.000183 |
| F14 | N/A | 0.001315 | 0.009108 | 0.000246 | 0.025748 | 0.001706 | 0.001315 | 0.002827 |
| F15 | 0.001008 | 0.004586 | 0.791337 | 0.000246 | N/A | 0.001008 | 0.000769 | 0.002827 |
| F16 | N/A | 0.000246 | 0.00044 | 0.000183 | 0.000183 | 0.000183 | 0.000183 | 0.000183 |
| F17 | 0.001008 | 0.002827 | N/A | 0.001008 | 0.002827 | 0.002827 | 0.002202 | 0.002827 |
| F18 | N/A | 0.01133 | 0.472676 | 0.000183 | 0.025748 | 0.000183 | 0.000183 | 0.000183 |
| F19 | 0.025748 | 0.000183 | 0.000183 | 0.000183 | N/A | 0.00033 | 0.00044 | 0.021134 |

with some of the algorithms provide p-values greater than 0.05 occasionally. This shows that the GOA algorithm is not significantly better on those functions. Also, GOA provides very competitive results on the F12_CEC2005, F16_CEC2005, and F24_CEC2005 test functions, since the high p-values show GSA was not significantly better. Since CEC test functions are very challenging and mimic different difficulties of a real search space, these results strongly demonstrate the merits of the proposed GOA algorithm compared to other algorithms in the literature.

To sum up, the discussions and findings of this section clearly demonstrate the quality of exploration, local optima avoidance, exploitation, and convergence of the GOA algorithm. The high exploration and local optima avoidance of this algorithm originates from the high repulsion rate between grasshoppers. The repulsive force requires grasshoppers to avoid each other and explore the search space extensively. This is the main reason for high local optima avoidance of GOA as well. Exploitation and convergence are encouraged by the attraction forces between the grasshoppers, and the adaptive comfort zone. High attractive forces between grasshoppers drive them quickly towards the best solution obtained so far. The adaptive comfort zone coefficient decreases proportional to the number of iterations, generating smaller repulsion forces and emphasising exploitation. The adaptive behaviour of the comfort zone coefficient also results in a proper balance between exploration and exploitation.

Although these findings strongly suggest that GOA is able to solve real problems, in the following section we use three real problems in the field of structural design to demonstrate and confirm the applicability of this algorithm in solving real problems with unknown search spaces.

## 4. Real applications

Solving structural design problems using stochastic optimisation techniques has been a popular research direction in the literature [46–54]. This section solves three of the conventional structural design problems using the proposed GOA algorithm.

### 4.1. Three-bar truss design problem

This structural design problem is one of the most widely-used case studies in the literature [55,56]. This problem is formulated as follows:

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

Fig. 11 shows the shape of this truss and the forces applied. As this figure and the problem formulation show, there are two structural parameters: the area of bars 1 and 3 and area of bar 2. The objective is to minimise the weight of the truss. This problem is subject to several constraints as well: stress, deflection, and buckling constraints.

The proposed GOA with 20 search agents and 650 iterations was employed on this problem. Since this problem is a constrained problem, a constraint handling method needed to be integrated with GOA. For the sake of simplicity, a death penalty has been utilised. It penalises the search agents that violate any of the constraints at any level with a large objective value. For verification, the results are compared to ALO, DEDS, PSO-DE, MBA, Ray and Sain, and Tsa methods and presented in Table 8. This table shows the optimal values for both variables and weight.

Inspecting the results of algorithms on this problem, it is evident that GOA managed to show very competitive results compared to ALO, DEDS, PSO-DE, and MBA with a better maximum function evaluation. Also, this algorithm outperforms the rest of the algorithms significantly. These results show that the GOA algorithm is able to handle the difficulties of a constrained search space efficiently.

### 4.2. Cantilever beam design problem

This is another popular structural design problem in the literature formulated as follows:

