# Modified Grasshopper Optimization Algorithm: Addressing Challenges in Constrained Optimization Problems

NAJWAN OSMAN-ALI, JUNITA MOHAMAD-SALEH, WAN-AMIR-FUAD-WAJDI OTHMAN, HAIDI IBRAHIM
School of Electrical and Electronic Engineering,
Universiti Sains Malaysia,
Nibong Tebal 14300, Penang,
MALAYSIA

*Abstract:* - The Grasshopper Optimization Algorithm (GOA), which is based on the grasshopper swarming characteristics, has been demonstrated to effectively solve global constrained optimization problems. However, the original GOA faces challenges, such as an imbalance between exploration and exploitation owing to its linear convergence parameter, slow convergence rate, and tendency to become trapped in local optima. To address these issues, this study proposes a Modified Grasshopper Optimization Algorithm (MGOA). The MGOA incorporates two key modifications: enhanced social interaction and a revised convergence parameter designed to balance exploitation and exploration. The algorithm was evaluated using the CEC2021 benchmark functions, and it outperformed the original GOA in most cases. Statistical analysis using the Wilcoxon signed-rank and Friedman ranking tests further confirmed the significant improvements achieved by the MGOA. These results demonstrate the potential of the MGOA as a more effective optimization approach than the original GOA.

*Key-Words:* - Grasshopper optimization algorithm, GOA, meta-heuristics, optimization, swarm intelligence, constrained optimization, CEC2021 benchmark functions, exploration and exploitation, Lévy flight.

Received: September 15, 2025. Revised: April 18, 2026. Accepted: May 11, 2026. Published: July 20, 2026.

## 1 Introduction

Optimization is the cornerstone of numerous scientific and industrial fields, offering swift and accurate resolutions to a diverse array of problems. However, this is a notoriously complex endeavor because of the nonlinear and multimodal objective landscapes and nonlinear constraints that characterize most of the optimization problems, [1]. Current methodologies for addressing these challenges fall under two paradigms [2]: conventional and nature-inspired.

Despite remarkable advances in computational power, traditional algorithms for solving optimization problems, such as trust-region, interior-point, and gradient-based methods, often require significant computational resources when calculating derivatives, particularly in the context of problems with discontinuous objective functions or those without derivatives in certain regions, [3]. This inherent limitation has inspired a shift towards nature-inspired methodologies. These innovative algorithms serve as global optimizers, utilizing interconnected agents to generate search movements within the solution space, thus offering an alternative and often more efficient path to problem-solving.

In this dynamic research environment, a profusion of nature-inspired algorithms, each with unique characteristics, has been developed, which can be categorized as either population-based or single-agent algorithms, [4]. Single-agent algorithms produce one solution per iteration, such as Iterated Local Search (ILS) [5], Guided Local Search (GLS) [6], and Variable Neighborhood Search (VNS) [7]. In contrast, population-based algorithms emulate the complex behaviors observed in nature, including swarming, human behavior, physics-inspired phenomena, and evolutionary processes, and typically generate multiple solutions per iteration, [8]. Swarm-based algorithms replicate collective behaviors, such as the movement of insects foraging for food, with notable examples being bees, ants, and grasshoppers. Algorithms in this category include the Artificial Bee Colony (ABC) [9], Particle Swarm Optimization (PSO) [10], Ant Colony Optimization (ACO) [11], and Bat Algorithm (BA), [12]. Human behavior serves as the foundation for algorithms such as the Imperialist

Competitive Algorithm (ICA) [13], Teaching-Learning-Based Algorithm (TLBA) [14], and Harmony Search (HS) [15]. Physics-based algorithms are inspired by natural phenomena, leading to methods such as Thermal Exchange Optimization (TEO) [16], Water Evaporation Optimization (WEO) [17], and Gravitational Search Algorithm (GSA) [18]. Finally, evolutionary algorithms draw on mechanisms such as selection, recombination, and mutation, with prominent examples including Differential Evolution (DE) [19], [20], Evolution Strategy (ES) [21], and Genetic Algorithm (GA), [22], [23].

Irrespective of their diverse methodologies, all nature-inspired algorithms share two fundamental phases: exploration and exploitation phases. The exploration phase facilitates a broad search across the solution space, whereas the exploitation phase focuses on refining promising regions to identify optimal solutions. Striking a balance between these two phases is crucial for the success of nature-inspired algorithms.

Despite the extensive range of optimization algorithms, no universal solution exists. This assertion is underlined by the "No Free Lunch (NFL) theorem" [24], which states that no algorithm can be expected to perform better than any other across all classes of optimization problems. Therefore, the development of a nature-inspired algorithm should consider a specific application rather than pursuing a universally applicable algorithm.

Within the landscape of nature-inspired algorithms, the GOA has demonstrated notable success in various applications, including determining the optimal location and size of battery energy storage systems to minimize cost while enhancing energy efficiency [25], optimizing prediction and control horizons in Model Predictive Control (MPC) for motion simulators to achieve faster processing times and improved simulation efficiency [26], and planning global paths for mobile robots in a known static environment [27].

Despite its success, the GOA is not without its limitations, [28]. One of its primary challenges is its linear convergence parameter, which disrupts the balance between the exploration and exploitation phases, [28]. This imbalance often leads to slow convergence and an increased likelihood of being trapped in local optima. To address these shortcomings, researchers have developed various improved versions of GOA, including variants and hybrids. Variants have incorporated techniques such as Levy Flight, [29], chaotic maps [30], and natural selection strategies with dynamic feedback mechanisms [31]. The combination of the original GOA and other nature-inspired algorithms is referred to as a hybrid. Examples of such hybrids include the Grey Wolf Optimizer (GWO) [32], ABC [33], and GA [34]. These hybrid methods generally improve the algorithm's capacity to escape local optima and resolve movement challenges, but often result in increased complexity and computational demands.

This study aims to overcome the limitations of the GOA by introducing two significant modifications, collectively referred to as the modified GOA (MGOA). The first modification enhances the social interaction mechanism by refining the manner in which grasshoppers interact during the optimization process. The second modification adjusts the convergence parameter to achieve a more effective balance between exploration and exploitation by leveraging the proposed social interaction strategy. The MGOA was evaluated using the CEC2021 benchmark functions for real-parameter optimization [35], and its performance was benchmarked against the original GOA.

The remainder of this paper is structured as follows: Section 2 provides an overview of the original GOA, and Section 3 elaborates on the details of the proposed MGOA. The experimental results using the CEC2021 benchmark functions to assess the MGOA performance are discussed in Section 4. Finally, the conclusions drawn from the simulation results are presented in Section 5

## 2 Grasshopper Optimization `Algorithm (GOA)

Grasshoppers exhibit varying behaviors that are contingent on their environmental context. Grasshoppers are solitary by nature and typically maneuver independently, striving to avoid conspecifics. However, under specific stimuli, such as tactile contact, its demeanor shifts towards aggression, triggering swarm formation. The swarm then migrates collectively to search for nourishment, [36]. This fascinating display of swarm intelligence serves as the foundational inspiration for the computational development of the Grasshopper Optimization Algorithm (GOA).

The original GOA capitalized on these behavioral aspects of grasshopper swarms, harnessing them as a novel search strategy in the vast landscape of optimization problems. Central to the efficacy of the GOA in converging towards a solution is the nuanced interaction that occurs

among grasshopper agents within the swarm. The GOA models three fundamental spheres of behavior among agents: attraction, comfort, and repulsion zones, [36]. These are illustrated in Fig. 1, which depicts dynamic interactions between grasshoppers.

In this model, each grasshopper is encircled by a comfort zone, which is visualized as a spherical boundary, as illustrated in Fig. 1. When another grasshopper enters this zone, it encounters a neutral force, where the attraction and repulsion forces are balanced. Using Grasshopper A as a reference, Grasshopper B, which is positioned within the comfort zone, experiences no net attraction or repulsion owing to the neutral force. In contrast, Grasshoppers such as Grasshopper C, which come closer to A inside this zone, are subjected to a repulsive force that pushes them away. In contrast, grasshoppers, such as D, located beyond the comfort zone, are drawn towards A by an attractive force. These dynamic interactions persist within the swarm, gradually guiding it towards convergence on the optimal solution.

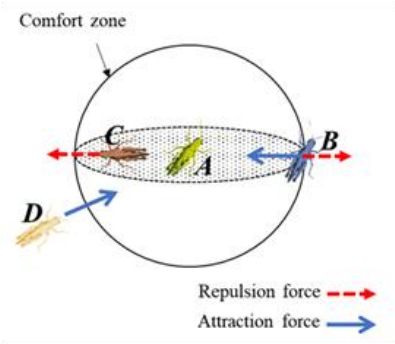


Fig. 1: Fundamental adjustment pattern in grasshopper swarm behavior
*Source: adapted from [37]*

The natural movement of grasshoppers within a swarm can be mathematically modeled. The foundational equation of the GOA, which determines the position of each grasshopper, is expressed as [37]

$$X_i = c\left(\sum_{j=1, j\neq i}^{N} c\frac{ub_d - lb_d}{2} s\left(\left|x_j^d - x_i^d\right|\right)\frac{x_j^d - x_i^d}{d_{ij}}\right) + T \quad (1)$$

In this context, X_i denotes the position of the *i*-th grasshopper across all dimensions, with *D* representing the total number of dimensions in the search space, where *d=1,2,⋯, D*. The positions of the *i*-th and *j*-th grasshoppers in the *d*-th dimension are represented by $x_i^d$ and $x_j^d$, respectively, [37]. *T* refers to the current best-known solution, [37]. The equation comprises two primary components: the first term, which involves the summation within brackets multiplied by *c*, and the second term, *T* ,[37]. The first term models the interactions between grasshoppers, confined within a space defined by $ub_d$ and $lb_d$, which represent the upper and lower bounds in the *d*-th dimension of the search space, respectively. The distance between the *i*-th and *j*-th grasshoppers is given by $\left|x_j^d - x_i^d\right|$. Parameter *s*, quantifying the strength of social interactions among grasshoppers, is defined as [37]

$$s(r) = fe^{-r/l} - e^{-r} \quad (2)$$

In this context, *f* and *l* represent the attraction intensity and length scale, respectively, with their optimal values in the original GOA defined as *f*=0.5 and *l*=1.5, [37]. The variable *r* scales the distance $\left|x_j^d - x_i^d\right|$ to fall within a range of 1 to 4, [37]. The coefficient *c*, appearing in the first term, is a parameter that decreases monotonically and linearly with each iteration. This parameter plays a pivotal role in managing the balance between exploration and exploitation by controlling the expansion and contraction of the grasshoppers' comfort, repulsion, and attraction zones. In the original GOA, this balance is achieved by linearly reducing *c* over time, following the equation, [37]:

$$c = c_{\max} - iter\frac{c_{\max} - c_{\min}}{L} \quad (3)$$

where $c_{max}$ and $c_{min}$ are the maximum and minimum values of *c*, respectively; *iter* is the current iteration; and *L* is the maximum number of iterations, [37]. The second term in (1) reflects the tendency of grasshoppers to move towards the food source, [37]. By iterating this process for all grasshoppers, the swarm eventually converges to an optimal solution.

