Hybrid Modified Golden Ratio Optimization Method and Sine Cosine Algorithm for Training of feed-forward neural network

Harshit Batra, Mukul Gupta, Kartik Bagri, Adwita Arora, Vijay Kumar Bohat

Department of Computer Science, Netaji Subhas University of Technology, Azad Hind Fauj Marg, Sector - 3, Dwarka, New Delhi - 110078, India

Center of Excellence in Artificial Intelligence, Netaji Subhas University of Technology, Azad Hind Fauj Marg, Sector - 3, Dwarka, New Delhi - 110078, India

Abstract

The pursuit of optimizing Feedforward Neural Networks (FNNs) has been a key research area due to its direct correlation with the enhancement of network performance across varied applications. Conventional gradient-based methods, such as backpropagation, have laid the foundation for FNN training. However, their inherent limitations, such as susceptibility to local optima, slow and sluggish convergence rates, and sensitivity to initialization choices, have left room for improvement. Recognizing these challenges in the gradient-based methods, there has been an accelerated shift towards metaheuristic algorithms, which leverage stochastic search capabilities, offering an innovative lens to explore the intricate optimization landscapes of FNNs.

In this backdrop, This study presents a novel hybrid approach by combining the capabilities of Golden Ratio Optimization Method with the Sine Cosine Algorithm, further enhanced by Lévy flight. This method, tern as the Hybrid Golden Ratio Optimization Method and Sine Cosine Algorithm(GROM-SCA), this method encapsulates diverse inspirations: the universally recognized proportionality of the Golden Ratio, the inherent oscillatory behavior of the sine and cosine functions and the Lévy flight's characteristic of promoting random, long-range exploration steps within the search space.

To evaluate the effectiveness of The GROM-SCA algorithm it was evaluated on two sets of widely recognized benchmark test functions, namely the CEC 2014 and 23 well-known benchmark functions. In these evaluations, they exhibited notable improvements in performance when contrasted with other optimization algorithms. Further, the efficiency of the MROM approach in the context of training feedforward neural networks (FNNs) was assessed. This assessment involved a comparison with numerous state-of-the-art optimization techniques across a diverse array of 30 classification datasets. The resultant findings establish the exceptional performance of GROM, solidifying its status as a superior choice among competitive optimization methods when tackling a range of classification Problems.

Keywords:
Golden Ratio Optimization Method, Sine - Cosine Algorithm, raining of feed-forward neural networks, Nature inspired algorithms, Metaheuristics.

1. Introduction

Artificial Neural Networks (ANNs) are a type of computational system that is inspired by the biological functioning of the human brain. These systems have been widely used in various fields such as pattern recognition [1], Natural Language Processing [2], Speech Recognition [3], and classification [4, 5, 6]. Their inherent capacity to approximate complex functions makes ANNs particularly adept at modeling non-linear relationships [7, 8, 9]. The architecture of an ANN comprises an interconnected assembly of artificial neurons, which receive inputs and compute corresponding

Email addresses: harshit.batra.ug20@nsut.ac.in (Harshit Batra), mukul.gupta.ug20@nsut.ac.in (Mukul Gupta), kartik.bagri.ug20@nsut.ac.in (Kartik Bagri), adwita.ug20@nsut.ac.in (Adwita Arora), vijay.bohat@nsut.ac.in (Vijay Kumar Bohat)

outputs. These neurons, akin to their biological counterparts, engage in information processing and signal transmission to other interconnected neurons. This inter-neuron communication closely mirrors the signal propagation within the human brain. Crucially, each connection within this network is assigned a "weight," and the essence of network learning involves determining the optimal values of these weights in conjunction with an additional input factor known as the "bias."

Feedforward Neural Networks (FNNs) [10, 11] stand as one of the most widely adopted and straightforward forms of artificial neural networks, extensively applied in tasks like speech recognition [12, 13], human face recognition [14, 15], and various other applications [16, 17, 18]. The architecture of a multilayer FNN has three main components: an input layer, one or more hidden layers, and an output layer. Out of which the hidden layers play a crucial role in processing information effectively, serving as intermediaries between the input and output layers. In a feedforward neural network, the flow of information is strictly unidirectional [10]. It commences with the input layer, progresses through any hidden layers, should they be incorporated (applicable especially in networks with multiple layers), and culminates in the output layer. This unidirectional flow is a defining characteristic of feedforward neural networks.

Numerous algorithms have been devised for training feedforward neural networks, with Backpropagation [19] being the most prominent among them. Backpropagation is a variant of the gradient-descent algorithm [20] and is widely favored for its effectiveness in optimizing the network's performance.The Backpropagation algorithm attempts to minimize the Mean Square Error (MSE) cost function [21]. However, it shares a common trait with many gradient-descent algorithms in that it exhibits a relatively modest convergence rate. This characteristic may render it susceptible to becoming trapped within local minima of the MSE cost function, thus limiting its ability to locate the global minimum efficiently. This highlights a significant challenge in optimizing neural networks. While Backpropagation remains an indispensable tool, its limitations necessitate ongoing research into more advanced optimization techniques that can effectively mitigate issues related to slow convergence and local minima entrapment.

Over the years, the academic community has turned to meta-heuristic algorithms as promising tools for such complex optimization problems [22, 23, 24, 25]. These algorithms have their roots in diverse inspirations, spanning Swarm Intelligence (SI), laws of physics, the principles of evolution, human behavior, adaptive strategies of plants, and the realm of mathematics.

Swarm Intelligence (SI) represents the pure essence of decentralized systems found in nature [26]. In these systems, individual agents, often simplistic in nature, interact with each other and their environment, resulting in emergent intelligent global behaviors. Algorithms taking cues from SI seek to emulate such collective behaviors and interactions to solve various optimization problems. Examples include the Artificial Bee Colony (ABC) algorithm [27], which mimics the food-foraging behavior of honey bee swarms, and the Particle Swarm Optimization (PSO) algorithm [28], an abstraction of the social behavior observed in flocks of birds. A significant contribution to this domain was made by Mendes et al. [29] They conducted an in-depth comparison of the conventional backpropagation methods with the PSO algorithm. The outcomes of their experiments exhibited the superiority of PSO in scenarios plagued with multiple local minima, making it a robust alternative in such neural network training contexts [29].

Diving into the realm of physics-inspired algorithms, one encounters methodologies that are based on the principles and laws governing the physical world [30]. These algorithms translate the dynamism of natural physical phenomena into computational strategies. The Gravitational Search Algorithm (GSA) [31, 32], for instance, derives its foundation from Newton's law of gravitation. Similarly, the Multiverse Optimizer (MVO) [33] conceptualizes the multiverse theory from cosmology, while the Lightning Search Algorithm (LSA) [34] replicates the erratic path of a lightning bolt.

Evolutionary-based algorithms find their foundation in the Darwinian principle of natural selection [35]. These algorithms mimic the process of evolution where potential solutions to an optimization problem evolve over time. The Genetic Algorithm (GA) [36], for instance, draws parallels with biological processes such as crossover (recombination), mutation, and selection. Here, potential solutions are encoded as chromosomes, and through a series of generations, these chromosomes evolve, aiming to produce fitter solutions to the problem at hand. Over time, less optimum solutions are phased out in favor of more optimum ones, replicating the survival of the fittest paradigm. Another example is Differential Evolution (DE) [37], which employs mechanisms of mutation and crossover to optimize real-valued scalar functions.

Human-based meta-heuristic algorithms are grounded in behaviors, characteristics, and cognitive processes distinctive to human beings. One noteworthy example in this category is the Cultural Algorithm (CA) [38]. This algorithm borrows from the realm of socio-cognitive theories, particularly the manner in which cultural knowledge

evolves and impacts individual and collective problem-solving abilities. Cultural Algorithms maintain a belief space that represents shared knowledge, which influences and is influenced by individual problem-solving entities. Another fascinating algorithm under this category is the Brain Storm Optimization (BSO) algorithm [39]. Inspired by the human brainstorming process.

On the mathematical frontier, several meta-heuristic algorithms have emerged, drawing upon foundational mathematical concepts and constants [40]. The Golden Ratio Optimization Method (GROM) is an example in this category. Nematollahi et al. [41] formulation of GROM harnesses the properties of the golden ratio, a number revered for its appearance across nature, art, and architectural marvels. What distinguishes GROM from many of its peers is the absence of tuning parameters. This makes it easier for researchers because they don't have to do the time-consuming and often inaccurate job of parameter calibration. Recognizing its potential, researchers have harnessed GROM in various of real-world applications. For instance, its capability was harnessed for detecting the COVID-19 virus by enhancing clustering techniques, utilizing deep residual features [42]. Another application was in the domain of power networks, where it was employed to optimize power flow in systems laden with unpredictable renewable energy sources [43]. A more interdisciplinary approach was evident when it was Hybridized with equilibrium optimization algorithms in a hybrid meta-heuristic feature selection methodology, aimed at identifying speech emotions [44].

Despite its innovative aspects, the GROM algorithm is not immune to criticism. Some researchers have identified certain limitations, such as its tendency to explore a limited range of solutions, a proclivity for slosw convergence, and occasional difficulties in escaping local optima. These shortcomings have spurred further research efforts aimed at refining the original GROM algorithm. Initial assessments suggest that these modified versions may provide a more balanced approach to both exploration and exploitation, particularly in the context of training Feedforward Neural Networks (FNNs).

The Sine Cosine Algorithm (SCA) [45], like the Golden Ratio Optimization Method (GROM), is another mathematics-based optimization algorithm. While GROM draws its inspiration from the golden ratio, SCA is centered around the periodic nature of trigonometric functions. This approach introduces a unique balance between exploration and exploitation in the search space, aiding in the identification of global and local optima.One of the defining features of the SCA is its dual-thread approach. By dividing its strategy into two distinct threads - global search and local development - SCA ensures a comprehensive scan of the solution space. While the global search employs major random fluctuations to explore unknown regions, the local development thread incorporates minor disturbances to intensively search in the vicinity of the present solution. This strategy, intrinsically rooted in the periodic oscillations of sine and cosine functions, provides SCA with a dynamic means to alternate between a broad-scale search and a fine-tuned, localized probe.

In response to the limitations observed in the GROM algorithm, a novel optimization technique known as the GROMSCA algorithm has been introduced in this study. The development of the GROMSCA algorithm was prompted by the recognizing constraints of both the Golden Ratio Optimization method (GROM) and the Sine Cosine Algorithm (SCA). The GROMSCA algorithm distinguishes itself by combining the strengths of these two established optimization techniques, GROM and SCA. The GROM method excels in efficiently exploring the search space, whereas the SCA method is known for its capacity to rapidly converge towards solutions. Through this amalgamation, the GROMSCA algorithm attains a harmonious equilibrium between exploration and convergence, yielding enhanced performance when applied to real-world problems.

To empirically validate the efficacy of the GROMSCA algorithm, an extensive evaluation was conducted on a set of 23 widely recognized benchmark problems. These benchmark problems are conventionally employed to assess the convergence characteristics of various nature-inspired optimization algorithms. The results of these evaluations demonstrated that the GROMSCA algorithm performed competitively when compared to state-of-the-art optimization techniques. In addition to these benchmark evaluations, the GROMSCA algorithm was also evaluated for the training of feedforward neural networks. The results of these evaluations showed that the GROMSCA algorithm was able to optimize the neural network with higher accuracy. Here the aim of this study is three-fold:

- To propose a novel optimization algorithm, GROMSCA, that combines the strengths of the Golden Ratio Optimization method (GROM) and Sine Cosine Algorithm (SCA) methods.
- To evaluate the effectiveness of the GROMSCA algorithm on a set of 23 well-known benchmark problems, which are commonly used to assess the convergence qualities of various nature-inspired optimization algorithms.

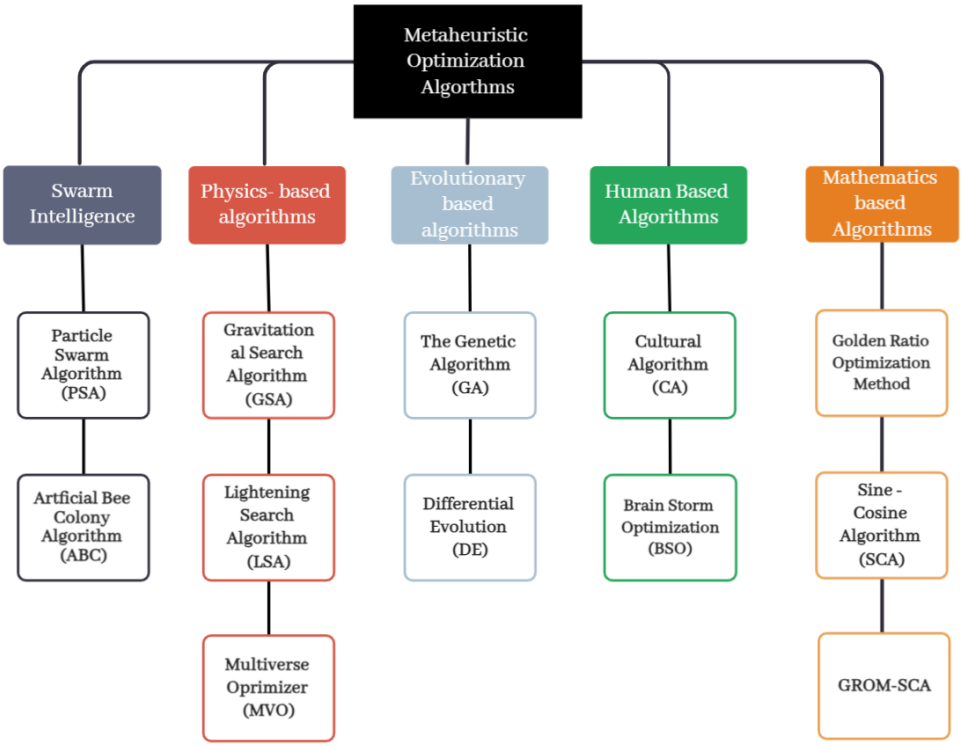


Figure 1: Classification of Metaheuristic Algorithms

- To evaluate the performance of the GROMSCA algorithm on a series of multilevel image thresholding problems, which involve identifying the most appropriate threshold value for an image.