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

**Table 6**
Results on CEC benchmark functions.

| F | GOA | | PSO | | GA | | DE | |
|---|---|---|---|---|---|---|---|---|
| | *Ave* | *std* | *ave* | *std* | *ave* | *std* | *ave* | *std* |
| F1_CEC2005 | 0.0000 | 0.0000 | 0.6040 | 1.0000 | 0.4972 | 0.8825 | 0.0424 | 0.0950 |
| F2_CEC2005 | 0.0000 | 0.0000 | 0.3341 | 0.1579 | 0.4860 | 0.1481 | 0.3483 | 0.1134 |
| F3_CEC2005 | 0.0037 | 0.0057 | 0.2299 | 0.2852 | 0.3130 | 0.0441 | 0.1311 | 0.0801 |
| F4_CEC2005 | 0.0000 | 0.0127 | 0.4569 | 0.3523 | 0.4807 | 0.0958 | 0.3726 | 0.0473 |
| F5_CEC2005 | 0.0000 | 0.0602 | 0.7003 | 0.4740 | 0.4788 | 0.0000 | 0.2737 | 0.0513 |
| F6_CEC2005 | 0.0000 | 0.0000 | 0.3078 | 0.1878 | 0.2109 | 0.2577 | 0.0037 | 0.0016 |
| F7_CEC2005 | 0.0000 | 0.0000 | 0.8524 | 0.6681 | 0.3416 | 0.3029 | 0.0759 | 0.0806 |
| F8_CEC2005 | 0.5505 | 0.7277 | 1.0000 | 0.0000 | 0.9702 | 0.1651 | 0.9816 | 0.4399 |
| F9_CEC2005 | 0.0000 | 0.0356 | 0.5743 | 0.8935 | 1.0000 | 0.3927 | 0.6055 | 0.1518 |
| F10_CEC2005 | 0.0000 | 0.0000 | 0.5391 | 1.0000 | 0.7874 | 0.0628 | 0.4904 | 0.1780 |
| F11_CEC2005 | 0.7570 | 1.0000 | 0.6283 | 0.8342 | 0.9965 | 0.3326 | 1.0000 | 0.0000 |
| F12_CEC2005 | 0.1087 | 0.3460 | 0.3531 | 1.0000 | 1.0000 | 0.4454 | 0.6698 | 0.2908 |
| F13_CEC2005 | 0.0000 | 0.0046 | 0.1030 | 0.0580 | 0.4551 | 0.0808 | 0.1484 | 0.0091 |
| F14_CEC2005 | 0.0154 | 0.0855 | 0.0000 | 0.9627 | 0.6053 | 0.2766 | 0.5800 | 0.0083 |
| F15_CEC2005 | 0.3527 | 1.0000 | 0.5065 | 0.8562 | 0.6289 | 0.2734 | 0.4217 | 0.3437 |
| F16_CEC2005 | 0.1520 | 0.8379 | 0.3316 | 0.7594 | 0.3718 | 0.0825 | 0.1878 | 0.0000 |
| F17_CEC2005 | 0.0000 | 0.7666 | 0.4744 | 1.0000 | 0.2712 | 0.0249 | 0.1401 | 0.0000 |
| F18_CEC2005 | 0.0000 | 0.0000 | 0.3981 | 0.7095 | 0.5096 | 0.2305 | 0.2369 | 0.2265 |
| F19_CEC2005 | 0.0000 | 0.0000 | 0.3494 | 0.3857 | 0.4993 | 0.2066 | 0.2122 | 0.0501 |
| F20_CEC2005 | 0.0000 | 0.0000 | 0.4166 | 0.8314 | 0.6018 | 0.3811 | 0.2662 | 0.0663 |
| F21_CEC2005 | 0.0000 | 0.0416 | 0.7469 | 0.0314 | 0.8364 | 0.0467 | 0.4840 | 0.3295 |
| F22_CEC2005 | 0.0000 | 0.0000 | 0.4000 | 0.7118 | 0.6116 | 0.2260 | 0.3587 | 0.1408 |
| F23_CEC2005 | 1.0000 | 0.3113 | 0.0000 | 0.6942 | 0.0608 | 0.7502 | 0.2773 | 0.4302 |
| F24_CEC2005 | 1.0000 | 0.0724 | 0.0000 | 0.7804 | 0.0458 | 0.7228 | 0.1298 | 0.2603 |
| F25_CEC2005 | 0.3449 | 0.0075 | 0.8530 | 0.0765 | 0.8299 | 0.0035 | 0.4108 | 0.0442 |

| F | GSA | | BA | | FPA | | FA | |
|---|---|---|---|---|---|---|---|---|
| | *ave* | *std* | *ave* | *std* | *ave* | *std* | *ave* | *std* |
| F1_CEC2005 | 0.4318 | 0.5622 | 1.0000 | 0.8805 | 0.1184 | 0.3697 | 0.1076 | 0.1120 |
| F2_CEC2005 | 0.2614 | 0.0257 | 1.0000 | 1.0000 | 0.0615 | 0.0320 | 0.1705 | 0.0191 |
| F3_CEC2005 | 0.1419 | 0.1081 | 1.0000 | 1.0000 | 0.0000 | 0.0000 | 0.0620 | 0.0640 |
| F4_CEC2005 | 0.8724 | 0.4499 | 1.0000 | 1.0000 | 0.0651 | 0.0685 | 0.0661 | 0.0000 |
| F5_CEC2005 | 0.8225 | 0.0534 | 1.0000 | 1.0000 | 0.2707 | 0.3970 | 0.1041 | 0.0675 |
| F6_CEC2005 | 0.1270 | 0.0762 | 1.0000 | 1.0000 | 0.0149 | 0.0323 | 0.0104 | 0.0067 |
| F7_CEC2005 | 0.9108 | 0.5975 | 1.0000 | 1.0000 | 0.0708 | 0.1859 | 0.0725 | 0.0476 |
| F8_CEC2005 | 0.2817 | 1.0000 | 0.0000 | 0.1382 | 0.9719 | 0.0924 | 0.9866 | 0.1094 |
| F9_CEC2005 | 0.4704 | 1 | 0.7013 | 0.3595 | 0.6643 | 0.3360 | 0.7155 | 0.0000 |
| F10_CEC2005 | 0.2536 | 0.7690 | 1.0000 | 0.8251 | 0.5307 | 0.9560 | 0.4252 | 0.0945 |
| F11_CEC2005 | 0.0000 | 0.4116 | 0.9638 | 0.2210 | 0.7825 | 0.0509 | 0.9867 | 0.0856 |
| F12_CEC2005 | 0.0000 | 0.1006 | 0.1440 | 0.6132 | 0.2963 | 0.0000 | 0.4474 | 0.4266 |
| F13_CEC2005 | 0.0516 | 0.0331 | 1.0000 | 1.0000 | 0.0824 | 0.0000 | 0.1328 | 0.0061 |
| F14_CEC2005 | 1.0000 | 0.1088 | 0.7244 | 1.0000 | 0.4733 | 0.0000 | 0.4933 | 0.0425 |
| F15_CEC2005 | 0.0000 | 0.0000 | 1.0000 | 0.8301 | 0.3127 | 0.1340 | 0.4956 | 0.8700 |
| F16_CEC2005 | 0.0000 | 1.0000 | 1.0000 | 0.9453 | 0.1872 | 0.1099 | 0.1364 | 0.0588 |
| F17_CEC2005 | 0.1055 | 0.5209 | 1.0000 | 0.6007 | 0.0966 | 0.1335 | 0.1530 | 0.4022 |
| F18_CEC2005 | 0.0306 | 0.0997 | 1.0000 | 1.0000 | 0.1485 | 0.4907 | 0.1162 | 0.0744 |
| F19_CEC2005 | 0.023 | 0.0478 | 1.0000 | 1.0000 | 0.1259 | 0.1073 | 0.0975 | 0.0111 |
| F20_CEC2005 | 0.056 | 0.3519 | 1.0000 | 1.0000 | 0.1484 | 0.1490 | 0.1075 | 0.0089 |
| F21_CEC2005 | 0.2677 | 1 | 1.0000 | 0.1528 | 0.6712 | 0.0793 | 0.5613 | 0.0000 |
| F22_CEC2005 | 0.1514 | 0.4313 | 1.0000 | 1.0000 | 0.2646 | 0.2646 | 0.2352 | 0.0618 |
| F23_CEC2005 | 0.9926 | 0.0000 | 0.4327 | 1.0000 | 0.3589 | 0.5613 | 0.0206 | 0.4696 |
| F24_CEC2005 | 0.5784 | 0.0000 | 0.0691 | 1.0000 | 0.3426 | 0.1616 | 0.0254 | 0.1246 |
| F25_CEC2005 | 0.0000 | 1.0000 | 1.0000 | 0.0000 | 0.3751 | 0.0623 | 0.2689 | 0.0595 |