## 3 Modified Grasshopper Optimization Algorithm (MGOA)

The concept behind this modification stems from research on swarming locusts. Instead of using the social interaction from the GOA in Fig. 1. The MGOA employs a more sophisticated social interaction model adapted from [38], which seems more promising for understanding the collective motion in grasshoppers.

Similar to the GOA, the Euclidean distance between grasshoppers is a critical element in the MGOA. The distance value determines the current action that the grasshopper should perform while maintaining its movement within the swarm. Each grasshopper checks the surrounding area for grasshoppers that are close to or too close to it. There are two possibilities for grasshoppers: a

neighbor exists, or the grasshopper has no neighbor in its proximity.

Depending on the position of its neighbors, it decides the specific steps to be taken. If the neighbor is the nearest neighbor, the grasshopper separates itself and moves a certain distance. Conversely, the grasshopper coheres and aligns when moving with its close-by neighbors. These steps are executed while moving forward to find the optimum food source. Throughout the steps, the grasshopper must be aware of its current position and neighbor, as well as its velocity and the velocity of its neighbors, particularly for cohesion and alignment.

Based on the proposed grasshopper social interactions, three main actions must be mathematically presented if a neighbor exists: avoidance, cohesion, and alignment. However, only one action is required for the grasshopper if no neighbors are nearby, which is random jumping.

A notable distinction between the MGOA and GOA is their use of grasshopper data. Whereas the GOA relies solely on the grasshopper position values in its equations, the MGOA incorporates both position and velocity values to represent stages in the proposed social interaction. In accordance with this interaction, the categorization of neighbors and non-neighbors for the i-th grasshopper is determined by the distance between the i-th grasshopper and other grasshoppers in the same population. This distance can be computed as:

$$d_{i,j} = \left\| X_j - X_i \right\| \tag{4}$$

where $d_{i,j}$ is the distance between the *i*-th and *j*-th grasshoppers. All distance values between the *i*-th grasshopper and other grasshoppers within the same population can be represented as:

$$D_i = \left( d_{i,j} \right)_{1 \le j \le N,\, j \ne i} \tag{5}$$

Once the distances are calculated, the neighbors and non-neighbors of the *i*-th grasshopper are identified as follows: The minimum distance, *MinDS*, in conjunction with a non-neighbor distance, *NND*, is used as a reference to identify the type of neighbor, where the relationship between *NND* and *MinDS* is as follows:

$$NND = \alpha MinDS \tag{6}$$

In (6), the multiplier $\alpha$ is employed to ensure a consistent separation between *the MinDS* and *NND*. Using a trial-and-error approach, $\alpha$=200 was identified as the optimal value for achieving the best results.

Fig. 2 displays the zones that separate neighbors from non-neighbors and differentiate between neighbor types. Using grasshopper *A* as a reference, grasshopper *B* is considered to be the nearest neighbor because the distance to *A* is lower than *MinDS,* which can be expressed as

$$0 < d_{i,j} \le MinDS \tag{7}$$

whereas grasshopper *C* is considered a close-by neighbor because of its distance from grasshopper *A*, which is larger than *MinDS* but smaller than *NND,* represented as

$$MinDS < d_{i,j} \le NND \tag{8}$$

Grasshopper *D* in Fig. 2 has a distance from *A* that is larger than *the NND* and is considered a non-neighbor.

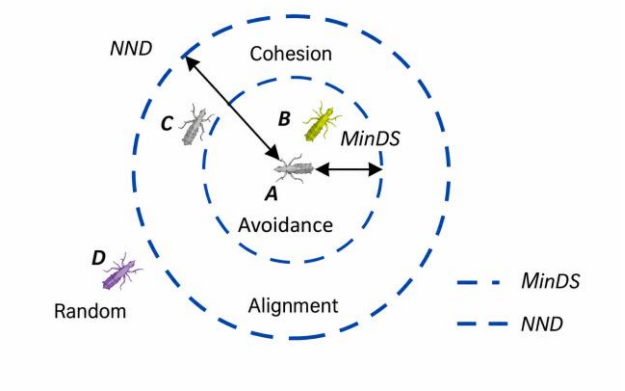


Fig. 2**:** MGOA grasshopper zones
*Source: created by the authors.*

Once the type of neighbor has been identified, the next step is to associate the actions with the *i*-th grasshopper based on the type of neighbor.

### 3.1 Action in which Neighbors Exist

Three main actions govern the movement of the *i*-th grasshopper with its neighbor. The first action is the avoidance stage, which occurs when the nearest grasshopper neighbor exists. The primary purpose of this stage was to simulate grasshopper behavior, which moves away each time it comes into close contact with other grasshoppers. Under avoidance, two cases must be considered: (i) the current grasshopper's fitness is better than that of its neighbors and (ii) the current grasshopper's fitness is inferior to that of one or more of its neighbors.

The updated position of the *i*-th grasshopper, $X_i^{new}$, can be obtained for the first case by adding the distance to the current position, using the following equation:

$$X_i^{new} = X_i + DTA \tag{9}$$

where *DTA* is *MinDS*, with unit distance *T* between the grasshopper and the position of the current best

solution found so far. The *DTA* value was calculated as follows:

$$DTA = \begin{cases} \dfrac{T - X_i}{\|T - X_i\|} \cdot MinDS, & \|T - X_i\| > 0 \\ R, \text{otherwise}, R \in \mathbb{R}, R \in [-1,1] \end{cases} \quad (10)$$

The use of *MinDS* multiplied by a unit vector is applicable only if the distance magnitude, $\|T - X_i\|$, is greater than zero. A random number between -1 and 1 was generated and assigned to the *DTA* if the condition was not fulfilled. Using the same *DTA* value, the updated velocity of the i-th grasshopper, $V_i^{new}$, is calculated as:

$$V_i^{new} = DTA \quad (11)$$

In the second case, under avoidance, a method similar to the first was used, adding a particular unit vector value to the current position. The difference from the second case is the addition of *DT*, which is the distance between the grasshopper and its neighbor with the lowest fitness value among the grasshopper neighbors. The updated position of the *i*-th grasshopper, $X_i^{new}$, is calculated as:

$$X_i^{new} = X_i + DT + DTA \quad (12)$$

where

$$DT = \begin{cases} \dfrac{X_{n,best} - X_i}{\|X_{n,best} - X_i\|} \cdot MinDS, & \|X_{n,best} - X_i\| > 0 \\ R, \text{otherwise}, R \in \mathbb{R}, R \in [-1,1] \end{cases} \quad (13)$$

$X_{n,best}$ in (13) is the neighbor of $X_i$ with the lowest fitness values. The use of *MinDS* with a unit vector is applicable only if the distance magnitude, $\|X_{n,best} - X_i\|$, is greater than zero. A random number between -1 and 1 was generated and assigned to the *DT* if the condition was not fulfilled. Using the same *DT* and *DTA*, the updated velocity of the *i*-th grasshopper, $V_i^{new}$ can be calculated as:

$$V_i^{new} = DT + DTA \quad (14)$$

The second and third actions, alignment and cohesion, respectively, can be calculated using a single equation as follows: Both actions resemble the swarming movement, where the grasshoppers align themselves with their neighbors' movements at similar velocities.

The alignment, *DA*, is *MinDS* with a unit velocity difference between the grasshopper velocity and the average velocity of its neighbors, $\bar{V}_{n,i}$, which can be calculated using the following equation:

$$DA = \begin{cases} \dfrac{\bar{V}_{n,i} - V_i}{\|\bar{V}_{n,i} - V_i\|} \cdot MinDS, & \|\bar{V}_{n,i} - V_i\| > 0 \\ R, \text{otherwise}, R \in \mathbb{R}, R \in [-1,1] \end{cases} \quad (15)$$

The purpose of using the *DA* is to adjust the grasshopper heading according to the average headings of its neighbors. Compared with *DA*, cohesion, *DC*, is used to adjust the grasshopper's position such that it remains close to its neighbors, $X_{n,i}$. The *DC* can be calculated by first calculating the mean distance between the grasshopper and its close-by neighbors, $\bar{D}_{close}$, as follows:

$$\bar{D}_{close,i} = \frac{1}{k}\left(\sum_{j=1}^{k} \|X_{close,j} - X_i\|\right) \quad (16)$$

where $k$ is the total number of close-by neighbors for the *i*-th grasshopper. $X_{close}$ is the position of the *i*-th grasshopper's close-by neighbors. The *DC* can then be calculated using the following equation:

$$DC = \begin{cases} \dfrac{\bar{D}_{close,i} - X_i}{\|\bar{D}_{close,i} - X_i\|} \cdot MinDS, & \|\bar{D}_{close,i} - X_i\| > 0 \\ R, \text{otherwise}, R \in \mathbb{R}, R \in [-1,1] \end{cases} \quad (17)$$