2. Related Work

The optimization of feedforward neural networks is an area of high interest for researchers. Many different optimization algorithms have been proposed and applied to this task, each with its own strengths and weaknesses.One approach that has been proposed is the use of the Gaussian Particle Swarm Optimization (GPSO) by Min Han and Jun Wang [46]. Their results show that the model is competitive with other algorithms. Another approach is the hybrid optimization strategy proposed by Seán McLoone et. al. [47]. This strategy combines gradient-descent methods of nonlinear weight optimization with single value decomposition for linear weight decomposition. The model was found to be superior when compared with other second-order gradient methods. Yue Xue et. al. [48] proposed the SPS-PSO (self-adaptive strategy and parameter based particle swarm optimization) to optimize the weight training of FNN. When compared with other evolutionary models, their model was found to be at an advantage. Jing-Ru Zhang et. al. [49] proposed a hybrid optimization technique for training FNN combining particle swarm and backpropagation. Their findings show that this approach is better than Adaptive particle swarm and backpropagation in terms of convergent accuracy and speed. Krzysztof Socha et. al. [50] presented a hybrid method consisting of Ant Colony Optimization (ACO) in addition to short runs of gradient descent algorithms (backpropagation). Results show that their algorithm is comparable to both genetic and gradient descent based algorithms. Elham Pashaei et. al. [51] proposed a black hole algorithm (BHA) combined with complementary learning algorithms and Lévy flight random walk for training FNN. Results showed their method to outperform eight other well-known algorithms using metaheuristics in terms of accuracy. Leong Kwan Li et. al. [52] presented a new strategy called the convex combination algorithm (CCA) for the optimization of single hidden layer FNN. Their findings show that this approach proves to

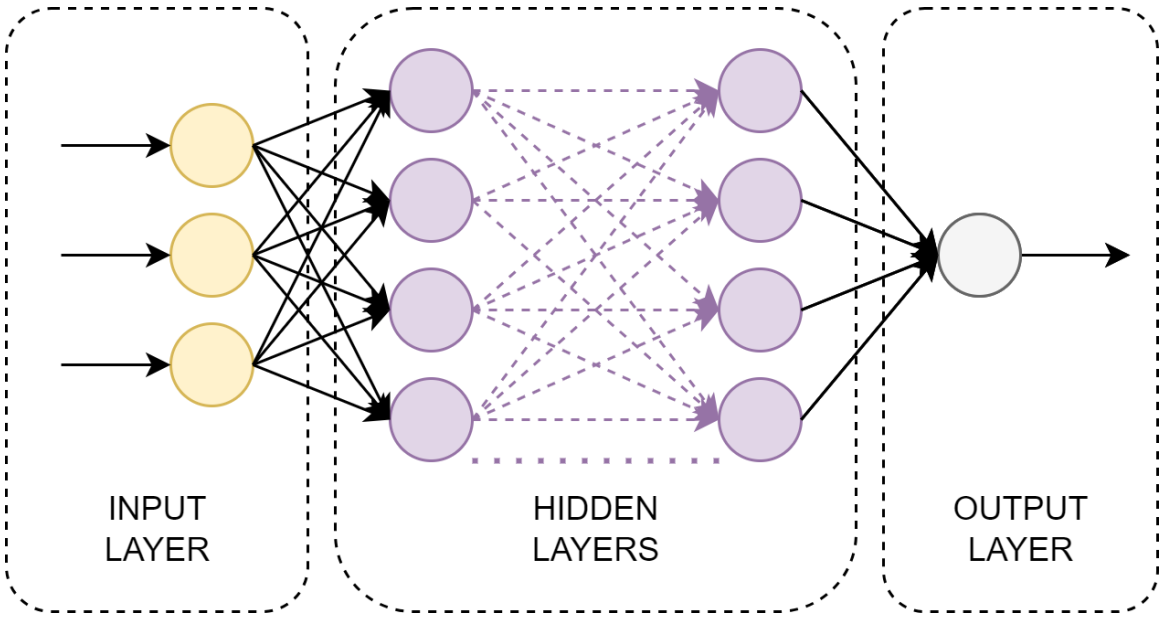


Figure 2: Multi-Layer Perceptron Architecture

be good at finding the optimal values of weights post-training. Y. Zhang et. al. [53] proposed a Bacterial Chemotaxis Optimization (BCO) approach as an alternative to the backpropagation algorithm commonly used for the training of FNN. On comparison with Genetic Algorithms (GA) and Taboo Search (TS), their approach was found to be superior. Thi-Kien Dao et. al. [54] gave a hybrid model combining improved multi-verse optimizer (MVO) and FNNs for identifying data failure in cluster heads in wireless sensor networks. Their findings show that the proposed algorithm is effective in this task. Ilyas Benmessahel et. al. [55] optimized FNN trained to improve intrusion detection systems using locust swarm optimization (LSO). On comparing with other heuristic-based algorithms like particle swarm and genetic algorithms, the LSO algorithms were found to be better. Jie-Sheng Wang et. al. [56] optimized a feedforward neural network using a hybrid approach of combining particle swarm optimization and gravitation search algorithm on sensor modelling of floatation process. Their findings suggest that the model met online soft-sensor standards. Tahir Sağ et. al. [57] used the Vortex Search (VS) Algorithm for optimization of FNN and compared it to other popular optimization algorithms like Artificial Bee Colony Optimization (ABC), Particle Swarm Optimization (PSO), Simulated Annealing (SA), Genetic Algorithm (GA) and Stochastic Gradient Descent (SGD) algorithms. Their findings suggest that VS algorithm displays competitive performance to all other algorithms.

3. Multi-layer Perceptron

Multi-Layer Perceptron (MLP)[58] are mathematical models which belong to the family of Artificial Neural Networks (ANNs) called Feed-Forward Neural Networks (FNNs). They are comprised of one or more hidden layers, with each layer containing one or multiple neurons. As an extension of the perceptron network, it is one of the most widely utilized neural network models in practice. An MLP with a single hidden layer is referred to as a shallow neural network. With an adequate number of hidden neurons, a single hidden layer MLP is capable of providing a universal approximation for nearly any problem involving tabular data. If an MLP comprises multiple hidden layers, it is known as a deep neural network. The addition of more hidden layers may not necessarily provide substantial benefits, as it can lead to a higher number of trainable parameters and increase the risk of overfitting. Fig. 2 depicts a general architecture of an MLP, which consists of three layers of nodes: an input layer, a hidden layer, and an output layer. The data flows only in a forward direction, starting from the input nodes, passing through the hidden nodes, and ending up at the output nodes. Except for the input nodes, each node serves as a neuron that incorporates a bias and conducts computations utilizing a non-linear activation function.

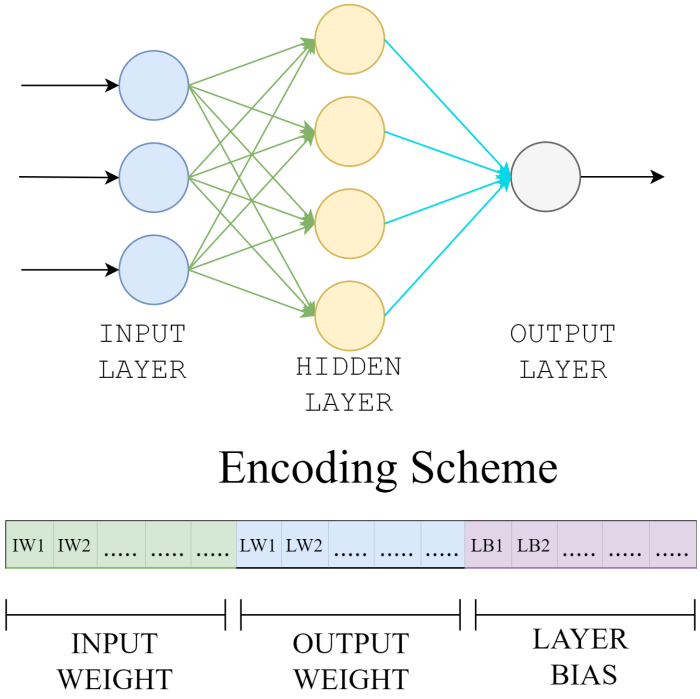


Figure 3: Encoding Scheme

3.1. Encoding Technique

The number of neurons in a feedforward neural network is determined by the task at hand. The input and output layers are determined by the number of features and classes in the dataset, while the number of neurons in the hidden layer can be estimated using the Kolmogorov theorem [59], which suggests that "the number of hidden neurons should be 2 times the number of input neurons plus 1".

The process of determining the optimal weight and bias values for a multilayer perceptron (MLP) involves representing these values in a specific format. To accomplish this, a vector encoding technique is utilized, in which the initialized solutions are expressed as a population of size N, denoted as $S = (S_1, S_2, \ldots, S_a, \ldots, S_N)$. Each member $S_a = (n_w, h_b, h_w, o_b)$ encodes the set of biases and weights for a single-layer feedforward neural network. The MLP's weight and bias values can be encoded as a vector and the size of vector is given using the following equation:

$$\mathbf{D} = (n \times h) + (h \times m) + |h_b| + |o_b|, \tag{1}$$

where n, h, and m are the number of input, hidden, and output neurons respectively, and $|h_b|$ and $|o_b|$ represent the cardinality of the set of biases of the hidden and output neurons. The weight values between the input and hidden layers are represented by n_w and the weight values between the hidden and output layers are represented by h_w. This vector encoding approach is commonly used in literature and is defined as:

$$\mathbf{P} = [w_{11}, w_{12}, \ldots, w_{1h}, w_{21}, w_{22}, \ldots, w_{2h}, \ldots, w_{n1}, w_{n2}, \ldots, w_{nh}, h_{b1}, h_{b2}, \ldots, h_{bh}, o_{b1}, o_{b2}, \ldots, o_{bm}], \tag{2}$$

where P is the encoded vector, w_{ij} denotes the weight between the i^{th} input neuron and the j^{th} hidden neuron, h_{bj} represents the bias of the j^{th} hidden neuron, and o_{bk} represents the bias of the k^{th} output neuron. Fig. 3 illustrates the coding scheme

3.2. Computing the Fitness Value in Multi-Layer Perceptron

The concept of a fitness function, also referred to as an objective function [60], is integral to assessing the effectiveness of a solution within a given problem domain. Within the context of a Multi-Layer Perceptron (MLP), the fitness value can be ascertained by following a two-step process.

The initial step involves the computation of a weighted sum for each hidden neuron, as illustrated in Eq. 3:

$$s_j = \sum_{i=1}^{n} w_{ij} x_i + h_{bj} \qquad (j = 1, 2, ..., h) \tag{3}$$

In this equation, w_{ij} represents the weight affiliated with the connection between the i^{th} node in the input layer, and the j^{th} node within the hidden layer. The output value of the i^{th} node in the input layer is denoted as x_i. Moreover, h_{bj} symbolizes the bias linked with the j^{th} node in the hidden layer.

The second phase involves calculating the output of the hidden neuron by applying an activation function to the previously calculated weighted sum of each hidden neuron. A variety of activation functions [61] exists, including ReLU [62], Leaky ReLU [63], and Tanh, among others. However, the sigmoid function is often favored due to its smooth gradient, preventing jumps in output values. The sigmoid function is represented in Eq. 4:

$$f(s_j) = \frac{1}{1 + e^{-s_j}} \tag{4}$$

Once the output of the hidden layer nodes has been determined, the output of the neurons in the output layer can be computed using Eq. 5:

$$o_k = \sum_{j=1}^{h} w_{jk} f(s_j) + o_{bk} \qquad (k = 1, 2, ..., m) \tag{5}$$

In this equation, w_{jk} designates the weight associated with the link between the j^{th} neuron in the hidden layer and the k^{th} neuron in the output layer. o_{bk}, on the other hand, indicates the bias of the k^{th} neuron in the output layer. The output of each neuron in the output layer is also subjected to the sigmoid function, as shown in Eq. 6:

$$o_k = f(o_k) = \frac{1}{1 + e^{-o_k}} \tag{6}$$

The final step in calculating the fitness value of each solution within the proposed MLP algorithm involves the utilization of the mean squared error (MSE). The MSE is computed using the weight and bias matrices given to the MLP, as per Eq. 7:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (c_i - o_i)^2 \tag{7}$$

In this equation, n denotes the number of training samples, o_i refers to the predicted values rendered by the neural network, and c_i signifies the actual class labels.

In addition to the MSE, classification accuracy serves as a valuable metric for evaluating the classification performance of the MLP on unseen data. This accuracy can be calculated using Eq. 8:

$$Accuracy = \frac{CS}{TS} \tag{8}$$

In this equation, TS represents the total number of samples present in the test dataset, while CS denotes the number of samples that have been correctly classified by the model.

This comprehensive methodology allows for a systematic and objective evaluation of the Multi-Layer Perceptron's performance. It accounts not only for the model's predictive accuracy but also for the quality of its predictions, as measured by the MSE. These metrics provide a holistic view of the MLP's fitness and its potential applicability to the problem at hand.

4. Sine Cosine Optimization Algorithm

The Sine Cosine Algorithm (SCA) [45] is an optimization algorithm inspired by trigonometric functions. It operates by initializing a random population and updating the solutions according to Eq. 9 and Eq. 10.

$$X_{i,new}^{j} = X_{i,old}^{j} + c_1 \cdot sin(c_2) \cdot |c_3 \cdot D_{i,old}^{j} - X_{i,old}^{j}| \tag{9}$$

$$X_{i,new}^{j} = X_{i,old}^{j} + c_1 \cdot cos(c_2) \cdot |c_3 \cdot D_{i,old}^{j} - X_{i,old}^{j}| \tag{10}$$

Here, $X_{i,new}^{j}$ denotes the position of i^{th} agent in the j^{th} dimension at the current iteration while $X_{i,old}^{j}$ denotes the position of i^{th} agent in the j^{th} dimension at the previous iteration. The variables c_1, c_2, and c_3 are random constants, and $D_{i,old}^{j}$ is the destination point of the i^{th} agent in the j^{th} dimension in the previous iteration. This destination point is the location of the best solution found so far. this is illustrated in the Fiq. 4

Traditionally, Eq. 9 and Eq. 10 are unified into a single equation, where the decision to update based on either equation is equally probable. This is formulated as follows:

$$X_{i,new}^{j} = \begin{cases} X_{i,old}^{j} + c_1 \cdot \sin(c_2) \cdot |c_3 \cdot D_{i,old}^{j} - X_{i,old}^{j}|, & \text{if } k < 0.5 \\ X_{i,old}^{j} + c_1 \cdot \cos(c_2) \cdot |c_3 \cdot D_{i,old}^{j} - X_{i,old}^{j}|, & \text{if } k \geq 0.5 \end{cases} \tag{11}$$

In this equation, k is a random number within the range [0,1]. The parameter c_1 is critical in determining the extent of exploration and exploitation in the SCA. In the initial stages of the algorithm, a larger value for c_1 is preferred to facilitate effective exploration. As the algorithm progresses, this value is gradually reduced according to Eq. 12:

$$c_1 = a \cdot (1 - t/T) \tag{12}$$

In this equation, a is a constant, t is the current iteration number, and T is the total number of iterations permitted. The Sine Cosine Algorithm is detailed in Algorithm 1 and the Fig. **??** provides its graphical representation.