Variable range $0.01 \leq x_1,\ x_2,\ x_3,\ x_4,\ x_5 \leq 100$,

Fig. 12 shows that the cantilever beam is built using five, hollow, square-section, box girders, and the lengths of those girders are the design parameters for this problem. There is also one constraint for this problem. The GOA algorithm with 20 search agents and a maximum of 650 iterations is employed to determine the optimum for this problem. The results are presented and compared to ALO, MMA, GCA_I, GCA-II, CS, and SOS for verification in Table 9.

The results in Table 9 show that GOA finds the second best optimal weight. However, this algorithm provides the lowest number maximum function evaluation.

### 4.3. 52-bar truss design

In this problem, the objective is to minimise the weight of a truss with 52 bars and 20 nodes. As shown in Fig. 13, four of the nodes are fixed and the bars are classified in 12 groups as follows, which are the main parameters to be optimised for this problem:

- *Group* 1: $A_1, A_2, A_3, A_4$
- *Group* 2: $A_5, A_6, A_7, A_8, A_9, A_{10}$
- *Group* 3: $A_{11}, A_{12}, A_{13}$
- *Group* 4: $A_{14}, A_{15}, A_{16}, A_{17}$
- *Group* 5: $A_{18}, A_{19}, A_{20}, A_{21}, A_{22}, A_{23}$
- *Group* 6: $A_{24}, A_{25}, A_{26}$
- *Group* 7: $A_{27}, A_{28}, A_{29}, A_{30}$

**Table 7**
P-values obtained from the Wilcoxon ranksum test.

| TP | GOA | PSO | GA | DE | GSA | BA | FPA | FA |
|---|---|---|---|---|---|---|---|---|
| F1_CEC2005 | N/A | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 |
| F2_CEC2005 | N/A | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0159 | 0.0079 |
| F3_CEC2005 | 0.2222 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 1.0000 | 0.0079 |
| F4_CEC2005 | N/A | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0952 | 0.0317 |
| F5_CEC2005 | N/A | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 |
| F6_CEC2005 | N/A | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 |
| F7_CEC2005 | N/A | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 |
| F8_CEC2005 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 1.0000 | 0.0079 | 0.0079 |
| F9_CEC2005 | N/A | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 |
| F10_CEC2005 | N/A | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 |
| F11_CEC2005 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | N/A | 0.0079 | 0.0079 | 0.0079 |
| F12_CEC2005 | 0.0556 | 0.0079 | 0.0079 | 0.0079 | N/A | 0.0556 | 0.0079 | 0.0079 |
| F13_CEC2005 | N/A | 0.0079 | 0.0079 | 0.0079 | 0.0159 | 0.0079 | 0.0079 | 0.0079 |
| F14_CEC2005 | 0.6905 | N/A | 0.0556 | 0.0952 | 0.0079 | 0.0317 | 0.1508 | 0.1508 |
| F15_CEC2005 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | N/A | 0.0079 | 0.0079 | 0.0079 |
| F16_CEC2005 | 0.2222 | 0.0952 | 0.0079 | 0.6905 | N/A | 0.0079 | 0.6905 | 0.6905 |
| F17_CEC2005 | N/A | 0.1508 | 0.1508 | 0.8413 | 0.3095 | 0.0079 | 1.0000 | 0.4206 |
| F18_CEC2005 | N/A | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 |
| F19_CEC2005 | N/A | 0.0079 | 0.0079 | 0.0079 | 0.0159 | 0.0079 | 0.0079 | 0.0079 |
| F20_CEC2005 | N/A | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 |
| F21_CEC2005 | N/A | 0.0079 | 0.0079 | 0.0079 | 0.0952 | 0.0079 | 0.0079 | 0.0079 |
| F22_CEC2005 | N/A | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | 0.0079 |
| F23_CEC2005 | 0.0317 | 0.0079 | 0.0079 | 0.0079 | N/A | 0.0079 | 0.0079 | 0.0079 |
| F24_CEC2005 | 0.4206 | 0.0079 | 0.0079 | 0.0079 | N/A | 0.0079 | 0.0079 | 0.0079 |
| F25_CEC2005 | 0.0079 | 0.0079 | 0.0079 | 0.0079 | N/A | 0.0079 | 0.0079 | 0.4206 |