Similar to the avoidance stage, two cases must be considered: (i) the current grasshopper's fitness is better than that of its neighbors and (ii) the current grasshopper's fitness is inferior to that of one or more of its neighbors. In the first case, $X_i^{new}$ can be calculated as:

$$X_i^{new} = X_i + DA + DC + DTA \quad (18)$$

where its velocity is given by

$$V_i^{new} = DA + DC + DTA \quad (19)$$

The second case has a slightly different equation from the first. The updated position value of the *i*-th grasshopper is calculated using the following equation:

$$X_i^{new} = X_i + DA + DC + DT + DTA \quad (20)$$

while the velocity of the *i*-th grasshopper is calculated as

$$V_i^{new} = DA + DC + DT + DTA \quad (21)$$

## 3.2 Condition in which Neighbors are Nonexistent

In the case where no neighbor exists within the grasshopper's proximity, only one action is required to update the *i*-th grasshopper's position. The action of the proposed social interaction involves random jumping in the search space. However, the best method for simulating the actual movement of randomly moving animals is the Lévy Flight (LF), [39]. Using *LF*, the unit vector used to update the *i*-th grasshopper position, called *JLF*, is calculated as follows:

$$JLF = rand * Levy(\beta) \oplus (X_i - T) \quad (22)$$

A random number in the interval from 0 to 1 is represented by *rand*, and the product $\oplus$ is an entry-wise multiplication. The position of the *i*-th grasshopper can then be updated by adding the *JLF* using the following equation:

$$X_i^{new} = X_i + JLF \tag{23}$$

Owing to the random jumping of the grasshopper to a new position, the velocity of the grasshopper can be associated with the unit vector JLF, which is used in (23). The velocity of the *i*-th grasshopper is calculated as:

$$V_i^{new} = JLF \tag{24}$$

There are two main requirements for implementing the LF: the direction that propels it towards the target position, which can be calculated using a uniform distribution, and the step length of this movement that follows the chosen Lévy distribution. A commonly used method that is simple and efficient for determining these features is the Mantegna algorithm, [40]. This algorithm produces a symmetric and stable Lévy distribution, [2]. The step length, *S*, according to [40], can be calculated as:

$$S = \frac{U}{|V|^{1/\beta}} \tag{25}$$

where $\beta$ is an important Lévy distribution index that adjusts the stability of the result bounded as:

$$0 < \beta \le 2 \tag{26}$$

while *U* and *V* are such that:

$$U \sim \mathrm{N}(0, \sigma_{\mathrm{u}}), \qquad V \sim \mathrm{N}\left(0, \sigma_{\mathrm{v}}^2\right) \tag{27}$$

The standard deviations $\sigma_{\mathrm{u}}$ and $\sigma_{\mathrm{v}}$ are defined by

$$\sigma_{\mathrm{u}} = \left[ \frac{\Gamma(1+\beta) \times \sin\left(\frac{\pi\beta}{2}\right)}{\Gamma\left(1+\frac{\beta}{2}\right) \times \beta \times 2^{(\beta-1)/2}} \right]^{1/\beta}, \qquad \sigma_{\mathrm{v}} = 1 \tag{28}$$

where $\Gamma(\ )$ denotes the gamma function.

### 3.3 Linear Exploration and Exploitation with Dynamic Weight (LEDW)

Linear exploration and exploitation with dynamic weight (LEDW) is an improved method for linear exploration and exploitation in GOA that utilizes the convergence parameter. The convergence parameter plays an important role in GOA. It controls how the agents search for the optimum value in the search space at the beginning, middle, and end of the iteration. This directly affects the exploration and exploitation capabilities of the algorithm.

Based on the proposed social interaction, the best value that can be used as a converging parameter value is *MinDS* because it directly affects the behavior of each grasshopper, as shown in (6) and (8). The value for *MinDS* can be calculated by utilizing a similar equation as coefficient *c* in (1) for the original GOA, where

$$MinDS = cMax - iter\left(\frac{cMax - cMin}{L}\right) \tag{29}$$

Both *cMax* and *cMin* values were used to control the size of the zones. The value of *cMin* was set to a constant value of 0.001, and the *cMax* value was calculated using the following equation:

$$cMax = \frac{|lb - ub|}{r}, 0 < r < \left\| X_{\max} - X_{\min} \right\| \tag{30}$$

Variables *lb* and *ub* are the upper and lower bounds of the search space, respectively. The upper and lower bound differences are divided by the range division value for the target position in the search space *r*. Fig. 3 shows how *cMax* is selected for a single target position in a two-dimensional (2D) optimization problem in this study. A large *r* value produces a small *cMax*, which decreases the search area around the target position. The value of *r* is nonzero and smaller than the norm difference between the lowest and furthest grasshoppers in the search space. The same concept was applied to hyperdimensional problems in this study. This method enables the MGOA to explore only the selected region and exploit a solution in less time than is possible.

A linear decrease in the *MinDS* value is shown in Fig. 4. Although similar to coefficient *c* in the original GOA in terms of a linearly decreasing parameter, the value for *MinDS* does not start with unity but instead starts with the value calculated using (29). During the early iterations, a large *MinDS* value ensured that the grasshoppers explored the search space for possible solutions (i.e., the exploration stage). As the number of iterations increased, the value of *MinDS* decreased. During this time, the grasshopper begins to converge to a possible solution. Near the end of the iteration, the value of *MinDS* decreased significantly, and most of the grasshoppers converged to an optimal point. However, because of the nature of the proposed social interaction, which constantly avoids the nearest neighbors, some grasshoppers still move away from the target position, although the movement is minimal. The ability of grasshoppers to move away from a target position reduces their susceptibility to the entrapment of local optima.

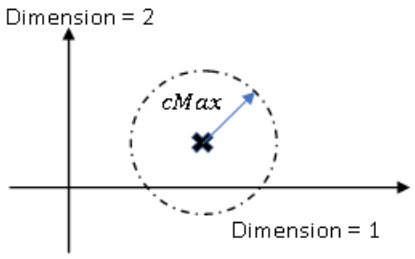


Fig. 3: *cMax* in the 2D search space
*Source: created by the authors*

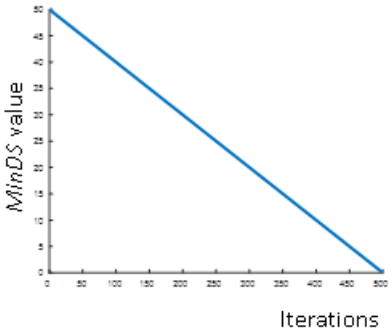


Fig. 4: Variation in *MinDS* using LEDW for 500 iterations with *r*=4, *lb*=-100, and *ub*=100
*Source: created by the authors.*

### 3.4 MGOA Implementation

The proposed Modified Grasshopper Optimization Algorithm (MGOA) employs LEDW in conjunction with the proposed social interaction methodology. The pseudocode for the MGOA is provided in Algorithm 1, and Table 1 lists the parameter settings used for the algorithm simulations. The parameters for the GOA were derived from the original research to evaluate the algorithm's performance, [37].

The algorithm begins by initializing the parameters and subsequently populating a set of grasshoppers with random positions and velocities. Subsequently, the fitness value of each grasshopper was calculated, and the optimal fitness value among all grasshoppers was designated as the target fitness $T$.

A primary loop iterates based on a predetermined number of iterations, $L$. Within each iteration, a nested loop operates to cycle through each grasshopper, thereby implementing the proposed social interaction method. Prior to each iteration using the set of grasshoppers, the *NND* and *MinDS* were computed at each iteration. As illustrated in Fig. 2, the *j*-th grasshopper is deemed a neighbor to the *i*-th grasshopper if

$$0 < d_{i,j} \le NND \tag{31}$$

and is classified as a non-neighbor; otherwise. Two fundamental conditions serve to differentiate grasshoppers: the existence and nonexistence of neighbors.

When neighbors are present, two nested conditions are instituted within the primary condition: the existence of nearest neighbors presented as

$$0 < d_{i,j} \le MinDS \tag{32}$$

and the absence of nearest neighbors, that is, the presence of close-by neighbors where

$$MinDS < d_{i,j} \le NND \tag{33}$$

Each outer nested condition houses two further nested conditions: (i) the current grasshopper's fitness is superior to that of its neighbors, and (ii) the current grasshopper's fitness is inferior to that of one or more of its neighbors. Based on these nested conditions, the position and velocity of the *i*-th grasshopper were updated.

In the absence of neighbors, the updated position and velocity calculations of the i-th grasshopper are derived by adding the current position value to a unit vector JLF computed using (22). The updated position and velocity are then determined using (23) and (24), respectively.