Algorithm 1: Sine Cosine Algorithm (SCA)

1: Initialize population
2: Specify the maximum number of iterations, T
3: **for** $t = 0 \rightarrow T - 1$ **do**
4: Evaluate the cost function for each individual solution.
5: Update the global best solution if a better one is found.
6: Update the parameters c_1, c_2, c_3, and k.
7: Update the position of each individual solution according to Eq. 9 or Eq. 10 based on the value of k.
8: **end for**

5. Golden Ratio Optimization Method

The Golden Ratio Optimization Method [41] is a derivative-free optimization algorithm that uses the golden ratio to search for the optimal solution to a problem.

5.1. First Phase

The first phase of the Golden Ratio Optimization Method (GROM) involves the random initialization of search agents within the search space of the problem. This is done by using the following equation:

$$X_i^d = LB^d + rand \times (UB^d - LB^d) \qquad i = 1, 2, 3.....N \tag{13}$$

where N is the number of search agents, X_i^d denotes the position of the i^{th} agent in the d^{th} dimension, and *rand* is a random value between [0,1]. The upper and lower bounds of the search space for the d^{th} dimension are represented

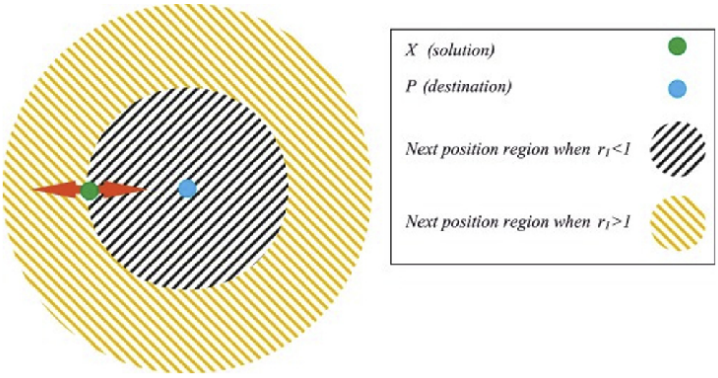


Figure 4: Next Position Updation in Sine Cosine Algorithm[https://www.researchgate.net/publication/321271206˙A˙novel˙hybrid˙GW SCA˙approach˙for˙optimization˙problems cite this]

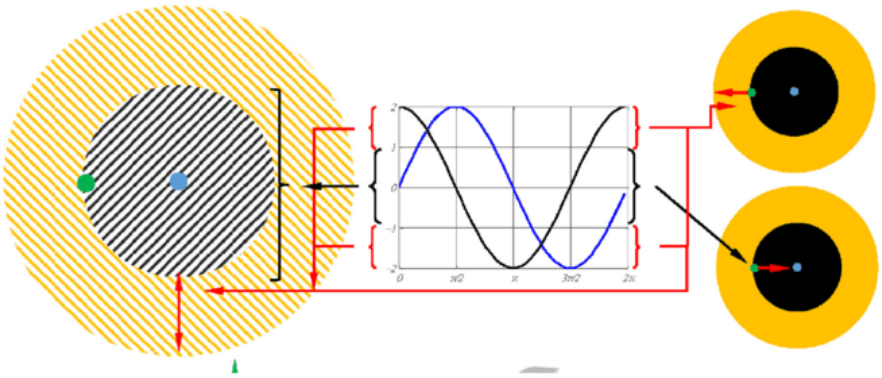


Figure 5: Fundamental Principle of the Sine Cosine Algorithm

by UB^d and LB^d, respectively. It is important to check that the search agents are within the predefined search space to avoid invalid solutions.

After the search agents have been randomly initialized, the worst agent is identified as X_{worst}, and the average position of all search agents is computed and denoted as $X_{average}$. The fitness value of each search agent is evaluated based on the objective function of the problem being solved. If the fitness value of $X_{average}$ is better than that of X_{worst}, then $X_{average}$ is substituted for X_{worst}.

5.2. Second Phase

In the second phase of the Golden Ratio Optimization Method (GROM), the population set is iterated over for each search agent, denoted by X_i. A random search agent, X_j, where $i \neq j$, is selected, and a comparison is made between X_i, X_j, and the average of the population set, $X_{average}$. The best among the three is assigned as X_b, the worst as X_w, and the last one as X_m. The fitness values of the three agents are ordered based on the inequality:

$$F_{best} < F_{medium} < F_{worst} \tag{14}$$

Next, a vector $\vec{X_t}$ is calculated as the difference between the medium and worst agents:

$$\vec{X_t} = \vec{X_m} - \vec{X_w} \tag{15}$$

To determine the magnitude of movement that will take place in the direction of the obtained vector X_b, the golden number and the Fibonacci formula are employed. The Fibonacci formula is applied as follows:

$$F_t = GF * \frac{\phi^t - (1-\phi)^t}{\sqrt{5}} \tag{16}$$

Here, GF is a constant with a value of 1.1618, t is the inverse of the maximum number of iterations, and ϕ is the golden ratio constant.

The new search agent position X_{new} is then calculated using the equation:

$$X_{new} = (1 - F_t)X_b + randF_t\vec{X_t} \tag{17}$$

Here, *rand* is a random value between 0 and 1. The new position is then evaluated to determine whether it improves the fitness of the current search agent. If it does, the search agent is updated with the new position. Otherwise, the search agent retains its old position. This process is summarized in the following equation:

$$X_i = \begin{cases} X_{new}, & \text{if the fitness improves} \\ X_{old}, & \text{otherwise} \end{cases} \tag{18}$$

Here, X_i represents the position of the i^{th} agent, X_{new} is its new position and X_{old} is its previous position.

5.3. Third Phase

The third phase of the Golden Ratio Optimization Method (GROM) aims to approach the optimal agent. This is accomplished by using the Golden ratio to update the search agents through the following equation:

$$X_{new} = X_{old} + rand * \frac{1}{GF} * X_{best} \tag{19}$$

Here, X_{new} represents the updated position of the search agent, X_{old} represents its previous position, *rand* is a randomly generated value between [0,1], GF is a constant with a value of 1.1618, and X_{best} represents the best agent among all the agents in the population. This equation helps the search agents approach the optimal agent.

The algorithm checks if the updated position is within the specified search space, and if not, brings it back into the search space. If the updated position results in improved fitness for the current search agent, the new position replaces the previous one as shown in the following equation:

$$X_i = \begin{cases} X_{new} & \text{if the fitness improves} \\ X_{old} & \text{otherwise} \end{cases} \tag{20}$$

Here, X_i represents the position of the i^{th} agent, X_{new} is its new position and X_{old} is its previous position. The algorithm continues this process until either a satisfactory solution is found or the maximum number of iterations is reached.

Algorithm 2: Golden Ratio Optimization Method (GROM)

Initialize population
Evaluate the fitness $f(X_i)$ for each search agent X_i
Set the best solution to be X_{best}, worst solution to be X_{worst}, and arithmetic mean of the population to be $X_{average}$
if Fitness of $X_{average} < X_{worst}$ **then**
    $X_{worst} = X_{average}$
end if
Check for Boundary condition
while $t < t_{total}$ **do**
    for $i = 1$ `to` N `(where` N `is population size)` **do**
        randomly select X_j where $i \neq j$
        Among X_i, X_j, and $X_{average}$, set the best solution to be X_b, worst solution to be X_w, and middle one to be X_m, respectively
        Update the search agent's location based on the Eqs. 14-18
    end for
    for $i = 1$ `to` N `(where` N `is population size)` **do**
        Update the search agent's location based on the Eqs. 19-20
    end for
    Check for Boundary condition
end while
Output the best solution

6. Lévy Flight

The phenomenon of Lévy flights[64], characterized by a sequence of small movements interspersed with occasional longer jumps as seen in Fig. 6, is an inherent feature of various stochastic processes in nature. Many animal search patterns can be described by Lévy flights, particularly when searching for food in a sparse environment. The effectiveness of these search patterns has inspired the development of various computational algorithms for optimization and search problems, such as Lévy flight particle swarm optimization (LFPSO)[65], Lévy-flight firefly algorithm(LFFA)[66] and Lévy-flight grasshopper optimization algorithm (LFGOA)[67].

The Lévy flight is characterized by random walks whose steps, denoted by s, follow a heavy tail probability distribution known as the Lévy distribution, which is formulated as:

$$L(s) \sim |s|^{-1-\beta}, \quad \beta \in (0, 2) \tag{21}$$

Here, β is the parameter that determines the shape of the Lévy distribution, and it is inversely proportional to the generated random step size.

The step sizes in Lévy flights are generated using two Gaussian-distributed variables, u and v, each with standard deviations σ_u and σ_v, respectively. The standard deviations and the variables are derived as follows:

$$\sigma_u = \left(\frac{\Gamma(1+\beta) \cdot \sin(\pi \cdot \frac{\beta}{2})}{\Gamma(\frac{1+\beta}{2}) \cdot \beta \cdot 2^{\frac{\beta-1}{2}}} \right)^{\frac{1}{\beta}} \tag{22}$$

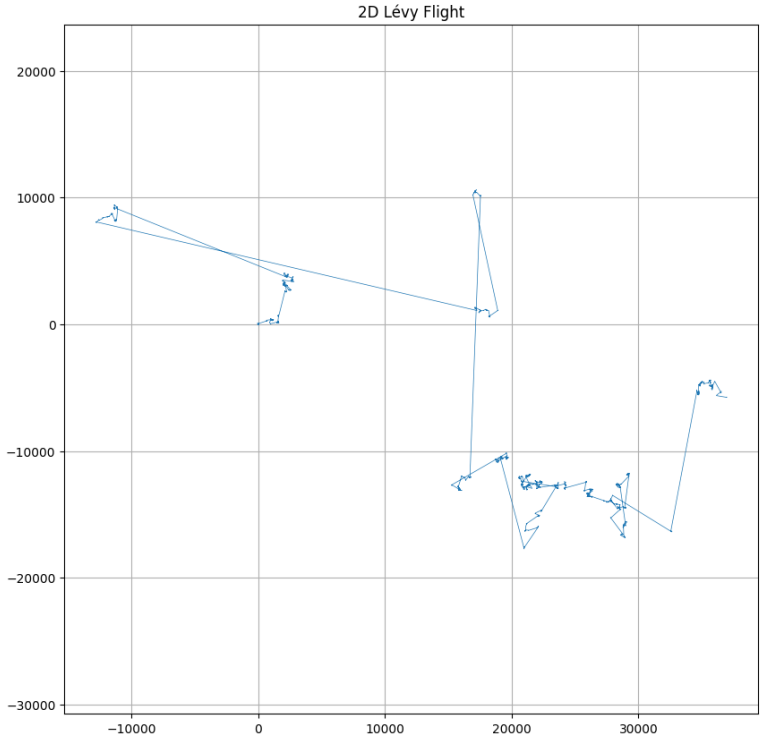


Figure 6: 2D Lévy Flight: Random Steps with Heavy-Tailed Distributions

$$\sigma_v = 1/\sigma_u \tag{23}$$

$$u \sim N(0, \sigma_u^2) \tag{24}$$

$$v \sim N(0, \sigma_v^2) \tag{25}$$

$$s = u|v|^{1/\beta} \tag{26}$$

By adopting the behaviour of Lévy flight in nature, nature-inspired algorithms can increase their robustness and versatility, providing a broader exploration of the search space and more efficient convergence to optimal solutions.

7. Proposed Algorithm

The Golden Ratio Algorithm (GROM) is a powerful optimization algorithm that is commonly used to find the best solutions to complex problems. However, one of the main drawbacks of GROM is its tendency to prematurely converge to local optima. This can result in suboptimal solutions, which may not be the best possible outcome for the problem at hand. To address this issue, a hybrid approach that combines the Golden Ratio Algorithm with other optimization techniques has been proposed.

The first phase of the proposed algorithm remains the initialization and evaluation of each search agent. The second phase of the hybrid algorithm is similar to the Golden Ratio optimization method. This is done because GROM has powerful exploration capabilities and is able to identify promising areas in the search space. The third phase of the hybrid algorithm is the sine-cosine algorithm, which is known for its ability to efficiently converge to a global optimum. The hybrid algorithm has Lévy flight embedded into both the second and third phases of the

algorithm. The Lévy flight distribution is a random walk algorithm that is known for its ability to explore the solution space of a problem in a more efficient manner. By using this algorithm, the hybrid algorithm can explore the entire solution space of the problem and find the best possible solution.

The combination of these three algorithms results in a powerful optimization algorithm that is able to find the best solutions to complex problems. The hybrid algorithm is able to explore the entire solution space of the problem and find the best possible solution by combining the strengths of the Sine-Cosine Algorithm, the Golden Ratio Algorithm, and the Lévy flight distribution.

One of the key advantages of this hybrid algorithm is its ability to avoid premature convergence to local optima. This is achieved by using the sine-cosine algorithm, which is a global optimization algorithm, and the Lévy flight distribution, which is a random walk algorithm that is able to explore the entire solution space of the problem.

The next few sections provide a detailed phase-wise explanation of the proposed algorithm

7.1. Phase 1

The proposed algorithm is a population-based heuristic, which means it utilizes a group of search agents to explore the search space and find the optimal solution. The algorithm begins by initializing the population of search agents randomly on the search space. This is done by assigning random values to the coordinates of each agent in the problem's dimensions. The position of the a^{th} agent in the d^{th} dimension is represented by x_a^d, where a ranges from 1 to K, and K is the number of agents in the search space. The number of dimensions in the problem is represented by dim. Eq27 Mathematically

$$X_a = x_a^1, ..., x_a^d, ..., x_a^{dim} \qquad a = 1, 2, ..., K \tag{27}$$

This initialization is done using the following equation.

$$x_a^d = LB^d + rand \times (UB^d - LB^d) \qquad d = 1, 2, 3.....dim \tag{28}$$

where $rand$ is a random value between [0,1] and the upper and lower bounds of the search space for the d^{th} dimension are represented by UB^d and LB^d, respectively. It is important to check that the search agents are within the predefined search space to avoid invalid solutions.