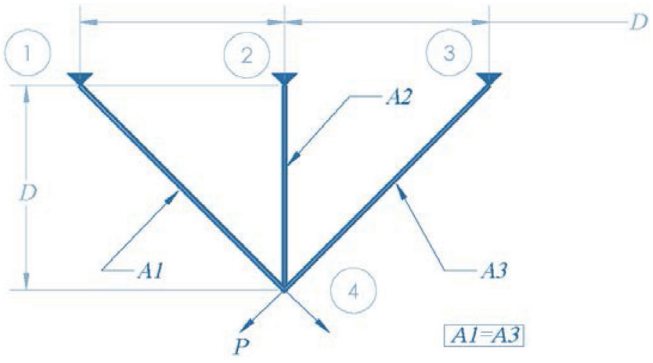


**Fig. 11.** Three-bar truss design problem.

- *Group* 8: $A_{31}$, $A_{32}$, $A_{33}$, $A_{34}$, $A_{35}$, $A_{36}$
- *Group* 9: $A_{37}$, $A_{38}$, $A_{39}$
- *Group* 10: $A_{40}$, $A_{41}$, $A_{42}$, $A_{43}$
- *Group* 11: $A_{44}$, $A_{45}$, $A_{46}$, $A_{47}$, $A_{48}$, $A_{49}$
- *Group* 12: $A_{50}$, $A_{51}$, $A_{52}$

The following list presents other parameters involved in this problem:

- $\rho = 7860.0\ kg/m^3$
- $E = 2.07e5\ MPa$
- *Stress limitation* $= 180\ MPa$
- *Maximum stress* $= 179.7652\ MPa$
- *Design variabe set are chosen from Table* 18
- $P_k = 100\ kN$, $P_y = 200\ kN$

This is a discrete problem, in which the values for the 12 parameters should be chosen from the 64 values available in Table 10. To make GOA discrete, we simply round the search agents to the nearest integer. This problem is solved using 30 search agents and 500 iterations, and the results are presented in Table 11. Inspecting the results of this table, it is evident that the GOA finds the best optimal value for this problem with the least number of function evaluations. This highlights the performance of GOA in solving real problems with more variables.

These results clearly demonstrate the merits of the GOA algorithm in solving real problems with unknown search spaces. The success of the GOA algorithm is due to several reasons. The exploration ability of GOA is high in the initial steps of optimisation, which is due to the large repulsion rate between grasshoppers. This assists GOA to explore the search space broadly and discover its promising regions. Then exploitation is high in the last steps of optimisation, which is due to the larger attraction forces between the grasshoppers. This behaviour causes local search and

**Table 8**
Comparison results of the three-bar truss design problem.

| Algorithm | Optimal values for variables | | Optimal weight | Max. Eval. |
|---|---|---|---|---|
| | $x_1$ | $x_2$ | | |
| GOA | 0.788897555578973 | 0.407619570115153 | 263.895881496069 | 13,000 |
| ALO [57] | 0.788662816000317 | 0.408283133832901 | 263.8958434 | 14,000 |
| DEDS [58] | 0.78867513 | 0.40824828 | 263.8958434 | 15,000 |
| PSO-DE [59] | 0.7886751 | 0.4082482 | 263.8958433 | 17,600 |
| MBA [56] | 0.7885650 | 0.4085597 | 263.8958522 | 20,000 |
| Ray and Sain [60] | 0.795 | 0.395 | 264.3 | N/A |
| Tsa [61] | 0.788 | 0.408 | 263.68 | N/A |
| CS [55] | 0.78867 | 0.40902 | 263.9716 | 15,000 |

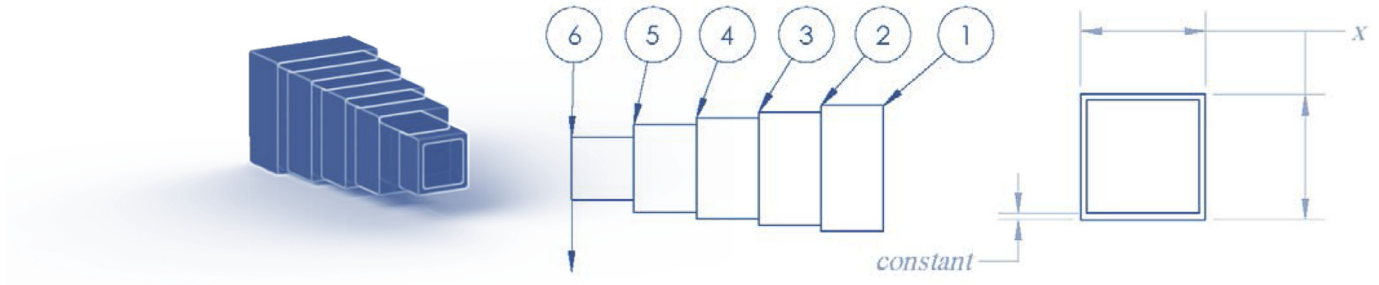


**Fig. 12.** Cantilever beam design problem.

**Table 9**
Comparison results for cantilever design problem.

| Algorithm | Optimal values for variables | | | | | Optimal | Max. |
|---|---|---|---|---|---|---|---|
| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | weight | Eval. |
| GOA | 6.011674 | 5.31297 | 4.48307 | 3.50279 | 2.16333 | 1.33996 | 13,000 |
| ALO [57] | 6.01812 | 5.31142 | 4.48836 | 3.49751 | 2.158329 | 1.33995 | 14,000 |
| MMA [62] | 6.0100 | 5.3000 | 4.4900 | 3.4900 | 2.1500 | 1.3400 | N/A |
| GCA_I [62] | 6.0100 | 5.3000 | 4.4900 | 3.4900 | 2.1500 | 1.3400 | N/A |
| GCA_II [62] | 6.0100 | 5.3000 | 4.4900 | 3.4900 | 2.1500 | 1.3400 | N/A |
| CS [55] | 6.0089 | 5.3049 | 4.5023 | 3.5077 | 2.1504 | 1.33999 | 2500 |
| SOS [63] | 6.01878 | 5.30344 | 4.49587 | 3.49896 | 2.15564 | 1.33996 | 15,000 |