The process of updating the grasshopper position using the MGOA was iteratively repeated for all grasshoppers. Any grasshopper straying out of the bounds was repositioned within the upper and lower bounds. Using the updated position $X$, the grasshopper with the best fitness value was identified. If the fitness value of the grasshopper exceeds the current value $T$, its position is designated as the target position for the subsequent iteration. Otherwise, the current $T$ value was retained for the next iteration. This process was performed for a specified number of iterations.

| **Algorithm 1 Pseudocode for MGOA** | |
|---|---|
| **1** | Set and calculate parameter values *L*, *N*, *r*, *cMin*, $\alpha$, $\beta$ |
| **2** | Generate the initial population of Grasshoppers, $X$, randomly |
| **3** | Generate the initial velocity of Grasshoppers, $V$, randomly |
| **4** | Evaluate the fitness $f(X)$ of each grasshopper $X$. |
| **5** | $T$ = best solution |
| **6** | **while** $(iter < L)$ **do** |
| **7** | Calculate $MinDS$ using (29)<br>Calculate *NND* using (6) |
| **8** | **for** $i = 1$ to *N* (all *N* grasshoppers in the population)<br>**do** |
| **9** | Calculate the distance between the current grasshopper with other grasshoppers using (4)<br>Identify $X_i$ neighbors and non-neighbors.<br>Identify $X_i$ close-by and nearest neighbors. |
| **10** | **if** neighbors exist, **then** |
| **11** | **if** neareast neighbors exist **then** |
| **12** | **if** $X_i$ fitness is better than the neighbors **then**<br>calculate $X_i^{new}$ using (9)<br>calculate $V_i^{new}$ using (11)<br>**else**<br>calculate $X_i^{new}$ using (12) |

| **Algorithm 1 Pseudocode for MGOA** | |
|---|---|
| | calculate $V_i^{new}$ using (14)<br>**end if** |
| **13** | **else** |
| **14** | **if** $X_i$ fitness is better than neighbors **then**<br>calculate $X_i^{new}$using (18)<br>calculate $V_i^{new}$using (19)<br>**else**<br>calculate $X_i^{new}$ using (20)<br>calculate $V_i^{new}$ using (21)<br>**end if** |
| **15** | **end if** |
| **16** | **else** |
| **17** | calculate the unit vector *JLF* using (22)<br>calculate $X_i^{new}$using (23)<br>calculate $V_i^{new}$using (24) |
| **18** | **end if** |
| **19** | **end for** |
| **20** | Evaluate the fitness $f(X)$ of each grasshopper. Bring the grasshopper back if it goes outside the boundaries. |
| **21** | Update $T$ if there is a better solution |
| **22** | $iter = iter + 1$ |
| **23** | **end while** |
| **24** | Return the best solution $T$ |

Table 1. Configuration parameters of the selected algorithms

| Algorithm | Parameter setting |
|---|---|
| GOA | $N = 30, L = 500\ c_{max} = 1, c_{min} = 0.00004, f = 0.5, l = 1.5$ |
| MGOA | $N = 30, L = 500, cMin = 0.001, r = 5, \boldsymbol{\alpha} = 200, \boldsymbol{\beta} = 1.5$ |

*Source: created by the authors.*

# 4 Performance Evaluation

All algorithms explored in this study were developed and executed using MATLAB R2021a on a workstation featuring an Intel(R) Core(TM) i7-9750H CPU operating at 2.60GHz with 32GB of RAM. The performance of the proposed MGOA was assessed using four experimental configurations.

The first experiment compared the performance of GOA and MGOA by analyzing the mean values, standard deviations, and optimal solutions obtained from ten classical benchmark functions. Five different operators were applied, resulting in the evaluation of 50 standard functions. The second experiment focused on an in-depth examination of the convergence behaviors of both GOA and MGOA.

The third experiment assessed the performance of the MGOA using nonparametric statistical methods, including Wilcoxon's signed-rank test and Friedman's ranking test. Finally, the fourth experiment investigates the computational complexity of the MGOA to provide insights into its efficiency.

## 4.1 CEC2021 Benchmark Test Functions

This study evaluated the performance of the proposed MGOA in comparison with that of the GOA using the CEC2021 benchmark functions. The details of the CEC2021 function suite are provided in Table 2. The benchmark set comprises four categories, totaling ten distinct functions. Function F1 represents a unimodal function, whereas functions F2 to F4 correspond to basic functions. Functions F5 to F7 illustrate hybrid functions, and functions F8 to F10 pertain to composition functions, where *Nf* indicates the number of basic functions combined to form a given function. The search space for these functions was defined by the upper and lower bounds of [-100,100]*D*, where *D* represents the dimensionality of the function. Notably, the CEC2021 benchmark functions are single-objective and bound-constrained, with transformations applied to bias, shift, and rotation.

Because operators are used to parameterize these benchmark functions, the performance of the algorithm can be assessed by evaluating it under various operator configurations across all test functions. Bias, shift, and rotation transformations are represented as binary parameters, where 1 denotes activation and 0 represents deactivation. The transformations examined in this study included (000), (010), (011), (100), and (110). The transformation (000) was used for the baseline performance. Transformation (010) was used to test the translation invariance, which is the algorithm's ability to find the optimum regardless of its location. Transformation (011) was used to test rotation invariance, which was used to check the algorithm's capability to handle correlated variables. Transformation (100) was used to verify the algorithm's ability to handle function scaling and offset for the objective values. Finally, transformation (110) is used to test the combined impact of the search space relocation and objective value offset. For each combination of parameter settings and transformations, an evaluation was conducted using the mean of the best values achieved in 30 independent trial runs. The algorithm considered optimal consistently achieved the lowest mean results across most benchmark functions.

Table 2. Overview of CEC2021 real-parameter benchmark functions with bound constraints

| Types | No | Functions |
|---|---|---|
| Unimodal Functions | F1 | Shifted and Rotated Bent Cigar Function |
| Basic Functions | F2 | Shifted and Rotated Schwefel's Function |
| | F3 | Shifted and Rotated Lunacek bi-Rastrigin Function |
| | F4 | Expanded Rosenbrock's plus Griewangk's Function |
| Hybrid Functions | F5 | Hybrid Function 1 ($Nf$ = 3) |
| | F6 | Hybrid Function 2 ($Nf$ = 4) |
| | F7 | Hybrid Function 3 ($Nf$ = 5) |
| Composition Functions | F8 | Composition Function 1 (Nf = 3) |
| | F9 | Composition Function 2 (Nf = 4) |
| | F10 | Composition Function 3 (Nf = 5) |
| Search range: [-100,100]$D$, $D$ = 10, $D$ = 20 | | |

*Source: adapted from [35]*

## 4.2 Experimental Results

The evaluation of the unimodal, basic, hybrid, and composition benchmark functions employed 30 search agents for 500 iterations. The results were generated from 30 independent trials with random initial conditions for each trial. These trials calculated statistical parameters, such as mean fitness (representing performance and reliability), standard deviation (indicating stability), and best fitness (signifying optimal capability).

The optimization results of the CEC2021 benchmark test functions using the proposed MGOA and original GOA are presented in Table 3, Table 4, Table 5, Table 6 and Table 7. The convergence curves of each benchmark function type for the (000) and (111) transformations, which provide a qualitative evaluation of the proposed algorithm, are shown in Fig. 5, Figure 6, Figure 7, Figure 8, Figure 9, Figure 10, Figure 11 and Fig. 12.

The unimodal test function F1 contains a single extreme point, making it ideal for evaluating the exploitation capability of the algorithm. As shown in Table 3, Table 4, Table 5, Table 6 and Table 7, MGOA surpasses GOA in the best, mean, and standard deviation values for the (000) transformation but is less effective under other transformations. Fig. 5 shows that the MGOA converges quickly compared to the GOA, with a lower fitness value obtained at the end of the iteration. Fig. 9 shows that for the (111) transformation, the MGOA produces a better fitness value at the beginning of the iteration, but is slightly worse than the GOA at the end of the iterations. The results indicate that for problems with extreme points, such as in F1, the MGOA is affected by bias, shift, and rotation transformations and requires a larger number of iterations to produce better results. One major cause is that when a neighbor is nonexistent, the i-th grasshopper will randomly jump to a certain position instead of converging to a point that most grasshoppers are moving towards. This reduces the number of grasshoppers that can be utilized during the exploitation. Although it works well when no transformations are applied, it becomes less efficient during the implementation of transformations.

The basic functions F2–F4 in CEC2021 feature multiple local optima, making them suitable for assessing the exploration capabilities of an algorithm. The results presented in Table 3, Table 4, Table 5, Table 6 and Table 7 indicate that the MGOA performs best in 11 of the 15 benchmark tests. The MGOA demonstrated superior performance in the F2 and F3. For instance, the best value of F2 under the transformation (000) was significantly lower than that obtained with the GOA, as illustrated in Fig. 6. A similar improvement is observed in F2 with the transformation (111), as shown in Fig. 10, where the MGOA exhibits faster convergence and achieves a better final fitness value than the GOA. Although the MGOA did not deliver the best outcomes for the remaining four benchmark functions, its results remained comparable to those of the GOA. This can be attributed to the MGOA's ability to perform random jumps in the absence of neighbors, allowing it to escape local optima effectively.

The hybrid functions F5 to F7 are composed of combinations of unimodal and multimodal basic functions. These functions were designed to assess the performance of the algorithm in both exploitation and exploration. The results in Table 3, Table 4, Table 5, Table 6 and Table 7 show that the MGOA outperformed the GOA in terms of std and the majority of functions in terms of the best and mean values. MGOA underperformed in F7(011) and F7(111), but the values for both MGOA and GOA were similar. Fig. 6 shows that the MGOA converges significantly faster than the GOA and produces better fitness values for F6(000). Applying the transformation to F6 produces slightly different results, where the MGOA still converges better than the GOA, but the final fitness value is close to that of the GOA. These results demonstrate that the MGOA has good exploitation and exploration capabilities for hybrid function optimization.

Composition functions F8 to F10 were designed to evaluate the algorithm's capability to escape local optima. These functions are ideal for simultaneously benchmarking exploration and exploitation in scenarios with multiple local optima. The results in

Table 3, Table 4, Table 5, Table 6 and Table 7 indicate that MGOA outperforms GOA in 10 of the 15 functions. While the MGOA shows a slightly lower performance in the remaining five functions, the differences are minimal. As depicted in Fig. 8 and Fig. 12, MGOA converges more rapidly than GOA. For F8(000), the fitness value achieved by the MGOA was notably smaller than that of the GOA, whereas for F8(111), the values were comparable. These findings suggest that MGOA exhibits satisfactory exploitation and exploration capabilities for composition functions.