After initializing the search agents, a boundary check is performed to ensure that all agents are within the problem's search space. The fitness of each search agent is then evaluated, which measures how well the agent's coordinates correspond to the optimal solution. The agent with the best fitness is designated as X_{Best}, the agent with the worst fitness is designated as X_{worst}, and the arithmetic mean of all agent's coordinates is designated as $X_{average}$. The fitness of $X_{average}$ is also evaluated, and if it is found to be better than X_{worst}, X_{worst} is replaced with $X_{average}$.

The algorithm then proceeds to repeat the process of evaluating the fitness of each search agent, updating X_{Best}, X_{worst}, and $X_{average}$ until a stopping criterion is met. This can include reaching a maximum number of iterations or finding a solution with a fitness value that meets a predefined threshold. The end result is the optimal solution, represented by the coordinates of X_{Best}.

7.2. Phase 2

In the second phase of the proposed algorithm, for each search agent X_i, another search agent X_j is randomly selected, where $i \neq j$. The fitness of these two agents, along with $X_{average}$, is then compared to determine the best, worst, and medium agents. This is represented by the equation:

$$F_{best} < F_{medium} < F_{worst} \tag{29}$$

where F represents the fitness of the corresponding agent. The best agent is labelled as X_b, the worst as X_w, and the last one as X_m.

To determine the magnitude of movement that will take place in the direction of the obtained vector X_b, the golden number and the Fibonacci formula are employed. The Fibonacci formula is applied as follows:

$$F_t = GF * \frac{\phi^t - (1-\phi)^t}{\sqrt{5}} \tag{30}$$

$$GF = 1.1618 \tag{31}$$

$$t = 1/t_{max} \tag{32}$$

where F_t is the magnitude of movement, GF is the golden number, ϕ is the golden ratio, t is the current iteration, and t_{max} is the maximum number of iterations.

The search agents in population-based evolutionary optimization algorithms are updated in such a way that the overall updating produces a better objective function. The main goal of this update is to improve the population's best solution. A random movement is also introduced to the new solution to search the whole domain of the problem. The following equation is used to update the solutions:

$$Xnew = (1 - F_t)X_b + rand * F_t * X_t : \tag{33}$$

where X_{new} is the updated position of the agent, *rand* is a random number between 0 and 1, and X_t is the vector obtained from the comparison of X_i, X_j, and $X_{average}$.

If the new position of the agent improves its fitness, it is updated, otherwise, the old position is retained as shown in eq.34. If the new position of the agent is found to be outside of the search space, the agent is brought back to the search space.

$$\begin{cases} X^i = X^i_{new} & If\ the\ fitness\ improves \\ X^i = X^i_{old} & otherwise \end{cases} \tag{34}$$

7.3. Phase 3

The updating mechanism in this phase involves two parts: sine-cosine update and Lévy flight update. In the sine-cosine update, a random value c is generated between 0 and 1. If $c < 0.5$, the positions of search agents are updated using the sine-cosine algorithm, which helps to exploit the regions around the current solutions. The updating equations for the sine-cosine algorithm are given by equations (35) and (36), where $X^i_{new,j}$ and $X^i_{old,j}$ represent the new and old positions of the i^{th} agent in the j^{th} dimension, r_1, r_2, and r_3 are random numbers, and X^{best}_j is the position of the best agent in the j^{th} dimension.

On the other hand, the Lévy flight update helps to explore new solutions in the search space. In this update, a random value b is generated between 0 and 1, and the position of the agent is updated using the equations (37) and (38), where τ is the metal ratio, *rand* is a random number between 0 and 1, $\odot$ represents element-wise multiplication, and Lévy(dim) is the Lévy flight distribution in the given dimension.

After the new positions are calculated using either the sine-cosine update or the Lévy flight update, the fitness of the agents in the new positions is evaluated. If the fitness improves, the agent is updated to the new position, as given by equation (39). Otherwise, the agent retains its old position. Additionally, if the new position of the agent is found to be outside of the search space, the agent is brought back to the search space.

For the sine-cosine update, a random value c is generated between 0 and 1. If $c < 0.5$, the positions of search agents are updated using the sine-cosine algorithm. The following equations are used for the update:

$$X^inew, j = X^iold, j + r_1 \cdot \sin(r_2) \cdot |r_3 \cdot X^{best} j - X^iold, j| \tag{35}$$

$$X^inew, j = X^iold, j + r_1 \cdot \cos(r_2) \cdot |r_3 \cdot X^{best} j - X^iold, j| \tag{36}$$

where $X^i_{new,j}$ and $X^i_{old,j}$ represent the new and old positions of the i^{th} agent in the j^{th} dimension, r_1, r_2, and r_3 are random numbers, and X^{best}_j is the position of the best agent in the j^{th} dimension.

For the Lévy flight update, a random value b is generated between 0 and 1. The position of the agent is updated using the following equations:

$$X^inew = X^iold + \frac{1}{\tau} \cdot rand \cdot (X^{best} - X^{worst}) + \odot \cdot \text{Lévy}(dim) \tag{37}$$

$$X^inew = (X^iold - X^{best}) + \frac{1}{\tau} \cdot rand \cdot (X^{average} - X^{worst}) + \odot \cdot \text{Lévy}(dim) \tag{38}$$

where ϕ is the golden ratio, *rand* is a random number between 0 and 1, $\odot$ represents element-wise multiplication, and Lévy(dim) is the Lévy flight distribution in the given dimension.

If the new position of the agent improves its fitness, it is updated; otherwise, the old position is retained. If the new position of the agent is found to be outside of the search space, the agent is brought back to the search space.

$$X^i = \begin{cases} X^i_{new}, & \text{if the fitness improves} \\ X^i_{old}, & \text{otherwise} \end{cases} \tag{39}$$

Algorithm 3: GROM-SCA Optimization Algorithm

Initialize X_w, X_{avg}, and X_b with zeros
Set X_w score to $-\infty$, X_{avg} score and best score to ∞
Check if *lb* and *ub* are lists
Initialize positions of search agents
Start Main loop
while iteration $l < Max_{iter}$ **do**
    Calculate X_{avg} position
    for $i = 1$ `to` $SearchAgents_{no}$ **do**
        Clip positions that go beyond boundaries
        Calculate fitness for each agent
        if current fitness > X_w score **then**
            Update X_w and its score
        end if
        if current fitness < best score **then**
            Update best score and position
        end if
    end for
    if X_{avg} score < X_w score **then**
        Update X_w based on X_{avg}
    end if
    for $i = 1$ `to` $SearchAgents_{no}$ **do**
        Phase 1: Update positions based on X_{avg}, X_w, and other agents
    end for
    for $i = 1$ `to` $SearchAgents_{no}$ **do**
        Phase 2: Update positions based on best position and Levy flight
    end for
    Update Convergence curve
    Print best fitness
end while
End of main loop
Output the best solution

8. Computational Complexity Analysis of GROM, SCA and GROM-SCA

This Section investigates and compares the computational complexity of the proposed GROM-SCA and its base algorithm GROM and SCA.

8.1. Computational Complexity of SCA

The SCA algorithm begins with the initialization of the search agent this is a *O(nd)* time operation here *n* represents the number of search agents to be initialized and *d* is the predefined dimension of the problem. An iterative evaluation and updating of the population is performed. Evaluation is a *O(nc)* time process for each iteration where n is the number of search agents and c is the computational cost of function evaluation making a total computational cost for

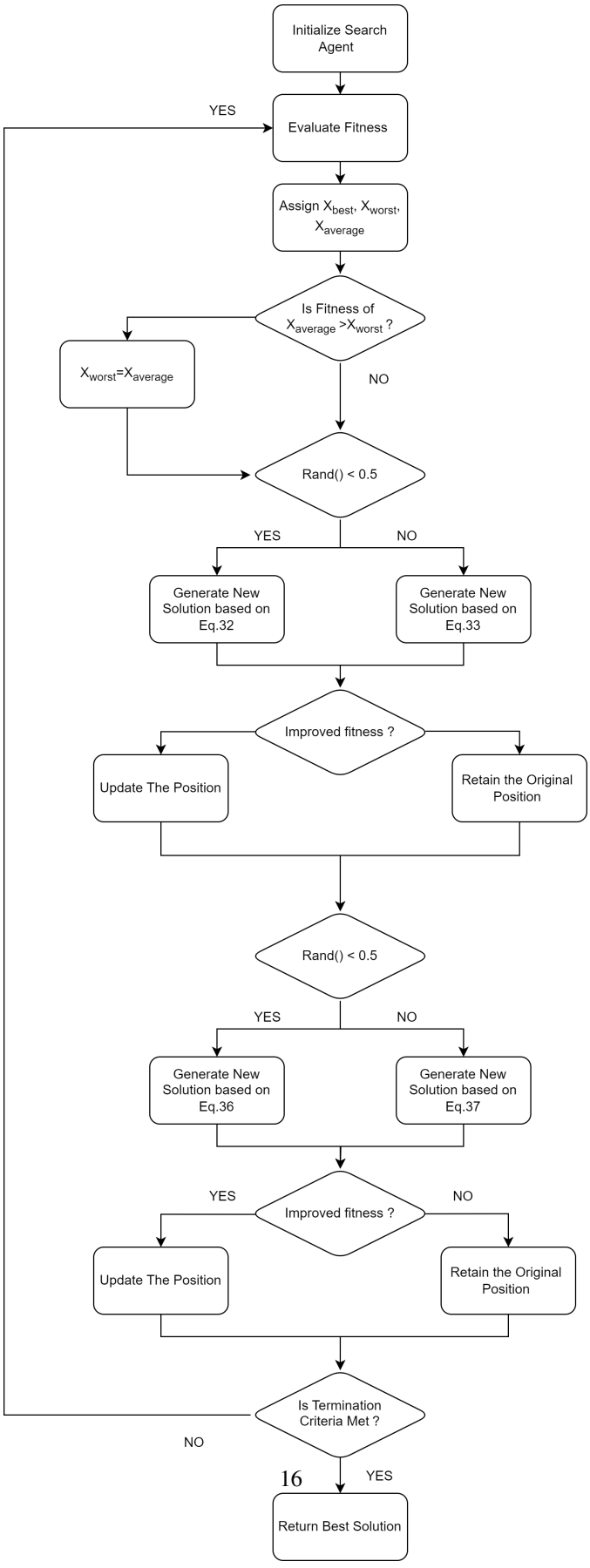


Figure 7: Flow Chart for GROM-SCA

T iteration as $O(Tcn)$. Whereas the position updation has a $O(nd)$ computational cost per iteration where n is $O(Tnd)$ as the total computational cost

$$\begin{aligned} O(SCA) &= O(initialization) + O(evaluation) + O(updation) \\ &= O(nd) + O(T * (nc)) + O(Tnd) \\ &= O(Tn(c + d)) \end{aligned} \tag{40}$$

8.2. Computational Complexity of GROM

8.2.1. Phase 1

The GROM algorithm begins with the initialization of population making it the first step of Phase 1 of the GROM algorithm. This first step can be achieved in *O(nd)* here n is the number of search agents to be initialized and d is the dimension of the problem. This is followed by the evaluation of the search agent this step is repeated till the termination criteria are met. Evaluation of fitness of search agent takes *O(nc)* time for each iteration making the overall computational cost as O(T*(nc)) for T iteration. Using linear time, the best and worst agents may be determined simultaneously. Finding the average position of every agent, however, may be done in constant time. now because of that, the algorithms' Phase 1's overall time complexity is

$$\begin{aligned} O(Phase\ 1\ of\ GROM) &= O(initialization) + O(fitnessevaluation) \\ &= O(nd) + O(T * (nc)) \end{aligned} \tag{41}$$

Where T is the total number of iterations, n is population size and d is dimension size and c is fitness evaluation cost

8.2.2. Phase 2

computation of F_t is crucial to Phase 2 of the GROM algorithm as it determines the step size for the search agent. Since each iteration must evaluate the value of the function F_t and evaluation is feasible in constant time, the evaluation of F_t is achievable in $O(T)$. The updating of the search agent's position comes next. The GROM algorithm generates new positions depending on eq.33. this production has a computational cost i.e. $O(n)$. As a result, the total cost of finding a new position for the search agent is $O(Tn)$ since there are $O(n)$ costs associated with each iteration. Here, n is the population size and T is the total number of iterations. The choice to modify the agent's position to a newly generated position is made using eq.34. Given that a fitness comparison between the new and old positions is necessary, each iteration has a computational cost of $O(cn)$. Consequently, the operation's overall cost is $O(Tcn)$.

$$\begin{aligned} O(Phase\ 2\ of\ GROM) &= O(new\ solution) + O(updation\ of\ solution) \\ &= O(Tn) + O(T * (nc)) \end{aligned} \tag{42}$$

8.2.3. Phase 3

Phase 3 has a similar schema to Phase 2. New positions are generated using the eq. **??** this operation costs $O(n)$ per iteration making the total cost as $O(Tn)$. GROM algorithm in Phase 3 produces a new solution using the eq. **??** which has a total of $O(n)$ cost per iteration hence total cost is $O(Tn)$. whereas MROM employs eq. **??** and eq. **??** both of which employed with equal probality and have a $O(n)$ computational cost per iteration hence total computational cost is $O(Tn)$. Based on equation **??**, the position of the search agent in GROM is updated. This can be done in $O(cn)$ time for each iteration and $O(Tcn)$ time for all iterations. MROM has a similar updation rule hence the computational time taken is $O(Tcn)$

$$\begin{aligned} O(Phase\ 3\ of\ GROM) &= O(new\ solution) + O(updation\ of\ solution) \\ &= O(Tn) + O(T * (nc)) \end{aligned} \tag{43}$$

In conclusion, GROM's time complexity is

$$\begin{aligned} O(GROM) &= O(Phase\ 1) + O(Phase\ 2) + O(Phase\ 3) \\ &= O(nd) + O(Tnc) + O(Tn) + O(Tnc) + O(Tn) + O(Tnc) \\ &= O(nd + Tnc) \end{aligned} \tag{44}$$

8.3. Computational Complexity of GROM-SCA

8.3.1. Phase 1

The proposed algorithm, inspired by GROM, also initiates Phase 1 with population initialization, a crucial initial step. This involves generating search agents for optimization, with a time complexity of $O(nd)$, where n represents the count of search agents and d signifies the problem's dimension. Subsequently, the algorithm proceeds to evaluate the fitness of these search agents iteratively until termination conditions are met. The fitness evaluation step takes $O(nc)$ time per iteration, resulting in an overall computational cost of $O(T \cdot (nc))$ over T iterations. Notably, similar to GROM, our proposed algorithm also enables simultaneous identification of both the best and worst agents in linear time. Calculating the average position of each agent can be achieved in constant time. Therefore, the overall time complexity of Phase 1 in our algorithm is determined as follows:

$$\begin{aligned} O(\text{Phase 1 of Proposed Algorithm}) &= O(\text{Initialization}) + O(\text{Fitness Evaluation}) \\ &= O(nd) + O(T \cdot (nc)) \end{aligned} \tag{45}$$

where T is the total number of iterations, n is the population size, d is the dimension size, and c is the fitness evaluation cost.