improving the accuracy of the solution(s) obtained in the exploration phase. The algorithm smoothly balances exploration and exploitation, initially emphasising local optima avoidance and then convergence. This behaviour is due to the proposal of the adaptive comfort zone coefficient. The gradual decrementing of this component brings the grasshopper closer to the target proportional to the number of iterations. Finally, the proposed target chasing mechanism requires GOA to save the best solution obtained so far as the target and drive the grasshoppers towards it with the hope of improving its accuracy or finding a better one in the search space.

Considering the simulations, results, discussion, and analyses of this paper, we believe that GOA is able to solve many optimisation problems effectively. GOA considers a given optimisation problem as a black box, so it does not need gradient information of the search space. Therefore, this algorithm can be applied to any optimisation problem in different fields subject to proper problem formulation.

## 5. Conclusion

This work proposed an optimisation algorithm called the Grasshopper Optimisation Algorithm. The proposed algorithm mathematically modelled and mimicked the swarming behaviour of grasshoppers in nature for solving optimisation problems. A mathematical model was proposed to simulate repulsion and attraction forces between the grasshoppers. Repulsion forces allow grasshoppers to explore the search space, whereas attraction forces encouraged them to exploit promising regions. To balance between exploration and exploitation, GOA was equipped with a coefficient that adaptively decreases the comfort zone of the grasshoppers. Finally, the best solution obtained so far by the swarm was considered as a target to be chased and improved by the grasshoppers.

In order to benchmark the performance of the proposed algorithm, a series of tests was conducted. Firstly, a set of 2D test functions was solved by the GOA to observe its performance qualitatively. This experiment and relevant discussions support the following conclusions:

- Grasshoppers effectively discover the promising regions of a given search space.

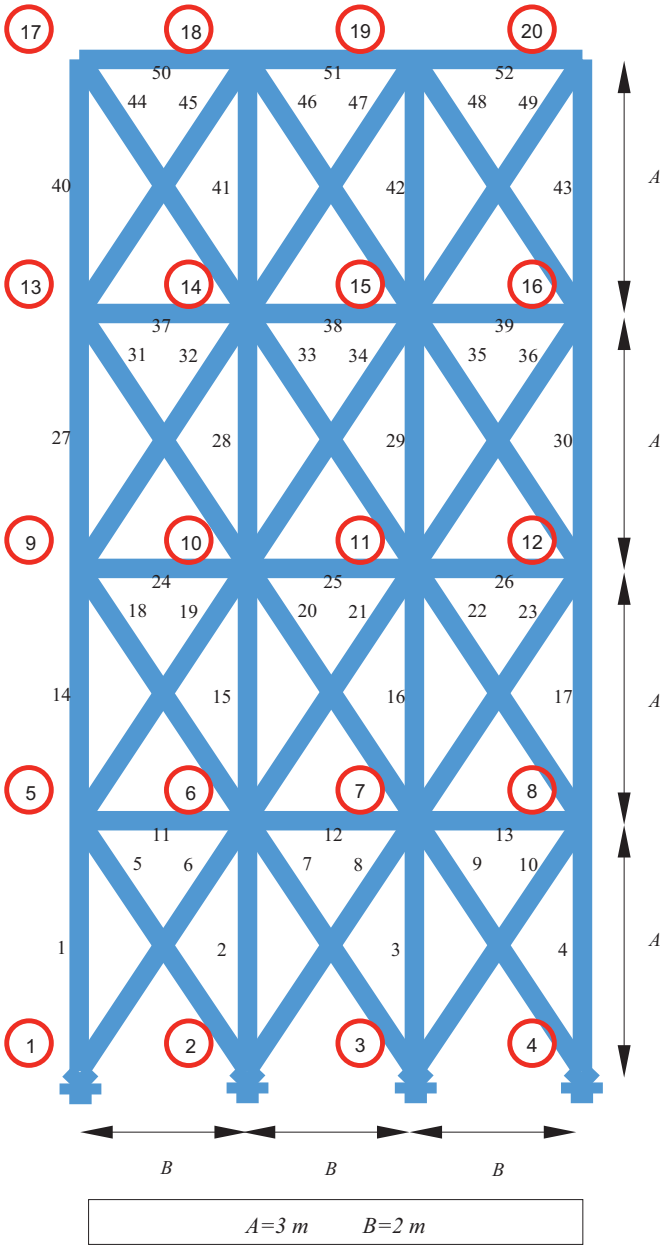


**Fig. 13.** Structure of a 52-bar truss.

- Grasshoppers face abrupt, large-scale changes in the initial steps of optimisation, which assist them to search globally.
- Grasshoppers tend to move locally in the final steps of optimisation, which allows them to exploit the search space.