Table 3. Optimization outcomes for the transformation (000)

| Functions | Criteria | MGOA | GOA |
|---|---|---|---|
| **F1** | **best** | **1.12E+04** | 1.02E+05 |
| | **mean** | **3.40E+04** | 6.19E+05 |
| | **std** | **1.77E+04** | 4.91E+05 |
| | **rank** | **1** | 2 |
| **F2** | **best** | **7.68E+00** | 1.63E+03 |
| | **mean** | **3.13E+02** | 2.63E+03 |
| | **std** | **3.20E+02** | 5.38E+02 |
| | **rank** | **1** | 2 |
| **F3** | **best** | **1.29E+01** | 4.31E+01 |
| | **mean** | **3.69E+01** | 9.88E+01 |
| | **std** | **1.45E+01** | 3.60E+01 |
| | **rank** | **1** | 2 |
| **F4** | **best** | **6.17E-04** | 3.21E+00 |
| | **mean** | **2.91E+00** | 7.43E+00 |
| | **std** | **2.17E+00** | 2.53E+00 |
| | **rank** | **1** | 2 |
| **F5** | **best** | **1.14E+01** | 3.06E+04 |
| | **mean** | **2.40E+02** | 3.53E+05 |
| | **std** | **2.16E+02** | 2.09E+05 |
| | **rank** | **1** | 2 |
| **F6** | **best** | **6.01E-01** | 2.09E+02 |
| | **mean** | **6.70E+00** | 6.39E+02 |
| | **std** | **9.98E+00** | 2.78E+02 |
| | **rank** | **1** | 2 |
| **F7** | **best** | **5.06E+00** | 5.31E+03 |
| | **mean** | **3.95E+01** | 8.77E+04 |
| | **std** | **5.73E+01** | 7.54E+04 |
| | **rank** | **1** | 2 |
| **F8** | **best** | **5.87E-01** | 3.95E+02 |
| | **mean** | **3.77E+01** | 1.80E+03 |
| | **std** | **5.61E+01** | 9.09E+02 |
| | **rank** | **1** | 2 |
| **F9** | **best** | **7.29E-01** | 5.83E+00 |
| | **mean** | **1.59E+00** | 1.73E+01 |
| | **std** | **1.01E+00** | 1.49E+01 |
| | **rank** | **1** | 2 |
| **F10** | **best** | **4.51E+01** | 5.10E+01 |
| | **mean** | 7.68E+01 | **7.02E+01** |
| | **std** | 1.58E+01 | **1.46E+01** |
| | **rank** | 2 | **1** |

*Source: created by the authors.*

Table 4. Optimization outcomes for the transformation (010)

| Functions | Criteria | MGOA | GOA |
|---|---|---|---|
| **F1** | **best** | 6.50E+05 | **2.88E+04** |
| | **mean** | 1.91E+06 | **5.18E+05** |
| | **std** | 1.31E+06 | **4.30E+05** |
| | **rank** | 2 | **1** |
| **F2** | **best** | **4.15E+02** | 1.45E+03 |
| | **mean** | **2.01E+03** | 2.49E+03 |
| | **std** | **5.42E+02** | 6.13E+02 |
| | **rank** | **1** | 2 |
| **F3** | **best** | **6.16E+01** | 7.62E+01 |
| | **mean** | **1.08E+02** | 1.14E+02 |
| | **std** | **2.76E+01** | 2.82E+01 |
| | **rank** | **1** | 2 |
| **F4** | **best** | 4.85E+00 | **2.72E+00** |
| | **mean** | 9.56E+00 | **7.08E+00** |
| | **std** | 2.94E+00 | **2.47E+00** |
| | **rank** | 2 | **1** |
| **F5** | **best** | **4.27E+03** | 1.05E+04 |
| | **mean** | **1.03E+05** | 6.44E+05 |
| | **std** | **8.73E+04** | 6.04E+05 |
| | **rank** | **1** | 2 |
| **F6** | **best** | **5.39E+01** | 3.36E+02 |
| | **mean** | **3.18E+02** | 6.88E+02 |
| | **std** | **1.57E+02** | 2.25E+02 |
| | **rank** | **1** | 2 |
| **F7** | **best** | **6.94E+03** | 9.40E+03 |
| | **mean** | **3.81E+04** | 2.25E+05 |
| | **std** | **2.13E+04** | 1.93E+05 |
| | **rank** | **1** | 2 |
| **F8** | **best** | 1.08E+02 | **1.03E+02** |
| | **mean** | **1.13E+02** | 9.30E+02 |
| | **std** | **1.17E+00** | 1.45E+03 |
| | **rank** | **1** | 2 |
| **F9** | **best** | **4.30E+02** | 4.44E+02 |
| | **mean** | **4.92E+02** | 5.12E+02 |
| | **std** | **2.75E+01** | 5.82E+01 |
| | **rank** | **1** | 2 |
| **F10** | **best** | **4.04E+02** | 4.21E+02 |
| | **mean** | **4.56E+02** | 5.00E+02 |
| | **std** | 5.06E+01 | **2.98E+01** |
| | **rank** | **1** | 2 |

*Source: created by the authors.*

Table 5. Optimization outcomes for the transformation (011)

| Functions | Criteria | MGOA | GOA |
|---|---|---|---|
| **F1** | **best** | 8.40E+05 | **1.82E+05** |
| | **mean** | 2.24E+06 | **9.28E+05** |
| | **std** | 9.89E+05 | **8.48E+05** |
| | **rank** | 2 | **1** |
| **F2** | **best** | **1.36E+03** | 1.79E+03 |
| | **mean** | **2.08E+03** | 2.60E+03 |
| | **std** | **4.51E+02** | 4.77E+02 |
| | **rank** | **1** | 2 |
| **F3** | **best** | 8.07E+01 | **5.92E+01** |
| | **mean** | **1.27E+02** | 1.27E+02 |
| | **std** | **2.55E+01** | 3.51E+01 |
| | **rank** | **1** | 2 |
| **F4** | **best** | 5.43E+00 | **2.85E+00** |
| | **mean** | 9.94E+00 | **7.32E+00** |
| | **std** | 2.66E+00 | **2.42E+00** |
| | **rank** | 2 | **1** |
| **F5** | **best** | 5.53E+04 | **5.51E+04** |
| | **mean** | **4.12E+05** | 4.80E+05 |
| | **std** | **1.73E+05** | 4.37E+05 |
| | **rank** | **1** | 2 |
| **F6** | **best** | 1.63E+02 | **1.62E+02** |
| | **mean** | **4.29E+02** | 7.54E+02 |
| | **std** | **1.70E+02** | 2.86E+02 |
| | **rank** | **1** | 2 |
| **F7** | **best** | 5.07E+04 | **1.20E+04** |
| | **mean** | 1.28E+05 | **1.28E+05** |
| | **std** | **5.25E+04** | 1.39E+05 |
| | **rank** | 2 | **1** |
| **F8** | **best** | 1.11E+02 | **1.04E+02** |
| | **mean** | **3.85E+02** | 1.78E+03 |
| | **std** | **8.55E+02** | 1.51E+03 |
| | **rank** | **1** | 2 |
| **F9** | **best** | 4.55E+02 | **4.27E+02** |
| | **mean** | 5.33E+02 | **4.94E+02** |
| | **std** | **5.04E+01** | 5.90E+01 |

| Functions | Criteria | MGOA | GOA |
|---|---|---|---|
| | rank | 2 | **1** |
| **F10** | best | 4.14E+02 | **4.01E+02** |
| | mean | 4.76E+02 | **4.47E+02** |
| | std | **2.87E+01** | 3.61E+01 |
| | rank | 2 | **1** |

*Source: created by the authors.*

Table 6. Optimization outcomes for transformation (110)

| Functions | Criteria | MGOA | GOA |
|---|---|---|---|
| **F1** | best | 6.50E+05 | **2.88E+04** |
| | mean | 1.91E+06 | **5.18E+05** |
| | std | 1.31E+06 | **4.30E+05** |
| | rank | 2 | **1** |
| **F2** | best | **4.15E+02** | 1.45E+03 |
| | mean | **2.01E+03** | 2.49E+03 |
| | std | **5.42E+02** | 6.13E+02 |
| | rank | **1** | 2 |
| **F3** | best | **6.16E+01** | 7.62E+01 |
| | mean | **1.08E+02** | 1.14E+02 |
| | std | **2.76E+01** | 2.82E+01 |
| | rank | **1** | 2 |
| **F4** | best | 4.85E+00 | **2.72E+00** |
| | mean | 9.56E+00 | **7.08E+00** |
| | std | 2.94E+00 | **2.47E+00** |
| | rank | 2 | **1** |
| **F5** | best | **4.27E+03** | 1.05E+04 |
| | mean | **1.03E+05** | 6.44E+05 |
| | std | **8.73E+04** | 6.04E+05 |
| | rank | **1** | 2 |
| **F6** | best | **5.39E+01** | 3.36E+02 |
| | mean | **3.18E+02** | 6.88E+02 |
| | std | **1.57E+02** | 2.25E+02 |
| | rank | **1** | 2 |
| **F7** | best | **6.94E+03** | 9.40E+03 |
| | mean | **3.81E+04** | 2.25E+05 |
| | std | **2.13E+04** | 1.93E+05 |
| | rank | **1** | 2 |
| **F8** | best | 1.08E+02 | **1.03E+02** |
| | mean | **1.13E+02** | 9.30E+02 |
| | std | **1.17E+00** | 1.45E+03 |
| | rank | **1** | 2 |
| **F9** | best | **4.30E+02** | 4.44E+02 |
| | mean | **4.92E+02** | 5.12E+02 |
| | std | **2.75E+01** | 5.82E+01 |
| | rank | **1** | 2 |
| **F10** | best | **4.04E+02** | 4.21E+02 |
| | mean | **4.56E+02** | 5.00E+02 |
| | std | 5.06E+01 | **2.98E+01** |
| | rank | **1** | 2 |