8.3.2. Phase 2

In Phase 2 of the proposed algorithm, akin to GROM, the computation of F_t plays a pivotal role as it determines the step size for the search agent. Evaluating F_t is crucial and feasible in constant time, resulting in a time complexity of $O(T)$. Following this, the update of the search agent's position takes place. Just as in GROM, the algorithm generates new positions based on Eq.33, incurring a computational cost of $O(n)$. Thus, the total cost of determining a new position for the search agent amounts to $O(Tn)$, considering the $O(n)$ costs per iteration. The decision to adopt the newly generated position over the old one is made using Eq.34. Given that a fitness comparison between the new and old positions is necessary, each iteration bears a computational cost of $O(cn)$. Consequently, the overall cost of these operations can be expressed as $O(Tcn)$. The summation of these steps yields the overall time complexity of Phase 2 in our algorithm:

$$\begin{aligned} O(\text{Phase 2 of Proposed Algorithm}) &= O(\text{New Solution}) + O(\text{Updation of Solution}) \\ &= O(Tn) + O(T \cdot (nc)) \end{aligned} \tag{46}$$

where T is the total number of iterations, n is the population size, d is the dimension size, and c is the fitness evaluation cost.

8.3.3. Phase 3

In Phase 3 of the GROM algorithm, the update mechanism is mainly composed of two techniques: the Sine-Cosine Update and the Lévy Flight Update. Interestingly, these two techniques are mutually exclusive; if the Sine-Cosine Update is applied, the Lévy Flight Update doesn't happen and vice versa. For the Sine-Cosine Update, the complexity involved in updating each agent's position across all dimensions is captured by $O(n \times d)$. Similarly, for the Lévy Flight Update, the complexity for updating each agent's position over all dimensions can be represented $O(n \times d)$ Considering that only one of these updates occurs in any given iteration, the effective complexity for the position update mechanism within an iteration remains $O(n \times d)$. Another significant operation in Phase 3 is the fitness evaluation of each agent. After a new position is determined, the fitness of this position is compared to the old one. Evaluating the fitness function for the new position involves a computational complexity of $O(c)$ for each agent. Given that there are n agents, the total complexity for the fitness evaluation per iteration becomes $O(cn)$ Aggregating the complexities of the position update mechanism and the fitness evaluation, the total complexity within each iteration is $O(n \times d + cn)$ Finally, since the algorithm is executed for a total of T iterations, the cumulative complexity for Phase 3 is given by:

$$O(\textit{Phase 3 of GROM}) = O(Tn(d + c)) \tag{47}$$

Thus, the overall complexity of Phase 3 of the GROM algorithm is described by $O(Tn(d + c))$.

Table 1 Presents the summary of the above complexity analysis. It is clear from the computational complexity analysis of GROM and SCA and GROM-SCA that the computational cost of both algorithms is similar.

Table 1: Computational Complexity of SCA, GROM and GROM-SCA

Steps of GROM	Complexity	Steps of MROM	Complexity
Phase 1	$N \times D$	Phase 1	$N \times D$
Phase 2	$T \times N + T \times N \times C$	Phase 2	$T \times N + T \times N \times C$
Phase 3	$T \times N + T \times N \times C$	Phase 3	$T \times N + T \times N \times C$

9. Training of Feed-forward Neural Networks Using Hybrid Golden Ratio Optimization Method and Sine Cosine Optimization Algorithm

When employing metaheuristic or evolutionary approaches to train a feed-forward neural network, it is essential to consider both the encoding scheme of potential solutions and the objective function used for optimization. The encoding scheme of the possible solutions refers to how the weights and biases of the neural network are represented. In the case of a feed-forward neural network with a single hidden layer, which is a popular architecture in the literature, the weights and biases can be encoded as a one-dimensional array, as shown in Fig 12. This encoding consists of three components: the weights of the connections between the input and hidden layers, the weights of the connections between the hidden and output layers, and the biases of each neuron.

The choice of objective function is also critical when training a neural network using metaheuristic or evolutionary approaches. The objective function determines how the fitness of each potential solution is evaluated. In this study, the mean squared error (MSE) is utilized as the objective function. MSE is computed as the average squared difference between the predicted and actual output values of the training data, as shown in Eq. 48.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{48}$$

Here, y represents the actual class label, $\hat{y}$ represents the predicted class label, and n represents the number of instances in the training dataset.

The training process of the proposed GROM-SCA algorithm can be summarized in the following steps:

- **Initialization**: The search agents are randomly generated within the range of $[-1, 1]$.
- **Mapping**: The solution vector is decoded to allocate the weights and biases to their respective connections and neurons in the neural network.
- **Evaluation**: The fitness of each search agent is evaluated using the MSE objective function with the training dataset.
- **Update**: The fitness value obtained from the evaluation phase is used to update the search agents using the suggested GROM-SCA method.

These steps are repeated until the termination requirements are met.

10. Result

Meta-heuristic algorithms are inherently probabilistic, and considering the vast number of functions that exist, evaluating them all is not practical. To overcome this challenge, researchers have proposed various benchmark functions to comprehensively assess the efficacy of optimization techniques. In the literature, there are several benchmark functions available such as the Genetic and Evolutionary Computation Conference (GECCO) and the IEEE Congress on Evolutionary Computation (CEC). To evaluate the proposed GROMSCA, two sets of benchmark functions were utilized, which include the 23 well-known benchmark functions test suite and the CEC 14 benchmark suite. These benchmarks were chosen because they are widely used in the literature.

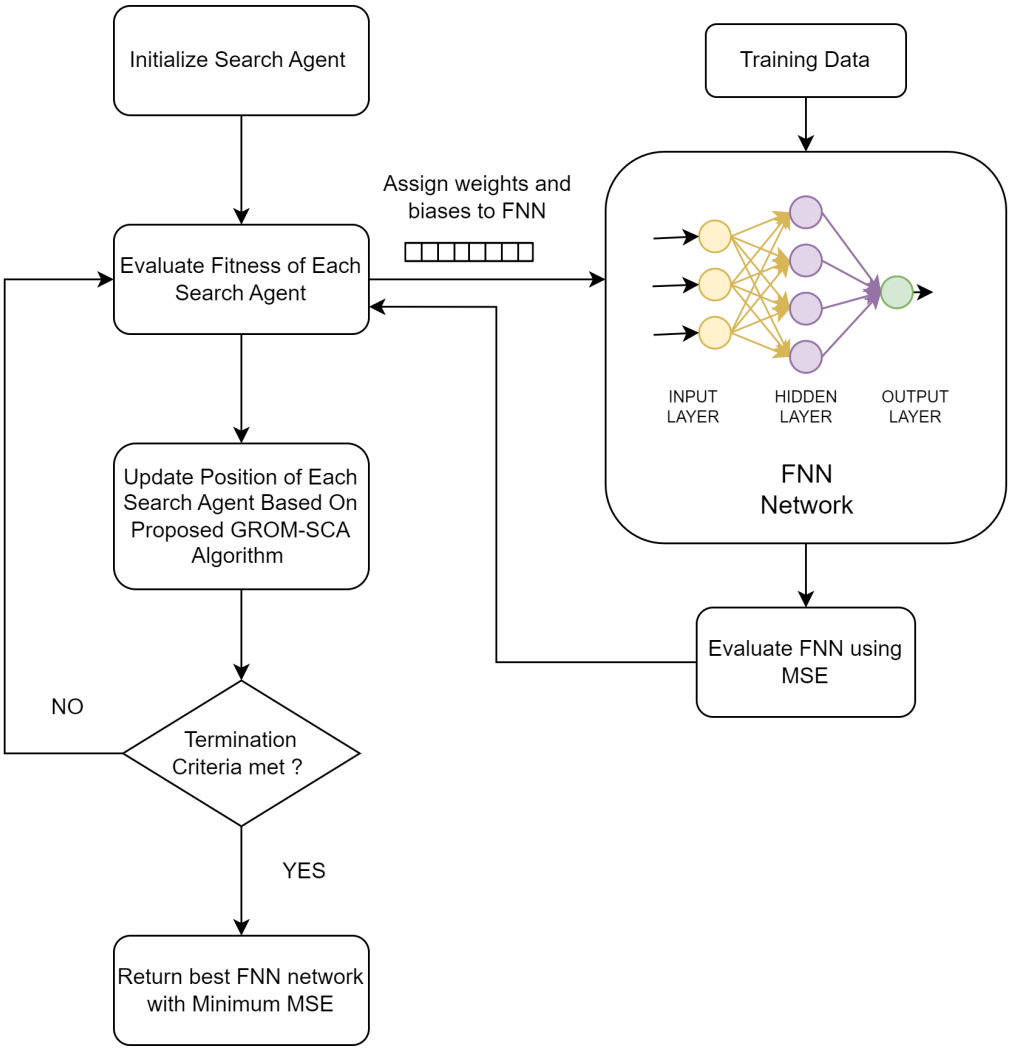


Figure 8: Training FNN Using GROM-SCA

10.1. Comparison on 23 Benchmark Functions

The 23 benchmark suit is a collection of test functions used to evaluate the performance of nature-inspired optimization algorithms. These test functions are designed to simulate various types of optimization problems that are commonly encountered in real-world applications. The 23 benchmark suit can be divided into three categories of test functions based on the characteristics of the optimization problem:

1. Unimodal Functions: These functions have a single global optimum and are relatively easy to optimize. Unimodal functions are commonly used to test the exploitative nature of optimization algorithms.

2. Multimodal Functions with High Dimensionality: These functions have multiple global optima and are more difficult to optimize than unimodal functions. Multimodal functions with high dimensionality are commonly used to test the ability of optimization algorithms to escape from local optima and to find the global optimum.

3. Multimodal Functions with Low Dimensionality: These functions have multiple global optima, but the dimensionality of the problem is low. Multimodal functions with low dimensionality are commonly used to test the robustness and balance between the exploration and exploitation of optimization algorithms.

10.1.1. Results Discussion

Category I: Unimodal Functions In the context of unimodal functions, the GROMSCA algorithm demonstrated remarkable efficiency in optimizing the corresponding test functions (f_1, f_2, f_3, f_4, f_5, f_6, f_7) in the 23 benchmark suite. The algorithm managed to locate the global optima for f_1, f_3, and f_4 in all test runs, which suggests the algorithm's robust optimization capabilities.

It is worth noting that unimodal functions, due to their single optimum nature, are generally considered less complicated optimization problems. Nevertheless, the consistent performance of GROMSCA on these functions provides a robust basis for validating its ability to efficiently exploit the search space.

In terms of comparative performance, GROMSCA outperformed other algorithms in locating the global optimum for f_1, f_2, and f_3. It also demonstrated a commendable approach towards the global optimum for f_4. While the HHO algorithm prevailed in f_5, GROMSCA was competitive with other algorithms. Similar patterns were observed for f_6, and for f_7, GROMSCA reasserted its dominance.

Category II: Multi-modal Functions with High Dimensions The 23 benchmark suite includes a set of multi-modal, high-dimensional functions (f_8 to f_{13}) that pose significant challenges to optimization algorithms due to the presence of numerous local optima. The existence of multiple local optima can potentially misguide the optimization algorithm, resulting in sub-optimal solutions.

Analysis of GROMSCA's performance on these test functions shows variability. For function f_8, GROMSCA struggled to achieve optimal results. However, for functions f_9 and f_{10}, it successfully located the global optima, outperforming the other algorithms. A similar performance was observed for the function f_{11}. For functions f_{12} and f_{13}, the PSO algorithm outperformed GROMSCA, achieving optimal results.

These results underline GROMSCA's strengths and weaknesses in the face of multi-modal, large-dimensional test functions. The algorithm's difficulty with some functions is likely due to the complex landscape of numerous local optima. Nevertheless, GROMSCA's successful location of the global optima on some test functions highlights its potential for effective optimization in real-world scenarios.

Category III: Multi-modal Functions with Low Dimensions The 23 benchmark suite includes a set of low-dimensional multi-modal functions (f_{14} to f_{23}). These functions are characterized by fewer local optima, but still pose considerable challenges for optimization algorithms, testing their resilience and ability to navigate intricate solution spaces.

GROMSCA's performance on these test functions was inconsistent. While the MVO algorithm outperformed GROMSCA in f_{14}, GROMSCA achieved optimal results in f_{15}. For f_{16}, most algorithms showed comparable performance, closely approaching the global optima. The PSO algorithm managed to find the global optima for f_{17}, while for f_{18} and f_{19}, most algorithms performed similarly. GROMSCA emerged as the best performing algorithm for f_{20} and f_{23}, while the GWO algorithm was the most effective for f_{21} and f_{22}.