**Table 10**
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 | 10,322.56 |
| 23 | 2.62 | 1690.319 | 55 | 16.9 | 10,903.2 |
| 24 | 2.63 | 1696.771 | 56 | 18.8 | 12,129.01 |
| 25 | 2.88 | 1858.061 | 57 | 19.9 | 12,838.68 |
| 26 | 2.93 | 1890.319 | 58 | 22 | 14,193.52 |
| 27 | 3.09 | 1993.544 | 59 | 22.9 | 14,774.16 |
| 28 | 3.13 | 2019.351 | 60 | 24.5 | 15,806.42 |
| 29 | 3.38 | 2180.641 | 61 | 26.5 | 17,096.74 |
| 30 | 3.47 | 2238.705 | 62 | 28 | 18,064.48 |
| 31 | 3.55 | 2290.318 | 63 | 30 | 19,354.8 |
| 32 | 3.63 | 2341.931 | 64 | 33.5 | 21,612.86 |

- The varying comfort zone coefficient requires grasshoppers to gradually balance exploration and exploitation, which helps GOA not to become trapped in local optima and find an accurate approximation of the global optimum.
- The GOA algorithm enhances the average fitness of grasshoppers, which shows that this algorithm is able to effectively improve the initial random population of grasshoppers.
- The fitness of target is improved over the course of iterations, which shows that the approximation of the global optimum becomes more accurate proportional to the number of iterations.

After the first experiment, four sets of challenging test functions were employed. The test functions were unimodal, multimodal, composite, and CEC2005. The GOA algorithm managed to outperform several algorithms in the literature. The findings and discussions of the second experiment support the following conclusions:

- Exploitation of the GOA is satisfactory on problems involving unimodal test functions.
- Exploration of the GOA is intrinsically high for multi-modal test functions.
- GOA properly balances exploration and exploitation when solving challenging problems involving composite test functions.
- GOA has the potential to significantly outperform several current algorithms when solving a range of current or new optimisation problems.

The last experiment was performed on three real problems in the field of structural design. All the problems were successfully solved, which demonstrates the practical merits of the proposed algorithm. From the results, findings, and discussions of the real applications, the following conclusions can be drawn:

- GOA is able to improve the initial random population for a real problem.
- The target is improved over the course of iterations, so the approximation of the global optimum become more accurate proportional to the number of iterations.
- GOA is able to solve real problems with unknown search spaces.

GOA is only able to solve single-objective problems with contentious variables. For future work, binary and multi-objective versions of this algorithm may be developed to solve discrete and multi-objective problems. The comfort zone parameter is an important coefficient in GOA, so it is worth investigating the impacts of different comfort zone functions on the performance of the algorithm. Solving optimisation problems in different fields could also be a valuable contribution. Tuning the main controlling parameters of GOA may also be beneficial.

## Appendix

Tables 12–15

**Table 11**
Comparison of GOA optimisation results with literature for the 52-bar truss design problem.

| Variables ($mm^2$) | PSO [64] | PSOPC [64] | HPSO [64] | DHPSACO [65] | MBA [66] | SOS [63] | GOA |
|---|---|---|---|---|---|---|---|
| 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 |
| No. of analyses | 150,000 | 150,000 | 5300 | 11,100 | 5450 | 2350 | 2300 |

**Table 12**
Unimodal benchmark functions.

| Function | Dim | Range | $f_{min}$ |
|---|---|---|---|
| $f_1(x) = \sum_{i=1}^{n} x_i^2$ | 30 | [−100,100] | 0 |
| $f_2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$ | 30 | [−10,10] | 0 |
| $f_3(x) = \sum_{i=1}^{i} (\sum_{j-1}^{i} x_j)^2$ | 30 | [−100,100] | 0 |
| $f_4(x) = \max_i \{\lvert x_i \rvert,\ 1 \le i \le n\}$ | 30 | [−100,100] | 0 |
| $f_5(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$ | 30 | [−30,30] | 0 |
| $f_6(x) = \sum_{i=1}^{n} ([x_i + 0.5])^2$ | 30 | [−100,100] | 0 |
| $f_7(x) = \sum_{i=1}^{n} i x_i^4 + random[0, 1)$ | 30 | [−1.28,1.28] | 0 |

**Table 13**
Multimodal benchmark functions.

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

**Table 14**
Composite benchmark functions.

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

**Table 15**
CEC2005 test functions.

| Function | Dim | Range | $f_{min}$ |
|---|---|---|---|
| F1_CEC2005: Shifted Sphere Function | 30 | [−100,100] | −450 |
| F2_CEC2005: Shifted Schwefel's Problem | 30 | [−100,100] | −450 |
| F3_CEC2005:Shifted Rotated High Conditioned Elliptic Function | 30 | [−100,100] | −450 |
| F4_CEC2005: Shifted Schwefel's Problem with Noise in Fitness | 30 | [−100,100] | −450 |
| F5_CEC2005: Schwefel's Problem with Global Optimum on Bounds | 30 | [−100,100] | −310 |
| F6_CEC2005: Shifted Rosenbrock's Function | 30 | [−100,100] | 390 |
| F7_CEC2005: Shifted Rotated Griewank's Function without Bounds | 30 | [−600, 600] | −180 |
| F8_CEC2005: Shifted Rotated Ackley's Function with Global Optimum on Bounds | 30 | [−32,32] | −140 |
| F9_CEC2005: Shifted Rastrigin's Function | 30 | [−5,5] | −330 |
| F10_CEC2005: Shifted Rotated Rastrigin's Function | 30 | [−5,5] | −330 |
| F11_CEC2005: Shifted Rotated Weierstrass Function | 30 | [−0.5,0.5] | 90 |
| F12_CEC2005: Schwefel's Problem | 30 | [−100,100] | −460 |
| F13_CEC2005: Expanded Extended Griewank's plus Rosenbrock's Function (F8F2) | 30 | [−3,1] | −130 |
| F14_CEC2005: Expanded Rotated Extended Scaffe's F6 | 30 | [−100,100] | −300 |
| F15_CEC2005: Hybrid Composition Function 1 | 30 | [−5,5] | 120 |
| F16_CEC2005: Rotated Hybrid Composition Function 1 | 30 | [−5,5] | 120 |
| F17_CEC2005: Rotated Hybrid Composition Function 1 with Noise in Fitness | 30 | [−5,5] | 120 |
| F18_CEC2005: Rotated Hybrid Composition Function 2 | 30 | [−5,5] | 10 |
| F19_CEC2005: Rotated Hybrid Composition Function 2 with a Narrow Basin for the Global Optimum | 30 | [−5,5] | 10 |
| F20_CEC2005: Rotated Hybrid Composition Function 2 with the Global Optimum on the Bounds | 30 | [−5,5] | 10 |
| F21_CEC2005: Rotated Hybrid Composition Function 3 | 30 | [−5,5] | 360 |
| F22_CEC2005: Rotated Hybrid Composition Function 3 with High Condition Number Matrix | 30 | [−5,5] | 360 |
| F23_CEC2005: Non-Continuous Rotated Hybrid Composition Function 3 | 30 | [−5,5] | 360 |
| F24_CEC2005: Rotated Hybrid Composition Function 4 | 30 | [−5,5] | 260 |
| F25_CEC2005: Rotated Hybrid Composition Function 4 without Bounds | 30 | [−5,5] | 260 |