*Source: created by the authors.*

Table 7. Optimization outcomes for transformation (111)

| Functions | Criteria | MGOA | GOA |
|---|---|---|---|
| **F1** | best | 8.40E+05 | **1.82E+05** |
| | mean | 2.24E+06 | **9.28E+05** |
| | std | 9.89E+05 | **8.48E+05** |
| | rank | 2 | **1** |
| **F2** | best | **1.36E+03** | 1.79E+03 |
| | mean | **2.08E+03** | 2.60E+03 |
| | std | **4.51E+02** | 4.77E+02 |
| | rank | **1** | 2 |
| **F3** | best | 8.07E+01 | **5.92E+01** |
| | mean | **1.27E+02** | 1.27E+02 |
| | std | **2.55E+01** | 3.51E+01 |
| | rank | **1** | 2 |
| **F4** | best | 5.43E+00 | **2.85E+00** |
| | mean | 9.94E+00 | **7.32E+00** |
| | std | 2.66E+00 | **2.42E+00** |
| | rank | 2 | **1** |
| **F5** | best | 5.53E+04 | **5.51E+04** |
| | mean | **4.12E+05** | 4.80E+05 |
| | std | **1.73E+05** | 4.37E+05 |
| | rank | **1** | 2 |
| **F6** | best | 1.63E+02 | **1.62E+02** |
| | mean | **4.29E+02** | 7.54E+02 |
| | std | **1.70E+02** | 2.86E+02 |
| | rank | **1** | 2 |
| **F7** | best | 5.07E+04 | **1.20E+04** |
| | mean | 1.28E+05 | **1.28E+05** |
| | std | **5.25E+04** | 1.39E+05 |
| | rank | 2 | **1** |
| **F8** | best | 1.11E+02 | **1.04E+02** |
| | mean | **3.85E+02** | 1.78E+03 |
| | std | **8.55E+02** | 1.51E+03 |
| | rank | **1** | 2 |
| **F9** | best | 4.55E+02 | **4.27E+02** |
| | mean | 5.33E+02 | **4.94E+02** |
| | std | **5.04E+01** | 5.90E+01 |
| | rank | 2 | **1** |
| **F10** | best | 4.14E+02 | **4.01E+02** |
| | mean | 4.76E+02 | **4.47E+02** |
| | std | **2.87E+01** | 3.61E+01 |
| | rank | 2 | **1** |

*Source: created by the authors.*

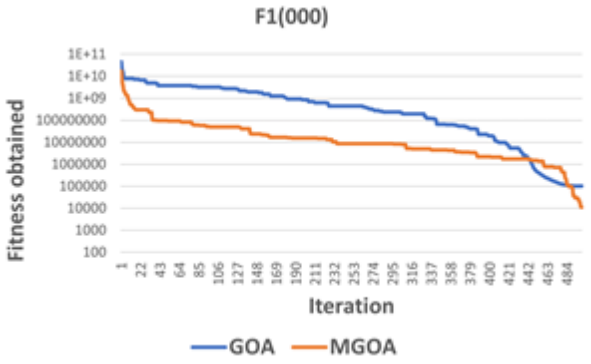


Fig. 5: Convergence trends for unimodal function F1(000)

*Source: created by the authors.*

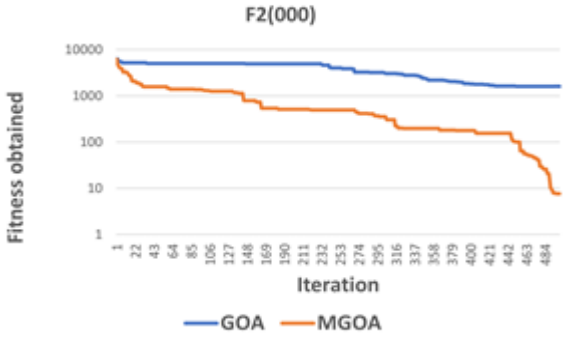


Fig. 6: Convergence trends for basic function F2(000)

*Source: created by the authors.*

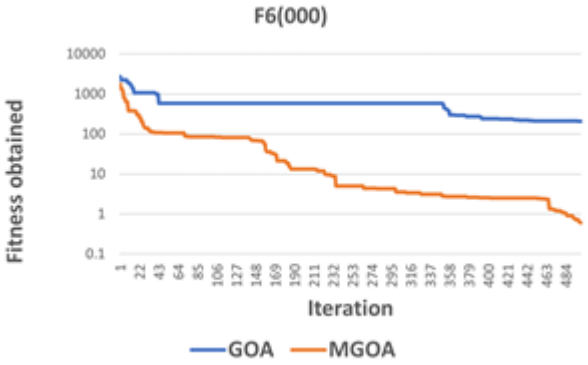


Fig. 7: Convergence trends for hybrid function F6(000)
*Source: created by the authors.*

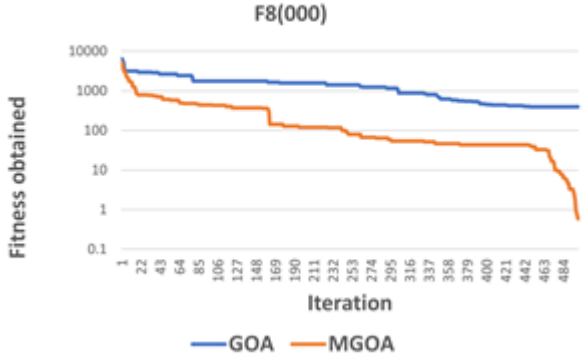


Fig. 8: Convergence trends for composite function F8(000)
*Source: created by the authors.*

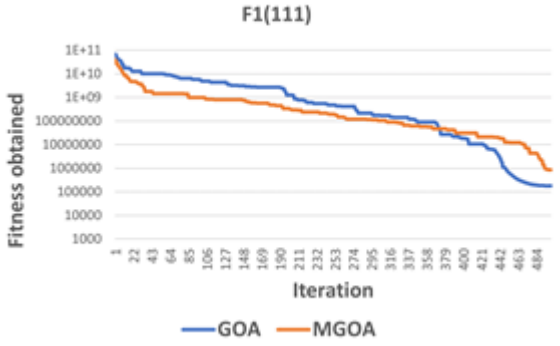


Fig. 9: Convergence trends for unimodal function F1(111)
*Source: created by the authors.*

In addition to evaluating the statistical performance based on the best, mean, and standard deviation values for the benchmark function, multiple nonparametric comparison tests were applied to further verify the results using the lowest mean values for all transformations. Two statistical methods were used: the Friedman ranking and Wilcoxon signed-rank tests. The Friedman ranking test evaluates and ranks the algorithms, assigning the lowest and highest ranks to the best- and worst-performing algorithms, respectively. The Wilcoxon signed-rank test, conducted at the 95% confidence level, was used to determine the statistical significance of the improvements introduced by the proposed algorithm.

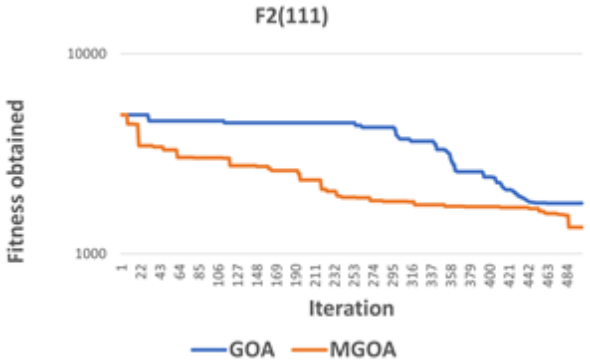


Fig. 10: Convergence trends for basic function F2(111)
*Source: created by the authors.*

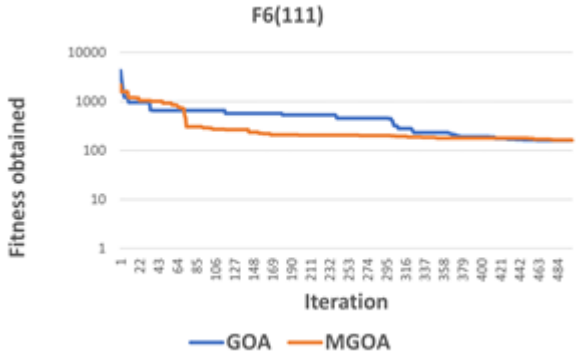


Fig. 11: Convergence trends for hybrid function F6(111)
*Source: created by the authors.*

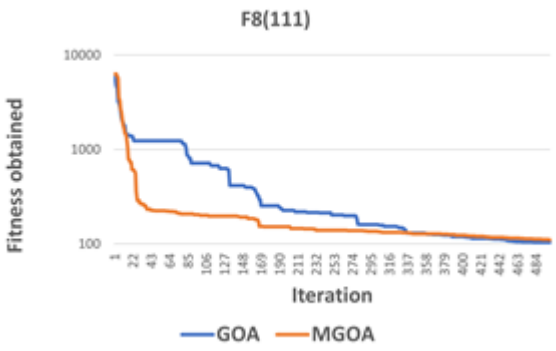


Fig. 12: Convergence trends for composite function F8(111)
*Source: created by the authors.*

The Friedman ranking test results for each benchmark function are presented in Table 3, Table 4, Table 5, Table 6 and Table 7, while the Wilcoxon signed-rank test results are provided in
Table 8, Table 9, Table 10, Table 11 and
Table 12. The summary results of the Friedman ranking and Wilcoxon signed-rank tests based on various transformations are shown in
Table 13 and *Source: created by the authors.*

Table 14, respectively.

According to the results summarized in Table 13, the MGOA achieved the best performance with a Friedman ranking sum of 65 compared to a sum of 85 for the GOA. MGOA has a lower ranking sum, which indicates a better overall ranking. MGOA outperformed GOA for all transformations except (111) and (011), where both algorithms achieved the same sum ranking.

The Wilcoxon signed-rank test confirmed the statistical significance of these improvements in the performance. The MGOA showed superior performance on 25 out of 50 benchmark functions and matched the GOA results on 12 functions. Although the MGOA performed slightly worse on 13 functions, its significant improvements and consistency on other benchmarks demonstrated its overall superiority. The Wilcoxon signed-rank test results align with the analysis of the best, mean, and standard deviation values, indicating that the MGOA provides statistically significant improvements over the GOA when applied to the CEC2021 benchmark functions.