10.1.2. convergence analysis

The convergence curve illustrated in Figure 11 delineates the convergence trajectories of GROM-SCA when applied to various functions within the 23 Benchmark test suite. A close examination of these convergence curves offers insights into the performance and characteristics of the proposed GROM-SCA Algorithm:

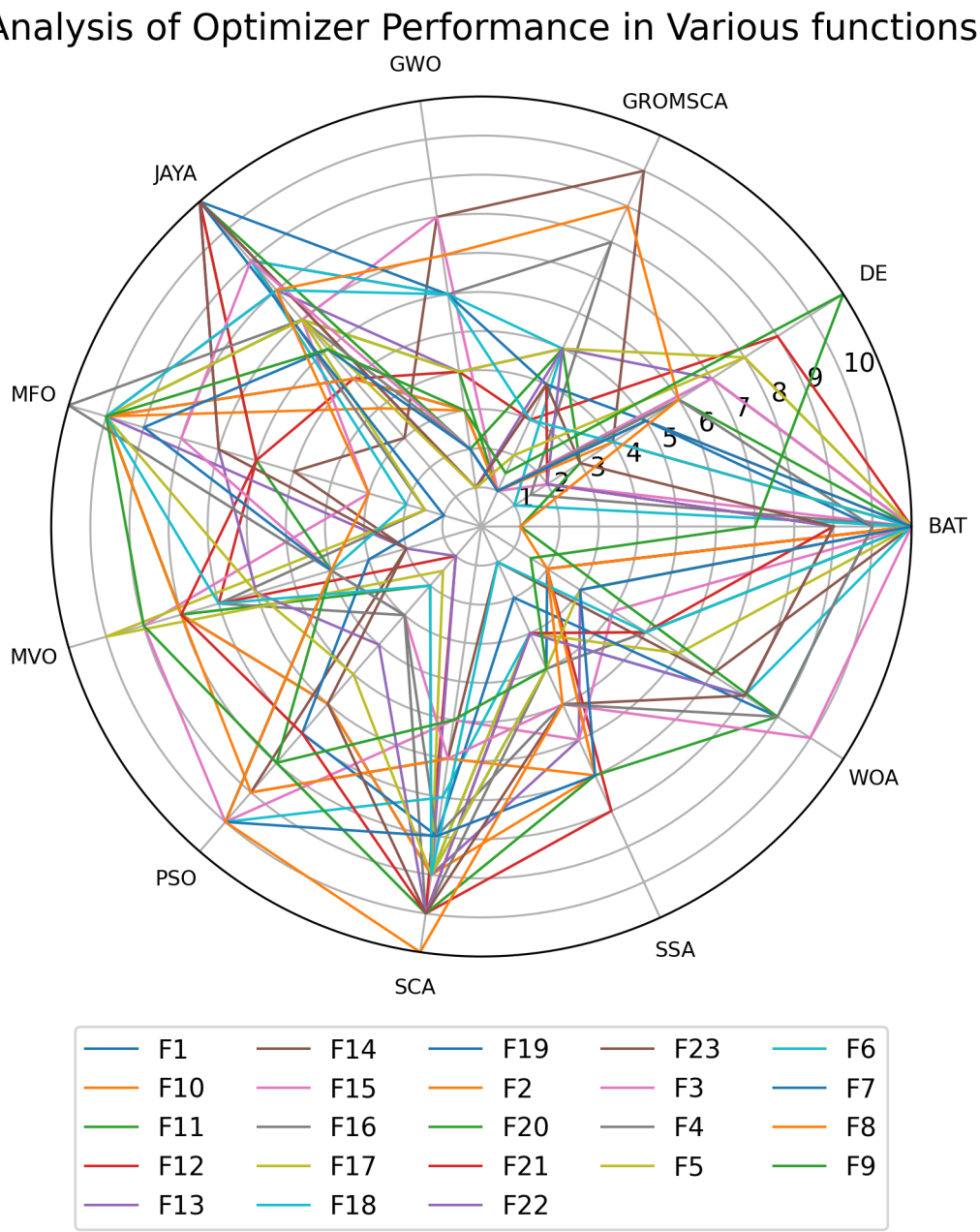


Figure 9: Caption

Table 2: Results on 23 benchmark

Optimizer		BAT	DE	GWO	JAYA	MFO	GROMSCA	MVO	PSO	SCA	SSA	WOA
Function	Metric											
	MEAN	2.591E+04	1.926E+00	1.319E-26	1.369E-01	2.333E+03	0.000E+00	2.099E+00	2.078E-03	1.937E+01	2.705E-08	1.517E-60
F1	BEST	1.452E+04	4.712E-06	7.612E-28	3.822E-06	1.339E-02	0.000E+00	1.229E+00	1.014E-04	1.574E-01	1.225E-08	2.002E-74
	STD	7.742E+03	7.557E+00	2.503E-26	7.483E-01	4.302E+03	0.000E+00	5.829E-01	2.559E-03	2.565E+01	1.026E-08	8.266E-60
	MEAN	1.194E+05	1.004E-03	3.339E-16	1.656E-03	3.301E+01	2.998E-107	8.757E+00	9.773E+00	9.760E-02	6.972E-01	4.109E-46
F2	BEST	3.488E+01	2.803E-04	6.112E-17	8.972E-05	2.910E-02	0.000E+00	6.450E-01	2.483E-02	2.946E-04	4.462E-02	2.417E-54
	STD	6.467E+05	1.956E-03	3.054E-16	6.140E-03	1.764E+01	1.642E-106	2.778E+01	8.512E+00	2.079E-01	7.284E-01	2.132E-45
	MEAN	7.938E+04	1.298E+04	2.522E-03	2.179E+04	1.858E+04	0.000E+00	4.363E+02	1.411E+02	1.283E+04	5.748E+02	5.211E+04
F3	BEST	3.298E+04	5.847E+03	1.494E-05	7.149E+03	2.414E+03	0.000E+00	1.948E+02	6.448E+01	2.177E+03	7.926E+01	3.602E+04
	STD	3.847E+04	3.642E+03	3.239E-03	1.018E+04	1.114E+04	0.000E+00	1.428E+02	6.129E+01	7.126E+03	3.918E+02	1.253E+04
	MEAN	6.240E+01	2.542E+01	5.032E-05	2.744E+01	6.149E+01	0.000E+00	3.281E+00	1.488E+00	4.210E+01	6.208E+00	5.649E+01
F4	BEST	4.399E+01	1.317E+01	3.659E-06	8.226E+00	3.864E+01	0.000E+00	1.385E+00	8.460E-01	8.872E+00	2.077E+00	2.107E+00
	STD	1.109E+01	8.396E+00	5.101E-05	8.800E+00	9.818E+00	0.000E+00	1.632E+00	4.000E-01	1.187E+01	2.822E+00	3.163E+01
	MEAN	5.406E+07	2.851E+03	2.737E+01	1.537E+04	1.553E+04	2.771E+01	4.355E+02	1.152E+02	2.254E+05	1.567E+02	2.822E+01
F5	BEST	8.630E+06	2.747E+01	2.591E+01	2.863E+01	3.566E+01	2.725E+01	4.092E+01	2.486E+01	1.583E+02	2.665E+01	2.755E+01
	STD	4.076E+07	1.225E+04	7.509E-01	4.456E+04	3.394E+04	1.954E-01	7.068E+02	8.151E+01	5.883E+05	4.830E+02	4.338E-01
	MEAN	2.578E+04	5.574E-01	8.680E-01	4.061E+00	1.333E+03	1.157E-01	1.963E+00	2.163E-03	1.082E+02	2.221E-08	9.730E-01
F6	BEST	1.307E+04	4.631E-06	1.026E-04	3.275E+00	2.236E-02	5.667E-02	6.804E-01	1.687E-04	4.211E+00	1.249E-08	4.566E-01
	STD	6.789E+03	1.848E+00	4.276E-01	5.774E-01	3.458E+03	2.419E-02	6.139E-01	3.829E-03	1.887E+02	7.295E-09	3.232E-01
	MEAN	1.362E+01	5.583E-02	3.355E-03	7.302E-02	4.940E+00	5.251E-04	4.721E-02	4.802E+00	1.697E-01	9.489E-02	5.466E-03
F7	BEST	1.675E+00	2.420E-02	6.173E-04	2.032E-02	4.670E-02	6.812E-05	1.716E-02	9.817E-02	1.971E-02	3.035E-02	2.289E-04
	STD	8.309E+00	2.692E-02	1.595E-03	5.486E-02	8.438E+00	3.951E-04	2.043E-02	6.250E+00	2.059E-01	4.092E-02	8.680E-03
	MEAN	-4.231E+03	-6.583E+03	-5.250E+03	-4.696E+03	-8.305E+03	-4.809E+03	-7.539E+03	-4.281E+03	-3.646E+03	-7.560E+03	-9.965E+03
F8	BEST	-6.246E+03	-7.695E+03	-7.410E+03	-7.216E+03	-9.718E+03	-6.605E+03	-9.043E+03	-6.504E+03	-4.249E+03	-8.678E+03	-1.257E+04
	STD	1.261E+03	5.816E+02	1.286E+03	9.623E+02	8.400E+02	9.504E+02	6.977E+02	1.373E+03	2.681E+02	6.134E+02	1.604E+03
	MEAN	1.021E+02	1.714E+02	1.336E+01	1.014E+02	1.543E+02	0.000E+00	1.211E+02	1.133E+02	6.092E+01	4.311E+01	1.782E-01
F9	BEST	5.075E+01	1.452E+02	2.842E-13	4.513E+01	1.025E+02	0.000E+00	6.855E+01	5.263E+01	2.059E-01	1.592E+01	0.000E+00
	STD	4.682E+01	1.498E+01	8.634E+00	3.771E+01	4.430E+01	0.000E+00	3.079E+01	2.830E+01	5.241E+01	1.655E+01	9.760E-01
	MEAN	1.714E+01	4.872E-01	1.686E-13	9.759E-03	1.320E+01	4.441E-16	2.046E+00	7.040E-01	1.268E+01	1.885E+00	4.115E-15
F10	BEST	1.527E+01	3.923E-04	9.992E-14	6.582E-04	3.273E-02	4.441E-16	1.003E+00	6.971E-03	7.297E-02	3.189E-05	4.441E-16
	STD	9.271E-01	6.637E-01	4.476E-14	1.268E-02	8.886E+00	0.000E+00	5.572E-01	6.741E-01	8.987E+00	7.452E-01	2.717E-15
	MEAN	2.271E+02	3.528E-02	6.499E-03	1.427E-01	9.250E+00	0.000E+00	9.555E-01	4.948E-03	1.405E+00	1.089E-02	1.131E-02
F11	BEST	1.150E+02	1.462E-05	0.000E+00	1.258E-04	2.774E-02	0.000E+00	7.869E-01	1.272E-05	3.368E-01	8.557E-06	0.000E+00
	STD	6.129E+01	1.007E-01	9.651E-03	2.322E-01	2.755E+01	0.000E+00	6.709E-02	6.169E-03	1.460E+00	1.051E-02	6.196E-02
	MEAN	6.330E+07	5.661E+04	6.389E-02	1.298E+00	2.198E+00	4.510E-02	2.694E+00	4.741E-05	6.446E+05	4.884E+00	6.530E-02
F12	BEST	8.237E+06	4.959E-04	1.091E-02	4.155E-01	3.129E-02	2.289E-02	5.468E-01	2.585E-06	1.223E+00	1.950E+00	1.187E-02
	STD	5.697E+07	2.913E+05	2.257E-02	7.784E-01	1.258E+00	9.664E-03	1.496E+00	5.265E-05	2.165E+06	2.176E+00	8.669E-02
	MEAN	1.546E+08	1.185E+05	9.255E-01	3.708E+05	1.367E+07	2.193E+00	3.503E-01	1.226E-02	3.878E+05	3.018E+00	8.154E-01
F13	BEST	4.608E+07	1.826E-03	2.074E-01	2.528E+00	1.791E-01	1.290E+00	1.306E-01	1.767E-05	7.017E+00	2.234E-07	2.116E-01
	STD	8.179E+07	5.486E+05	2.799E-01	2.030E+06	7.487E+07	2.502E-01	1.900E-01	2.062E-02	1.439E+06	7.396E+00	3.631E-01
	MEAN	2.536E+01	1.229E+00	3.191E+00	1.235E+00	1.592E+00	4.782E+00	9.980E-01	4.664E+00	2.282E+00	9.980E-01	3.053E+00
F14	BEST	9.980E-01	9.980E-01	9.980E-01	9.980E-01	9.980E-01	9.980E-01	9.980E-01	9.980E-01	9.980E-01	9.980E-01	9.980E-01
	STD	6.993E+01	9.228E-01	3.335E+00	6.191E-01	1.206E+00	4.355E+00	3.794E-11	3.571E+00	2.475E+00	1.487E-16	3.782E+00
	MEAN	1.066E-02	3.287E-03	4.529E-03	2.499E-03	1.559E-03	3.819E-04	2.848E-03	1.103E-02	1.167E-03	8.217E-04	8.374E-04
F15	BEST	8.242E-04	3.075E-04	3.160E-04	5.282E-04	6.644E-04	3.078E-04	4.516E-04	7.003E-04	4.963E-04	3.083E-04	3.124E-04
	STD	1.891E-02	6.822E-03	8.057E-03	2.181E-03	3.562E-03	2.357E-04	5.947E-03	9.896E-03	3.694E-04	3.248E-04	5.468E-04
	MEAN	-9.500E-01	-1.032E+00	-1.032E+00	-1.031E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00
F16	BEST	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00
	STD	2.490E-01	8.247E-17	2.292E-08	3.201E-03	0.000E+00	1.743E-06	4.619E-07	2.220E-16	6.526E-05	2.071E-14	7.817E-09
	MEAN	4.207E-01	4.370E-01	3.979E-01	3.998E-01	3.979E-01	3.979E-01	5.517E-01	3.979E-01	4.023E-01	3.979E-01	3.979E-01
F17	BEST	3.979E-01	3.979E-01	3.979E-01	3.979E-01	3.979E-01	3.979E-01	3.979E-01	3.979E-01	3.981E-01	3.979E-01	3.979E-01
	STD	1.248E-01	1.404E-01	3.319E-06	4.953E-03	0.000E+00	2.136E-05	5.854E-01	0.000E+00	3.687E-03	4.406E-14	5.522E-05
	MEAN	1.920E+01	3.000E+00	3.000E+00	3.410E+00	3.000E+00	3.000E+00	3.000E+00	3.000E+00	3.000E+00	3.000E+00	3.908E+00
F18	BEST	3.000E+00	3.000E+00	3.000E+00	3.000E+00	3.000E+00	3.000E+00	3.000E+00	3.000E+00	3.000E+00	3.000E+00	3.000E+00
	STD	2.801E+01	1.091E-15	7.038E-05	1.132E+00	1.184E-15	3.429E-05	4.661E-06	2.076E-15	3.819E-04	1.426E-13	4.973E+00
	MEAN	-3.811E+00	-3.863E+00	-3.862E+00	-3.635E+00	-3.863E+00	-3.863E+00	-3.863E+00	-3.862E+00	-3.854E+00	-3.863E+00	-3.846E+00
F19	BEST	-3.863E+00	-3.863E+00	-3.863E+00	-3.827E+00	-3.863E+00	-3.863E+00	-3.863E+00	-3.863E+00	-3.859E+00	-3.863E+00	-3.863E+00
	STD	1.961E-01	1.311E-03	2.625E-03	1.520E-01	0.000E+00	3.083E-04	6.977E-06	2.405E-03	2.552E-03	9.319E-14	2.774E-02
	MEAN	-3.270E+00	-3.263E+00	-3.268E+00	-1.561E+00	-3.218E+00	-3.271E+00	-3.265E+00	-3.195E+00	-2.927E+00	-3.209E+00	-3.259E+00
F20	BEST	-3.322E+00	-3.322E+00	-3.322E+00	-2.759E+00	-3.322E+00	-3.322E+00	-3.322E+00	-3.322E+00	-3.111E+00	-3.322E+00	-3.321E+00
	STD	6.014E-02	6.046E-02	8.069E-02	6.013E-01	4.993E-02	6.576E-02	6.148E-02	1.274E-01	1.675E-01	3.882E-02	8.134E-02
	MEAN	-5.132E+00	-8.070E+00	-8.718E+00	-1.106E+00	-6.986E+00	-8.437E+00	-8.036E+00	-7.149E+00	-2.641E+00	-7.642E+00	-8.019E+00
F21	BEST	-1.015E+01	-1.015E+01	-1.015E+01	-2.036E+00	-1.015E+01	-1.013E+01 bn	-1.015E+01	-1.015E+01	-7.374E+00	-1.015E+01	-1.015E+01
	STD	3.216E+00	3.277E+00	2.451E+00	5.207E-01	3.524E+00	2.258E+00	2.672E+00	3.560E+00	2.116E+00	3.223E+00	2.878E+00
	MEAN	-7.225E+00	-9.671E+00	-1.040E+01	-1.630E+00	-8.873E+00	-8.657E+00	-9.031E+00	-8.706E+00	-2.796E+00	-9.042E+00	-7.454E+00
F22	BEST	-1.040E+01	-1.040E+01	-1.040E+01	-4.329E+00	-1.040E+01	-1.039E+01	-1.040E+01	-1.040E+01	-8.300E+00	-1.040E+01	-1.040E+01
	STD	3.602E+00	2.238E+00	1.402E-03	7.972E-01	2.855E+00	2.381E+00	2.834E+00	2.744E+00	2.038E+00	2.546E+00	2.822E+00
	MEAN	-4.261E+00	-9.355E+00	-9.997E+00	-1.701E+00	-6.825E+00	-9.455E+00	-8.854E+00	-8.759E+00	-3.376E+00	-8.798E+00	-5.776E+00
F23	BEST	-1.054E+01	-1.054E+01	-1.054E+01	-4.123E+00	-1.054E+01	-1.053E+01	-1.054E+01	-1.054E+01	-6.458E+00	-1.054E+01	-1.053E+01
	STD	2.950E+00	2.693E+00	2.043E+00	7.641E-01	3.807E+00	1.973E+00	3.154E+00	3.075E+00	1.522E+00	3.006E+00	2.547E+00