## References


[1] Coello Coello CA. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Comput Meth Appl Mech Eng 2002;191:1245–87.
[2] Marler RT, Arora JS. Survey of multi-objective optimization methods for engineering. Struct Multidiscipl Optim 2004;26:369–95.
[3] Mirjalili S, Mirjalili SM, Lewis A. Grey wolf optimizer. Adv Eng Softw 2014;69:46–61.
[4] Spall JC. Introduction to stochastic search and optimization: estimation, simulation, and control, vol. 65. John Wiley & Sons; 2005.
[5] Dasgupta D, Michalewicz Z. Evolutionary algorithms in engineering applications. Springer; 1997.
[6] Yang X-S. Nature-inspired metaheuristic algorithms. Luniver press; 2010.
[7] Mirjalili S, Lewis A. S-shaped versus V-shaped transfer functions for binary particle swarm optimization. Swarm Evol Comput 2013;9:1–14.
[8] Davis L. Bit-climbing, representational bias, and test suite design. ICGA 1991:18–23.
[9] Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by simmulated annealing. Science 1983;220:671–80.
[10] L.J. Fogel, A.J. Owens, and M.J. Walsh, "Artificial intelligence through simulated evolution," 1966.
[11] Glover F. Tabu search-part I. ORSA J Comput 1989;1:190–206.
[12] H.R. Lourenço, O.C. Martin, and T. Stutzle, "Iterated local search," arXiv preprint math/0102188, 2001.
[13] Holland JH. Genetic algorithms. Sci Am 1992;267:66–72.
[14] Eberhart RC, Kennedy J. A new optimizer using particle swarm theory. In: Proceedings of the sixth international symposium on micro machine and human science; 1995. p. 39–43.
[15] Colorni A, Dorigo M, Maniezzo V. Distributed optimization by ant colonies. In: Proceedings of the first European conference on artificial life; 1991. p. 134–42.
[16] Storn R, Price K. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 1997;11:341–59.
[17] Eiben AE, Schippers C. On evolutionary exploration and exploitation. Fundamenta Informaticae 1998;35:35–50.
[18] Kaveh A, Farhoudi N. A new optimization method: dolphin echolocation. Adv Eng Softw 2013;59:53–70.
[19] Kaveh A, Farhoudi N. Dolphin monitoring for enhancing metaheuristic algorithms: layout optimization of braced frames. Comput Struct 2016;165:1–9.
[20] Yang X-S. Firefly algorithm, stochastic test functions and design optimisation. Int J Bio Inspired Comput 2010;2:78–84.
[21] Yang X-S. Firefly algorithm, Levy flights and global optimization. In: Research and development in intelligent systems XXVI. Springer; 2010. p. 209–18.
[22] Yang X-S. A new metaheuristic bat-inspired algorithm. In: Nature inspired cooperative strategies for optimization (NICSO 2010). Springer; 2010. p. 65–74.
[23] Yang X-S, Deb S. Cuckoo search via Lévy flights. In: Nature & biologically inspired computing, 2009. NaBIC 2009. World congress on; 2009. p. 210–14.
[24] Yang X-S, Deb S. Engineering optimisation by cuckoo search. Int J Math Model Numer Optim 2010;1:330–43.
[25] Cuevas E, Echavarría A, Ramírez-Ortegón MA. An optimization algorithm inspired by the States of Matter that improves the balance between exploration and exploitation. Appl Intel 2014;40:256–72.
[26] Cuevas E, Echavarría A, Zaldívar D, Pérez-Cisneros M. A novel evolutionary algorithm inspired by the states of matter for template matching. Expert Syst Appl 2013;40:6359–73.
[27] Yang X-S. Flower pollination algorithm for global optimization. In: Unconventional computation and natural computation. Springer; 2012. p. 240–9.
[28] Wolpert DH, Macready WG. No free lunch theorems for optimization. Evol Comput IEEE Trans 1997;1:67–82.
[29] Chen S. Locust Swarms-A new multi-optima search technique. In: Evolutionary Computation, 2009. CEC'09. IEEE Congress on; 2009. p. 1745–52.
[30] Chen S. An analysis of locust swarms on large scale global optimization problems. In: Artificial Life: borrowing from biology. Springer; 2009. p. 211–20.
[31] Chen S, Vargas YN. Improving the performance of particle swarms through dimension reductions—A case study with locust swarms. In: Evolutionary computation (CEC), 2010 IEEE congress on; 2010. p. 1–8.
[32] Lewis A. LoCost: a spatial social network algorithm for multi-objective optimisation. In: Evolutionary computation, 2009. CEC'09. IEEE congress on; 2009. p. 2866–70.
[33] Cuevas E, Cortés MAD, Navarro DAO. Optimization based on the behavior of locust swarms. In: Advances of evolutionary computation: methods and operators. Springer; 2016. p. 101–20.
[34] BoussaïD I, Lepagnot J, Siarry P. A survey on optimization metaheuristics. Inf Sci 2013;237:82–117.
[35] Gogna A, Tayal A. Metaheuristics: review and application. J Exp Theor Artif Intel 2013;25:503–26.
[36] Zhou A, Qu B-Y, Li H, Zhao S-Z, Suganthan PN, Zhang Q. Multiobjective evolutionary algorithms: A survey of the state of the art. Swarm Evol Comput 2011;1:32–49.
[37] Simpson SJ, McCaffery A, HAeGELE BF. A behavioural analysis of phase change in the desert locust. Biol Rev 1999;74:461–80.
[38] Rogers SM, Matheson T, Despland E, Dodgson T, Burrows M, Simpson SJ. Mechanosensory-induced behavioural gregarization in the desert locust Schistocerca gregaria. J Exp Biol 2003;206:3991–4002.
[39] Topaz CM, Bernoff AJ, Logan S, Toolson W. A model for rolling swarms of locusts. Eur Phys J Special Top 2008;157:93–109.
[40] Yao X, Liu Y, Lin G. Evolutionary programming made faster. In: Evolutionary computation, IEEE transactions on, 3; 1999. p. 82–102.
[41] Digalakis J, Margaritis K. On benchmarking functions for genetic algorithms. Int J Comput Math 2001;77:481–506.
[42] M. Molga and C. Smutnicki, "Test functions for optimization needs," *Test functions for optimization needs*, 2005.
[43] X.-S. Yang, "Test problems in optimization," *arXiv preprint arXiv:1008.0549*, 2010.
[44] Rashedi E, Nezamabadi-Pour H, Saryazdi S. GSA: a gravitational search algorithm. Inf Sci 2009;179:2232–48.
[45] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.-P. Chen, A. Auger, et al., "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization," *KanGAL report*, vol. 2005005, p. 2005, 2005.
[46] Kaveh A. Colliding Bodies Optimization. In: Advances in metaheuristic algorithms for optimal design of structures. Springer; 2014. p. 195–232.
[47] Kaveh A, Mahdavi V. Colliding bodies optimization: a novel meta-heuristic method. Comput Struct 2014;139:18–27.
[48] Kaveh A, Bakhshpoori T. A new metaheuristic for continuous structural