Table 8. Results of the Wilcoxon Signed-Rank Test for Transformation (000)

| **Functions** | **Criteria** | **MGOA** |
|---|---|---|
| F1 | *p*-values | 1.733E-06 |
| | *h* | + |
| F2 | *p*-values | 1.734E-06 |
| | *h* | + |
| F3 | *p*-values | 1.734E-06 |
| | *h* | + |
| F4 | *p*-values | 3.182E-06 |
| | *h* | + |
| F5 | *p*-values | 1.734E-06 |
| | *h* | + |
| F6 | *p*-values | 1.733E-06 |
| | *h* | + |
| F7 | *p*-values | 1.734E-06 |
| | *h* | + |
| F8 | *p*-values | 1.734E-06 |
| | *h* | + |
| F9 | *p*-values | 1.734E-06 |
| | *h* | + |
| F10 | *p*-values | 0.0471617 |
| | *h* | - |

*Source: created by the authors.*

Table 9. Results of the Wilcoxon Signed-Rank Test for Transformation (010)

| **Functions** | **Criteria** | **MGOA** |
|---|---|---|
| F1 | *p*-values | 7.69E-06 |
| | *h* | - |
| F2 | *p*-values | 6.04E-03 |
| | *h* | + |
| F3 | *p*-values | 5.17E-01 |
| | *h* | = |
| F4 | *p*-values | 1.20E-03 |
| | *h* | - |
| F5 | *p*-values | 2.88E-06 |
| | *h* | + |
| F6 | *p*-values | 3.51E-06 |
| | *h* | + |
| F7 | *p*-values | 7.69E-06 |
| | *h* | + |
| F8 | *p*-values | 6.73E-01 |
| | *h* | = |
| F9 | *p*-values | 2.37E-01 |
| | *h* | = |
| F10 | *p*-values | 5.29E-04 |
| | *h* | + |

*Source: created by the authors.*

Table 10. Results of the Wilcoxon Signed-Rank Test for Transformation (011)

| Functions | Criteria | MGOA |
|---|---|---|
| F1 | *p*-values | 3.72E-05 |
| | *h* | - |
| F2 | *p*-values | 3.32E-04 |
| | *h* | + |
| F3 | *p*-values | 6.73E-01 |
| | *h* | = |
| F4 | *p*-values | 1.29E-03 |
| | *h* | - |
| F5 | *p*-values | 9.59E-01 |
| | *h* | = |
| F6 | *p*-values | 3.11E-05 |
| | *h* | + |
| F7 | *p*-values | 2.62E-01 |
| | *h* | = |
| F8 | *p*-values | 2.07E-02 |
| | *h* | + |
| F9 | *p*-values | 1.48E-02 |
| | *h* | - |
| F10 | *p*-values | 7.73E-03 |
| | *h* | - |

*Source: created by the authors.*

Table 11. Results of the Wilcoxon Signed-Rank Test for Transformation (110)

| Functions | Criteria | MGOA |
|---|---|---|
| F1 | *p*-values | 7.69E-06 |
| | *h* | - |
| F2 | *p*-values | 6.04E-03 |
| | *h* | + |
| F3 | *p*-values | 5.17E-01 |
| | *h* | = |
| F4 | *p*-values | 1.20E-03 |
| | *h* | - |
| F5 | *p*-values | 2.88E-06 |
| | *h* | + |
| F6 | *p*-values | 3.51E-06 |
| | *h* | + |
| F7 | *p*-values | 7.69E-06 |
| | *h* | + |
| F8 | *p*-values | 6.73E-01 |
| | *h* | = |
| F9 | *p*-values | 2.37E-01 |
| | *h* | = |
| F10 | *p*-values | 5.29E-04 |
| | *h* | + |

*Source: created by the authors.*

Table 12. Results of the Wilcoxon Signed-Rank Test for Transformation (111).

| Functions | Criteria | MGOA |
|---|---|---|
| F1 | *p*-values | 3.72E-05 |
| | *h* | - |
| F2 | *p*-values | 3.32E-04 |
| | *h* | + |
| F3 | *p*-values | 6.73E-01 |
| | *h* | = |
| F4 | *p*-values | 1.29E-03 |
| | *h* | - |
| F5 | *p*-values | 9.59E-01 |
| | *h* | = |
| F6 | *p*-values | 3.11E-05 |
| | *h* | + |
| F7 | *p*-values | 2.62E-01 |
| | *h* | = |
| F8 | *p*-values | 2.07E-02 |
| | *h* | + |
| F9 | *p*-values | 1.48E-02 |
| | *h* | - |
| F10 | *p*-values | 7.73E-03 |
| | *h* | - |

*Source: created by the authors.*

Table 13. Summary of the Friedman ranking test.

| Operators | GOA | MGOA |
|---|---|---|
| 000 | 19 | 11 |
| 010 | 18 | 12 |
| 011 | 15 | 15 |
| 110 | 18 | 12 |
| 111 | 15 | 15 |
| Sum | 85 | 65 |
| Rank | 2 | 1 |

*Source: created by the authors.*

Table 14. Wilcoxon signed-rank test summary

| Operators | + | = | - |
|---|---|---|---|
| 000 | 9 | 0 | 1 |
| 010 | 5 | 3 | 2 |
| 011 | 3 | 3 | 4 |
| 110 | 5 | 3 | 2 |
| 111 | 3 | 3 | 4 |
| Sum | 25 | 12 | 13 |

*Source: created by the authors.*

### 4.3 Algorithm Complexity

The methodology for evaluating the computational complexity of algorithms is outlined in [35], where the complexity is assessed by analyzing the memory usage and time required by the algorithm to solve a predefined problem with a specific input size. The computational complexity tests for the proposed MGOA and original GOA were conducted using the approach described in [35]. This methodology involves calculating $t_0$, $t_1$, $t_2$, and $\hat{t}_2$. Here, $t_0$ represents the time required to execute the specified codes, [35].

$x = 0.55$
for $i$ = 1:200000
$x = x + x; x = x / 2; x = x * x; x = sqrt(x);$
$x = \log(x); x = \exp(x); x = x/(x+2);$
end

$t_1$ corresponds to the time required to perform 200,000 evaluations of a selected benchmark function, which in this study was benchmark function F1 from the CEC2021 suite. $t_2$ denotes the execution time of the algorithm for F1 with the same number of evaluations, and $\hat{t}_2$ represents the average of $t_2$ across five independent runs [35].

This procedure was applied to both the GOA and MGOA, and the results are presented in Table 15. The final row of the table indicates that the computational cost of the MGOA was significantly lower than that of the GOA. This improvement was attributed to the proposed social interaction mechanism, which simplified the computational process while maintaining a higher level of accuracy than the original GOA.

Table 15. MGOA and GOA computational complexity

| Variable | (seconds) | |
|---|---|---|
| | MGOA | GOA |
| $t_o$ | 8.49E-03 | 8.49E-03 |
| $t_1$ | 2.44E+02 | 2.44E+02 |
| $\hat{t_2}$ | 9.09E+02 | 3.93E+03 |
| $(\hat{t_2} - t_1)/t_o$ | **7.84E+04** | 4.34E+05 |

*Source: created by the authors.*

## 5 Conclusion

This study introduces a modified version of GOA, termed MGOA. The proposed enhancements to the GOA include an adjusted social interaction mechanism and a modified linear parameter convergence strategy. The evaluation results based on the CEC2021 benchmark functions demonstrated

that MGOA outperformed GOA in most of the tested benchmark functions across various transformation combinations, including shift, bias, and rotation. The revised social interaction mechanism, along with improved parameter convergence, enhances the balance between exploitation and exploration processes compared to the original GOA. Furthermore, the proposed modifications enable the MGOA to escape local optima better than the GOA.

Future work should focus on further improving MGOA's ability to address complex problems, particularly those involving transformation combinations. Enhancements can be achieved by refining the parameter convergence mechanism that regulates exploitation and exploration in the algorithm. Although the current version of the MGOA uses linear parameter convergence, exploring nonlinear convergence strategies could potentially yield even greater improvements in algorithm performance.

*Acknowledgement:*
This research was supported by Universiti Sains Malaysia. We thank our colleagues at Universiti Sains Malaysia who provided insight and expertise that greatly assisted the research, although they may not agree with all the interpretations/conclusions of this study.

*References:*

[1] X. S. Yang, Nature-inspired optimization algorithms: Challenges and open problems, *J. Comput. Sci.*, Vol. 46, 2020, Art. no. 101104. https://doi.org/10.1016/j.jocs.2020.101104.

[2] X.-S. Yang and M. Karamanoglu, Nature-inspired computation and swarm intelligence: a state-of-the-art overview, in *Nature-Inspired Computation and Swarm Intelligence*, Elsevier, 2020, pp. 3–18. https://doi.org/10.1016/B978-0-12-819714-1.00010-5.

[3] Y. Jin, H. Wang, and C. Sun, Classical Optimization Algorithms, in *Data-Driven Evolutionary Optimization*, Studies in Computational Intelligence, Vol. 975, Springer, 2021, pp. 19–45. https://doi.org/10.1007/978-3-030-74640-7_2.

[4] M. A. Tahir, H. F. Khan, and M. M. Khan, Comparative Study of Nature-Inspired Algorithms, in *Re-imagining Diffusion and Adoption of Information Technology and Systems (TDIT 2020)*, IFIP AICT, Vol. 617, Springer, 2020, pp. 353–362. https://doi.org/10.1007/978-3-030-64849-7_32.

[5] H. R. Lourenço, O. C. Martin, and T. Stützle, Iterated Local Search, in *Handbook of Metaheuristics*, Int. Series in Oper. Res. & Manag. Sci., Vol. 57, Springer, 2003, pp. 320–353. https://doi.org/10.1007/0-306-48056-5_11.

[6] C. Voudouris and E. Tsang, Guided local search and its application to the traveling salesman problem, *Eur. J. Oper. Res.*, Vol. 113, No. 2, 1999, pp. 469–499. https://doi.org/10.1016/S0377-2217(98)00099-X.

[7] N. Mladenović and P. Hansen, Variable neighborhood search, *Comput. Oper. Res.*, Vol. 24, No. 11, 1997, pp. 1097–1100. https://doi.org/10.1016/S0305-0548(97)00031-2.