- GROM-SCA demonstrates a notably accelerated convergence in comparison to its foundational algorithm across the majority of the benchmark functions. This enhanced convergence rate diminishes the requisite computational time, positioning GROM-SCA as a superior alternative for intricate optimization endeavors.
- The convergence rate of the proffered technique maintains a commendable consistency across a heterogeneous assortment of benchmark functions. This emphasizes the inherent robustness and stability of GROM-SCA, assuring its dependable performance across a broad spectrum of optimization challenges.
- Intriguingly, GROM-SCA appears to excel, especially on benchmark suites characterized by an augmented number of variables and multifaceted functions. Such an observation accentuates the algorithm's innate capacity to adapt to diverse solution terrains and convoluted solution dimensions, thereby enhancing its relevance in sophisticated real-world contexts.

Figure **??** presents a Box plot representation detailing the performance of GROM-SCA across the 23 benchmark functions. This illustrative depiction furnishes a more holistic comprehension of the statistical nuances of the devised method:

- The Box plot associated with GROM-SCA manifests remarkable consistency over diverse benchmark functions, further corroborating the method's inherent robustness and resilience.
- In a comparative assessment with the foundational algorithm, GROM-SCA manifests a superior median performance metric. Such an observation underscores that the incorporation of the GROM-SCA technique has indeed augmented the efficacy of the optimization procedure.
- While GROM-SCA's performance does acknowledge the presence of outliers, their significance is markedly subdued compared to other comparative cases. Such a trend suggests that the advanced method is more insulated from extreme scenarios, thereby ensuring a more consistent and reliable outcome.
- The statistical representation of GROM-SCA's performance, as captured by the Box plot, attests not only to the algorithm's robustness and consistency but also underscores its commendable mean performance coupled with reduced variance. This diminished variability is especially coveted as it curtails the probability of unforeseen results in distinct algorithmic executions.

10.1.3. Analyzing the Effect of Population Size on Algorithm Efficiency

Population size and iteration are two critical parameters that can significantly affect the performance of an optimization algorithm. In this section, an investigation into the effect of population size on the efficiency of the proposed GROMSCA algorithm when applied to the 23 benchmark test functions is presented. By conducting a series of experiments using different population sizes and iteration numbers, the study aims to analyze the impact of these parameters on the optimization process of the test functions and the resulting accuracy of the solutions found. This provides valuable insights into the behavior of the GROMSCA algorithm with respect to iteration and population size on the 23 benchmark test functions.

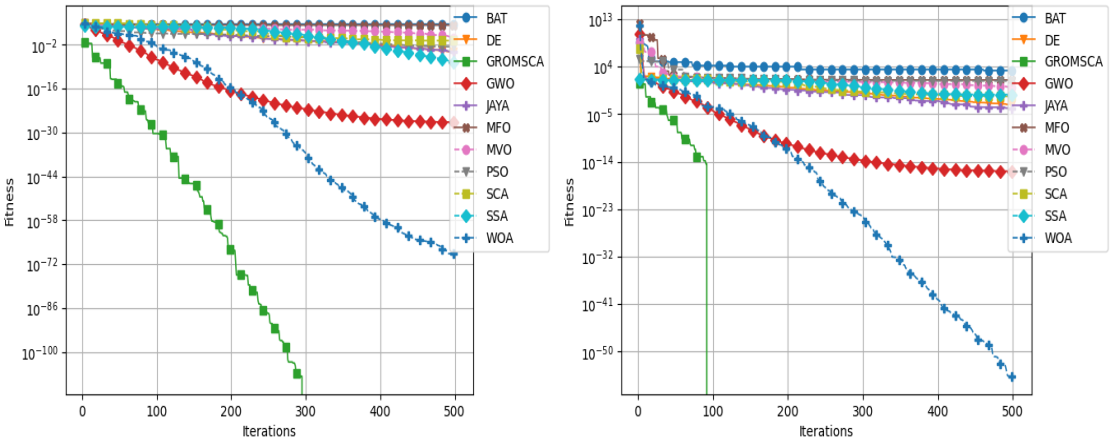
10.1.4. Analyzing the Effect of Population and Iteration on Algorithm Efficiency

10.2. CEC 14 Benchmark suit

The CEC 14 benchmark suit is a widely used set of test functions for evaluating the performance of various nature-inspired optimization algorithms. It comprises of four categories of test functions, each designed to test the optimization capabilities of the algorithms under different scenarios.

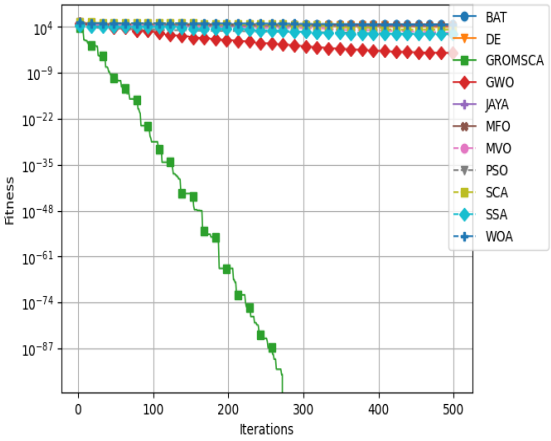
Category A: Unimodal functions, which includes functions f_1 to f_5, is designed to test the optimization capabilities of the algorithms in unimodal scenarios. These functions are relatively simple and have only one global optimum. However, they can be used to assess the efficiency of the algorithms in terms of the number of function evaluations required to find the optimal solution.

Category B: Multi-modal Functions, which includes functions f_6 to f_10, is designed to test the optimization capabilities of the algorithms in multi-modal scenarios. These functions have multiple global optima, which can make

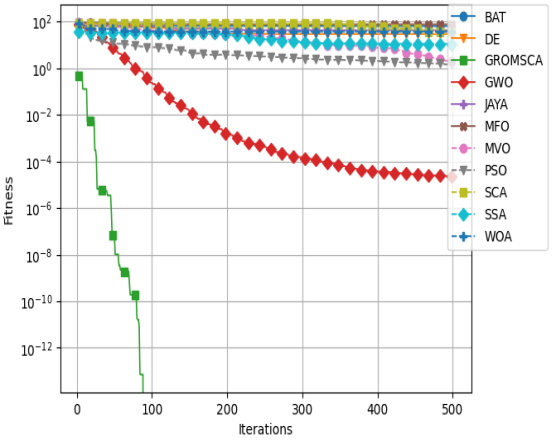


(a) F1

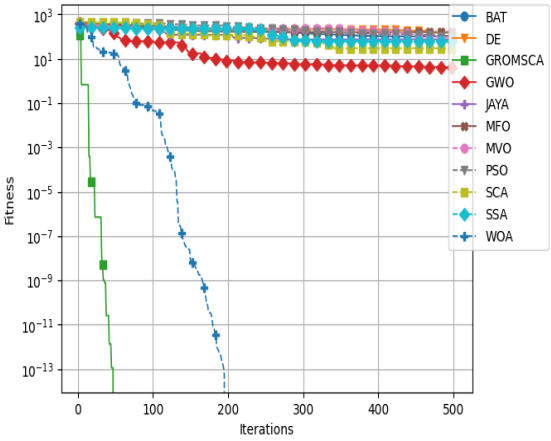
(b) F2



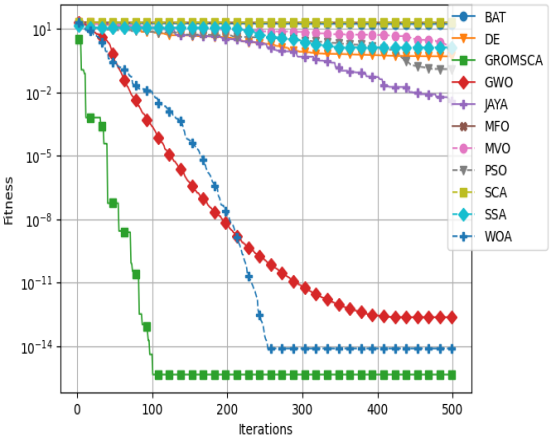
(c) F3



(d) F4

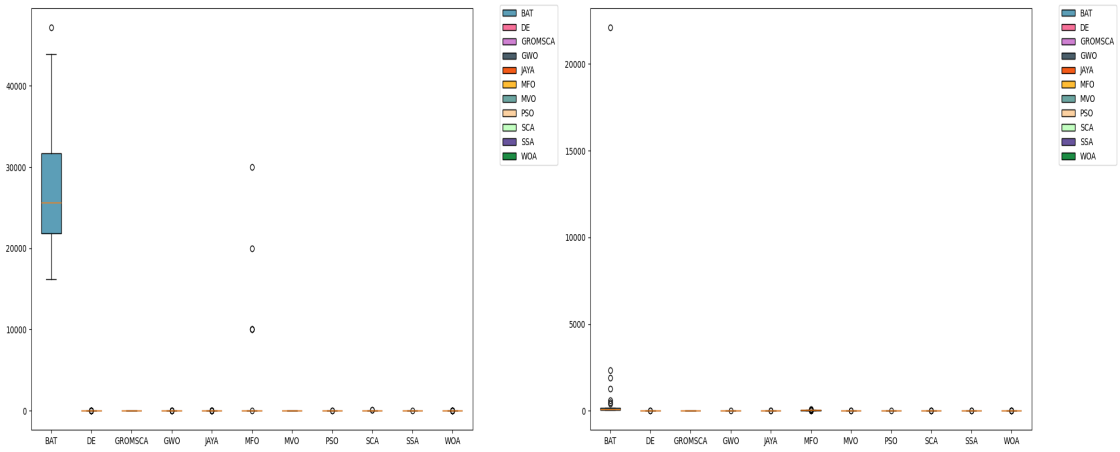


(e) F9

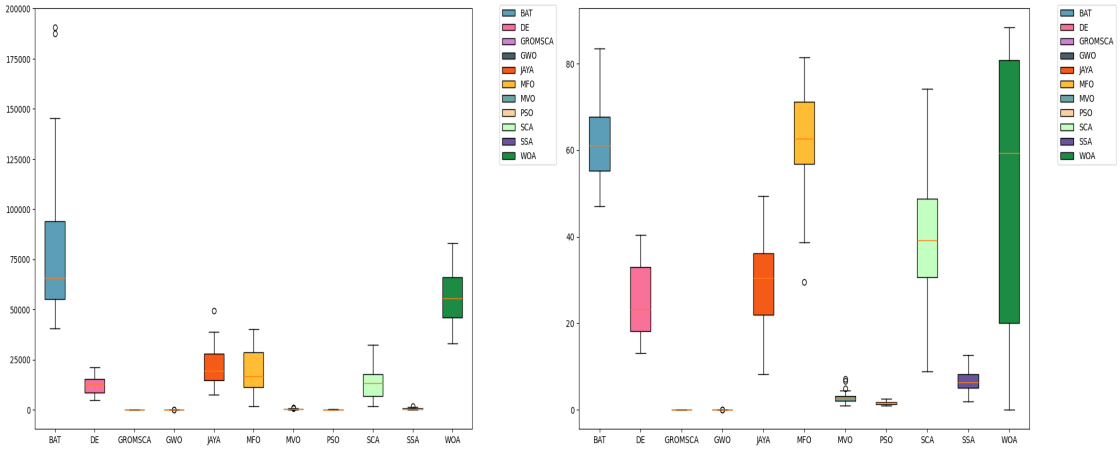


(f) F10

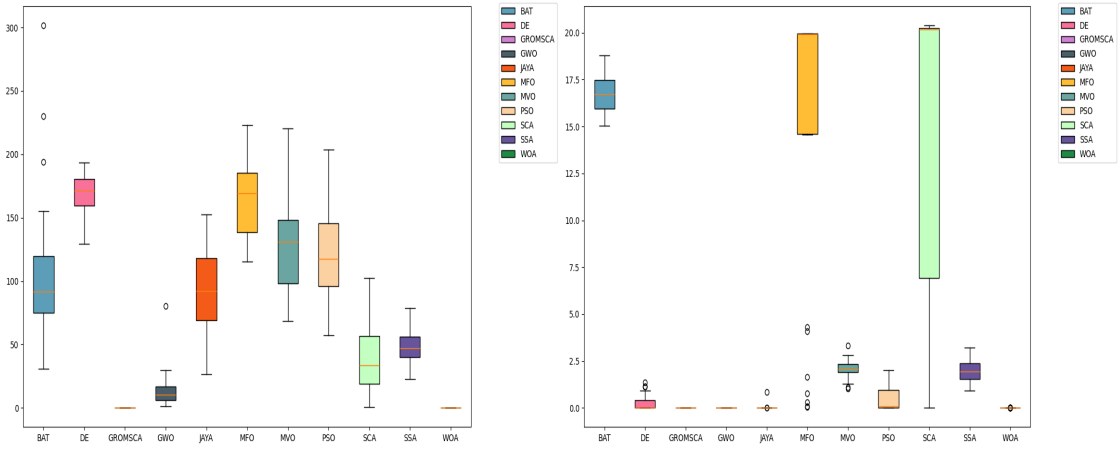
Figure 10: Convergence plot for selected F23 benchmark suit



(a) F1 (b) F2



(c) F3 (d) F4



(e) F9 (f) F10

Figure 11: Convergence plot for selected F23 benchmark suit

it difficult for the algorithms to find the optimal solution. These functions can be used to assess the resilience of the algorithms and their ability to find the global optima in the presence of multiple local optima.