optimization: water evaporation optimization. Struct Multidiscipl Optim 2016:1–21.
[49] Kaveh A, Bakhshpoori T, Afshari E. An efficient hybrid particle swarm and swallow swarm optimization algorithm. Comput Struct 2014;143:40–59.
[50] Kaveh A, Khayatazad M. A new meta-heuristic method: ray optimization. Comput Struct 2012;112:283–94.
[51] Kaveh A, Khayatazad M. Ray optimization for size and shape optimization of truss structures. Comput Struct 2013;117:82–94.
[52] Kaveh A, Nasrollahi A. A new hybrid meta-heuristic for structural design: ranked particles optimization. Struct Eng Mech 2014;52:405–26.
[53] Kaveh A, Talatahari S. A novel heuristic optimization method: charged system search. Acta Mech 2010;213:267–89.
[54] Kaveh A, Mirzaei B, Jafarvand A. An improved magnetic charged system search for optimization of truss structures with continuous and discrete variables. Appl Soft Comput 2015;28:400–10.
[55] Gandomi AH, Yang X-S, Alavi AH. Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Eng Comput 2013;29:17–35.
[56] Sadollah A, Bahreininejad A, Eskandar H, Hamdi M. Mine blast algorithm: A new population based algorithm for solving constrained engineering optimization problems. Appl Soft Comput 2013;13:2592–612.
[57] Mirjalili S. The ant lion optimizer. Adv Eng Softw 2015;83:80–98.
[58] Zhang M, Luo W, Wang X. Differential evolution with dynamic stochastic selection for constrained optimization. Inf Sci 2008;178:3043–74.
[59] Liu H, Cai Z, Wang Y. Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization. Appl Soft Comput 2010;10:629–40.
[60] Ray T, Saini P. Engineering design optimization using a swarm with an intelligent information sharing among individuals. Eng Optim 2001;33:735–48.
[61] Tsai J-F. Global optimization of nonlinear fractional programming problems in engineering design. Eng Optim 2005;37:399–409.
[62] Chickermane H, Gea H. Structural optimization using a new local approximation method. Int J Numer Methods Eng 1996;39:829–46.
[63] Cheng M-Y, Prayogo D. Symbiotic organisms search: A new metaheuristic optimization algorithm. Comput Struct 2014;139:98–112.
[64] Li L, Huang Z, Liu F. A heuristic particle swarm optimization method for truss structures with discrete variables. Comput Struct 2009;87:435–43.
[65] Kaveh A, Talatahari S. A particle swarm ant colony optimization for truss structures with discrete variables. J Constr Steel Res 2009;65:1558–68.
[66] Sadollah A, Bahreininejad A, Eskandar H, Hamdi M. Mine blast algorithm for optimization of truss structures with discrete variables. Comput Struct 2012;102:49–63.