[8] Y. Meraihi, A. B. Gabis, S. Mirjalili, and A. Ramdane-Cherif, Grasshopper Optimization Algorithm: Theory, Variants, and Applications, *IEEE Access*, Vol. 9, 2021, pp. 50001–50024. https://doi.org/10.1109/ACCESS.2021.3067597

[9] D. Karaboga and B. Basturk, Artificial Bee Colony (ABC) Optimization Algorithm for Solving Constrained Optimization Problems, in *Foundations of Fuzzy Logic and Soft Computing (IFSA 2007)*, LNAI, Vol. 4529, Springer, 2007, pp. 789–798. https://doi.org/10.1007/978-3-540-72950-1_77.

[10] R. Eberhart and J. Kennedy, A new optimizer using particle swarm theory, in *MHS'95, Proc. 6th Int. Symp. Micro Machine and Human Science*, 1995, pp. 39–43. https://doi.org/10.1109/MHS.1995.494215.

[11] M. Dorigo, V. Maniezzo, and A. Colorni, Ant system: optimization by a colony of cooperating agents, *IEEE Trans. Syst. Man Cybern. Part B*, Vol. 26, No. 1, 1996, pp. 29–41. https://doi.org/10.1109/3477.484436.

[12] B. Alsalibi, L. Abualigah, and A. T. Khader, A novel bat algorithm with dynamic membrane structure for optimization problems, *Appl. Intell.*, Vol. 51, No. 4, 2021, pp. 1992–2017. https://doi.org/10.1007/s10489-020-01898-8.

[13] E. Atashpaz-Gargari and C. Lucas, Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition, in *2007 IEEE Congress on Evolutionary Computation (CEC)*, 2007, pp. 4661–4667. https://doi.org/10.1109/CEC.2007.4425083.

[14] R. V. Rao, V. J. Savsani, and D. P. Vakharia, Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems, *Comput. Des.*, Vol. 43, No. 3, 2011, pp. 303–315. https://doi.org/10.1016/j.cad.2010.12.015.

[15] Z. W. Geem, J. H. Kim, and G. V. Loganathan, A New Heuristic Optimization Algorithm: Harmony Search, *Simulation*, Vol. 76, No. 2, 2001, pp. 60–68. https://doi.org/10.1177/003754970107600201.

[16] A. Kaveh and A. Dadras, A novel meta-heuristic optimization algorithm: Thermal exchange optimization, *Adv. Eng. Softw.*, Vol. 110, 2017, pp. 69–84. https://doi.org/10.1016/j.advengsoft.2017.03.014.

[17] A. Kaveh and T. Bakhshpoori, Water Evaporation Optimization: A novel physically inspired optimization algorithm, *Comput. Struct.*, Vol. 167, 2016, pp. 69–85. https://doi.org/10.1016/j.compstruc.2016.01.008.

[18] E. Rashedi, H. Nezamabadi-pour, and S. Saryazdi, GSA: A Gravitational Search Algorithm, *Inf. Sci.*, Vol. 179, No. 13, 2009, pp. 2232–2248. https://doi.org/10.1016/j.ins.2009.03.004.

[19] R. Storn and K. Price, Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, *J. Glob. Optim.*, Vol. 11, No. 4, 1997, pp. 341–359. https://doi.org/10.1023/A:1008202821328.

[20] M. Akif Şahman, Parameter analysis of differential evolution-based oversampling approach for highly imbalanced datasets, *Int. J. Intell. Syst. Appl. Eng.*, Vol. 9, No. 2, 2021, pp. 69–80. https://doi.org/10.18201/ijisae.2021.231.

[21] H.-G. Beyer and H.-P. Schwefel, Evolution strategies – A comprehensive introduction, *Nat. Comput.*, Vol. 1, No. 1, 2002, pp. 3–52. https://doi.org/10.1023/A:1015059928466.

[22] J. H. Holland, Genetic Algorithms, *Sci. Am.*, Vol. 267, No. 1, 1992, pp. 66–72. https://doi.org/10.1038/scientificamerican0792-66.

[23] M. Kurdi, An effective genetic algorithm with a critical-path-guided Giffler and Thompson crossover operator for job shop scheduling problem, *Int. J. Intell. Syst. Appl. Eng.*, Vol. 7, No. 1, 2019, pp. 13–18.

[24] D. H. Wolpert and W. G. Macready, No free lunch theorems for optimization, *IEEE Trans. Evol. Comput.*, Vol. 1, No. 1, 1997, pp. 67–82. https://doi.org/10.1109/4235.585893.

[25] A. L. Bukar, C. W. Tan, and K. Y. Lau, Optimal sizing of an autonomous photovoltaic/wind/battery/diesel generator microgrid using grasshopper optimization algorithm, *Sol. Energy*, Vol. 188, 2019, pp. 685–696. https://doi.org/10.1016/j.solener.2019.06.050.

[26] S. Al-Serri et al., Implementation of the Grasshopper Optimisation Algorithm to Optimize Prediction and Control Horizons in Model Predictive Control-based Motion Cueing Algorithm, in *2022 IEEE Int. Conf. Systems, Man, and Cybernetics (SMC)*, 2022, pp. 3317–3323. https://doi.org/10.1109/SMC53654.2022.9945416.

[27] A. Shareef and S. Al-Darraji, Grasshopper optimization algorithm based path planning for autonomous mobile robot, *Bull. Electr. Eng. Informatics*, Vol. 11, No. 6, 2022, pp. 3551–3561. https://doi.org/10.11591/eei.v11i6.4098.

[28] H. Feng, H. Ni, R. Zhao, and X. Zhu, An Enhanced Grasshopper Optimization Algorithm to the Bin Packing Problem, *J. Control Sci. Eng.*, Vol. 2020, Art. ID 3894987, 2020, pp. 1–19. https://doi.org/10.1155/2020/3894987.

[29] Z. Elmi and M. O. Efe, Multi-objective grasshopper optimization algorithm for robot path planning in static environments, in *2018 IEEE Int. Conf. Industrial Technology (ICIT)*, 2018, pp. 244–249. https://doi.org/10.1109/ICIT.2018.8352184.

[30] S. Arora and P. Anand, Chaotic grasshopper optimization algorithm for global optimization, *Neural Comput. Appl.*, Vol. 31, No. 8, 2019, pp. 4385–4405. https://doi.org/10.1007/s00521-018-3343-2.

[31] J. Wu et al., Distributed trajectory optimization for multiple solar-powered UAVs target tracking in urban environment by Adaptive Grasshopper Optimization Algorithm, *Aerosp. Sci. Technol.*, Vol. 70, 2017, pp. 497–510. https://doi.org/10.1016/j.ast.2017.08.037.

[32] T.-C. Teng, M.-C. Chiang, and C.-S. Yang, A Hybrid Algorithm based on GWO and GOA for Cycle Traffic Light Timing Optimization, in *2019 IEEE Int. Conf. Systems, Man and Cybernetics (SMC)*, 2019, pp. 774–779. https://doi.org/10.1109/SMC.2019.8914661.

[33] B. P. Dahiya, S. Rani, and P. Singh, Lifetime Improvement in Wireless Sensor Networks Using Hybrid Grasshopper Meta-Heuristic, in *Proc. ICRIC 2019*, Lecture Notes in Electrical Engineering, Vol. 597, Springer, 2020, pp. 305–320. https://doi.org/10.1007/978-3-030-29407-6_23.

[34] M. M. Annie Alphonsa and N. MohanaSundaram, A reformed grasshopper optimization with genetic principle for securing medical data, *J. Inf. Secur. Appl.*, Vol. 47, 2019, pp. 410–420. https://doi.org/10.1016/j.jisa.2019.05.007.

[35] A. W. Mohamed, A. A. Hadi, A. K. Mohamed, P. Agrawal, A. Kumar, and P. N. Suganthan, Problem definitions and evaluation criteria for the CEC 2021 special session and competition on single objective bound constrained numerical optimization, Tech. Report, Nanyang Technol. Univ. Singapore, 2020.

[36] L. Abualigah and A. Diabat, A comprehensive survey of the Grasshopper optimization algorithm: results, variants, and applications, *Neural Comput. Appl.*, Vol. 32, No. 19, 2020, pp. 15533–15556. https://doi.org/10.1007/s00521-020-04789-8.

[37] S. Saremi, S. Mirjalili, and A. Lewis, Grasshopper Optimisation Algorithm: Theory and application, *Adv. Eng. Softw.*, Vol. 105, 2017, pp. 30–47. https://doi.org/10.1016/j.advengsoft.2017.01.004.

[38] J. Dkhili, U. Berger, L. M. Idrissi Hassani, S. Ghaout, R. Peters, and C. Piou, Self-organized spatial structures of locust groups emerging from local interaction, *Ecol. Modell.*, Vol. 361, 2017, pp. 26–40. https://doi.org/10.1016/j.ecolmodel.2017.07.020.

[39] E. H. Houssein, M. R. Saad, F. A. Hashim, H. Shaban, and M. Hassaballah, Lévy flight distribution: A new metaheuristic algorithm for solving engineering optimization problems, *Eng. Appl. Artif. Intell.*, Vol. 94, 2020, Art. no. 103731. https://doi.org/10.1016/j.engappai.2020.103731.

[40] R. N. Mantegna, Fast, accurate algorithm for numerical simulation of Lévy stable stochastic processes, *Phys. Rev. E*, Vol. 49, No. 5, 1994, pp. 4677–4683. https://doi.org/10.1103/PhysRevE.49.4677.

**Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)**

- Najwan Osman-Ali was responsible for conducting the formal analysis, carrying out the investigation, developing the methodology, implementing the software, and drafting the original manuscript.
- Junita Mohamad-Saleh oversaw supervision and project administration and contributed to reviewing and editing the manuscript.
- Wan-Amir-Fuad-Wajdi Othman provided supervision, technical consultancy, and assisted with the manuscript review.
- Haidi Ibrahim was responsible for reviewing and supervising the journal content.

**Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself.**

**No** funding was received for conducting this study.

**Conflict of Interest**

The authors declare no conflicts of interest relevant to the content of this study.