Category C: Hybrid Functions, which includes functions f_{11} to f_{15}, is designed to test the optimization capabilities of the algorithms in hybrid scenarios. These functions are a combination of unimodal and multi-modal functions and can be used to assess the algorithms' ability to adapt to different scenarios.

Category D: Composition Functions, which includes functions f_{16} to f_{20}, is designed to test the optimization capabilities of the algorithms in composition scenarios. These functions are a combination of multiple functions and can be used to assess the algorithms' ability to handle complex and challenging optimization problems.

10.2.1. Result Discussion

10.2.2. Convergence Analysis

11. Feedforward neural network (Multi-layer Perceptron) training using MROM

In this section, we put forward the implementation of the GROMSCA approach as a solution to the issue of training feedforward neural networks (FNNs). In order to provide a comprehensive understanding of the experiment, we will first elaborate on the FNN design and the representation of the search agents employed in the study. Subsequently, we will present the experimental setup, findings, and analysis to establish the efficacy of the proposed method.

11.1. Encoding MROM search agent vectors and design of FNN.

In the process of designing a Feedforward Neural Network (FNN), selecting the number of hidden layers and neurons in each layer is a crucial task as it determines the network's performance. However, there is no established method for selecting these values, leading to the use of different strategies. In this study, a novel approach is proposed where a single hidden layer FNN is utilized, and the number of neurons is calculated using the following equation:

$$m = 2n + 1 \tag{49}$$

where 'm' represents the number of neurons in the hidden layer, and 'n' represents the number of input characteristics. The proposed approach utilizes a search agent vector that consists of three components, namely the biases, the weights connecting the nodes in the input layer to the hidden layer, and the weights connecting the nodes in the hidden layer to the output layer. This search agent vector is constructed by combining 'n' input nodes, 'm' hidden layer nodes, and a single output node, as illustrated in Fig [insert figure reference]. This representation enables an efficient exploration of the solution space during the optimization process.

11.2. Dataset

To assess the effectiveness of the proposed method for training Feed Forward Neural Networks, a set of 14 standard classification datasets was chosen for experimentation. These datasets were obtained from the University of California at Irvine (UCI) Machine Learning repository and the DELVE repository. Table [insert table reference] provides the details of these datasets, including information on the number of classes, features, training, and testing instances.

It is noteworthy that the chosen datasets possess varying characteristics, such as the number of features, training, and testing instances, which presents a significant challenge for the FNN trainers. This diversity in dataset features allows for a more robust evaluation of the proposed method, as it can be tested on a wide range of data with varying complexities. Moreover, the use of widely recognized classification datasets in the literature permits a comparison of the proposed method's performance with other existing techniques.

Moreover, the selected datasets encompass a diverse set of application domains, implying that the proposed method can be applied to a wide range of real-world scenarios. The adoption of standard datasets also simplifies the replication and validation of the proposed method by other researchers in the field.

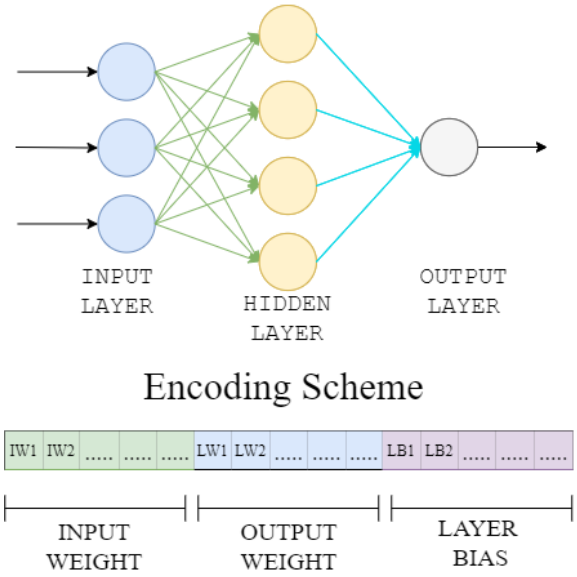


Figure 12: Search agent formation by weights and biases

11.3. Preprocessing

In the experimental setup, all 14 datasets were divided into two parts, with 90% of the data being used for training and 10% for testing. This proportion ensures that a sufficient amount of data is available for training the FNN while still providing an accurate evaluation of its performance on unseen data. Additionally, the datasets were stratified sampled to ensure that the class distribution is preserved, which is important for maintaining the integrity of the results.

To remove the effect of features having different scales, all input features were normalized. Normalization is a preprocessing step that is commonly used in machine learning to ensure that all features are on a similar scale. This is important because features with large values can dominate the optimization process, resulting in a suboptimal solution. The normalization was done by using the following equation:

$$v' = \frac{v_i - min_d}{max_d - min_d} \tag{50}$$

Where v' is the normalized value of the feature, v_i is the original value of the feature, min_d is the minimum value of the feature, and max_d is the maximum value of the feature. This equation transforms the feature values to the interval $[min_d, max_d]$, ensuring that all features are on a similar scale, which improves the performance of the FNN.

11.4. Results and discussions

12. Conclusion and future work

In this study, the recently proposed GROMSCA algorithm was utilized as a FNN trainer for the first time. This work was motivated by the potential of the GROMSCA algorithm as a powerful optimization technique. The problem of training a Feedforward neural network was formulated for the GROMSCA algorithm and the weights and biases of FNNs were optimized using the proposed technique. The results obtained from this approach demonstrated excellent classification accuracy on eight datasets with various characteristics.

The findings of this study revealed that the proposed strategy is more effective at training FNNs than other training techniques, such as BAT, CS, PSO, GA, MFO, JAYA, HHO, which have been employed in the literature. This is due to the ability of the GROMSCA algorithm to effectively search the solution space and converge to a global optimum.

Dataset Name	No. of Classes	No. of Features	No. of Training Sample	No. of Testing sample
Aggregation	6	2	709	79
Aniso	3	2	1350	150
appendicitis	2	7	96	10
Balance	3	4	563	62
Banknote	2	4	1235	137
Blobs	3	2	1350	150
Blood	2	4	673	75
Circles	2	2	1350	150
Diagmosis_II	2	6	108	12
Ecoli	5	7	302	34
Flame	2	2	216	24
Glass	4	9	193	21
Heart	2	13	243	27
Ionosphere	2	34	316	35
Iris	3	4	135	15
Iris2D	3	2	135	15
Jain	2	2	336	37
Liver	3	6	310	35
Moons	2	2	1350	150
Mouse	3	2	441	49
Path-based	3	2	270	30
Seeds	2	7	189	21
Smiley	4	2	450	50
Sonar	2	60	187	21
Varied	3	2	1350	150
Vary-Density	3	2	135	15
Vertebral2	2	6	279	31
Vertebral 3	3	6	279	31
Wdbc	2	30	512	57
Wine	3	13	160	18

Data-Sets		MROM	GROM	BAT	CS	DE	FFA	GA	HHO	JAYA	MFO	PSO	SCA	SSA	WOA
	avg	1.20E-02	1.20E-02	1.20E-02	1.40E-02	1.20E-02	2.10E-02	1.77E-01	1.20E-02	1.20E-02	1.20E-02	1.20E-02	1.20E-02	1.20E-02	1.20E-02
Aggregation	std	3.53E-18	3.53E-18	3.53E-18	1.10E-02	3.53E-18	3.05E-02	2.82E-17	3.53E-18	3.53E-18	3.53E-18	3.53E-18	3.53E-18	3.53E-18	3.53E-18
	best	1.20E-02	1.20E-02	1.20E-02	7.50E-02	1.20E-02	1.39E-01	1.77E-01	1.20E-02	1.20E-02	1.20E-02	1.20E-02	1.20E-02	1.20E-02	1.20E-02
	avg	7.07E-01	7.04E-01	6.74E-01	7.05E-01	7.06E-01	7.07E-01	3.53E-01	7.07E-01	7.05E-01	7.07E-01	7.07E-01	7.05E-01	7.07E-01	7.07E-01
Aniso	std	1.13E-16	3.71E-03	8.50E-02	3.23E-03	2.03E-03	1.13E-16	5.65E-17	1.13E-16	2.53E-03	1.13E-16	1.13E-16	2.71E-03	1.13E-16	1.13E-16
	best	7.07E-01	7.07E-01	7.07E-01	7.07E-01	7.07E-01	7.07E-01	3.53E-01	7.07E-01	7.07E-01	7.07E-01	7.07E-01	7.07E-01	7.07E-01	7.07E-01
	avg	8.21E-01	8.18E-01	8.12E-01	8.03E-01	7.99E-01	8.03E-01	7.27E-01	8.06E-01	7.84E-01	5.19E-01	7.84E-01	7.99E-01	8.18E-01	8.21E-01
Appenditics	std	1.60E-02	3.39E-16	2.30E-02	4.80E-02	5.00E-02	3.40E-02	2.26E-16	3.10E-02	6.50E-02	1.20E-02	5.50E-02	5.50E-02	3.39E-16	3.70E-02
	best	9.09E-01	8.18E-01	8.18E-01	9.09E-01	9.09E-01	8.18E-01	7.27E-01	8.18E-01	9.09E-01	5.39E-01	9.09E-01	9.09E-01	8.18E-01	9.09E-01
	avg	4.62E-01	4.41E-01	4.97E-01	4.89E-01	4.77E-01	5.15E-01	4.60E-01	4.60E-01	4.77E-01	4.34E-01	5.15E-01	4.34E-01	5.06E-01	4.33E-01
Balance	std	5.80E-02	4.80E-02	2.40E-02	2.40E-02	4.10E-02	1.70E-02	5.65E-17	4.50E-02	3.70E-02	6.30E-02	2.00E-02	6.00E-02	1.90E-02	5.00E-02
	best	5.23E-01	5.07E-01	5.23E-01	5.23E-01	5.23E-01	5.39E-01	4.60E-01	5.39E-01	5.23E-01	5.23E-01	5.23E-01	5.23E-01	5.23E-01	5.23E-01
	avg	9.92E-01	9.20E-01	9.83E-01	9.61E-01	9.44E-01	9.90E-01	5.57E-01	9.79E-01	9.52E-01	9.88E-01	9.84E-01	9.43E-01	9.11E-01	9.64E-01
Banknote	std	7.00E-03	4.70E-02	1.90E-02	2.20E-02	3.20E-02	9.00E-03	0.00E+00	2.60E-02	2.80E-02	6.00E-03	7.00E-03	3.30E-02	5.80E-02	2.80E-02
	best	1.00E+00	9.71E-01	1.00E+00	1.00E+00	1.00E+00	1.00E+00	5.57E-01	1.00E+00	9.92E-01	1.00E+00	9.92E-01	9.92E-01	9.92E-01	9.92E-01
	avg	6.52E-01	6.53E-01	6.53E-01	6.53E-01	6.53E-01	8.33E-01	3.20E-01	6.53E-01	6.52E-01	6.53E-01	6.53E-01	6.53E-01	6.53E-01	6.52E-01
Blobs	std	2.00E-03	2.26E-16	1.00E-03	2.26E-16	2.26E-16	2.26E-16	5.65E-17	2.26E-16	3.00E-03	2.26E-16	2.26E-16	2.26E-16	2.26E-16	3.00E-03
	best	6.53E-01	6.53E-01	6.53E-01	6.53E-01	6.53E-01	8.33E-01	3.20E-01	6.53E-01	6.53E-01	6.53E-01	6.53E-01	6.53E-01	6.53E-01	6.53E-01
	avg	7.96E-01	7.97E-01	8.01E-01	8.03E-01	7.94E-01	6.53E-01	8.00E-01	8.00E-01	7.97E-01	8.03E-01	7.99E-01	7.96E-01	8.00E-01	7.99E-01
Blood	std	1.10E-02	9.00E-03	7.00E-03	1.00E-02	2.10E-02	2.26E-16	2.26E-16	1.00E-02	7.00E-03	1.30E-02	1.10E-02	1.70E-02	1.00E-02	6.00E-03
	best	8.13E-01	8.13E-01	8.40E-01	8.40E-01	8.26E-01	6.53E-01	8.00E-01	8.13E-01	8.13E-01	8.40E-01	8.40E-01	8.40E-01	8.40E-01	8.13E-01
	avg	9.98E-01	8.97E-01	8.55E-01	8.81E-01	8.80E-01	7.97E-01	8.00E-01	8.84E-01	9.26E-01	9.97E-01	1.00E+00	8.43E-01	9.86E-01	8.76E-01
Circles	std	7.00E-03	8.90E-02	6.00E-02	7.30E-02	7.10E-02	1.30E-02	2.26E-16	8.40E-02	6.40E-02	1.00E-02	0.00E+00	8.00E-02	3.70E-02	8.60E-02
	best	1.00E+00	1.00E+00	9.60E-01	1.00E+00	1.00E+00	8.26E-01	8.00E-01	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00
	avg	8.33E-01	8.33E-01	8.33E-01	8.33E-01	8.33E-01	8.33E-01	0.00E+00	8.33E-01	8.33E-01	8.33E-01	8.33E-01	8.33E-01	8.33E-01	8.33E-01
Diagnosis_II	std	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16	0.00E+00	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16
	best	8.33E-01	8.33E-01	8.33E-01	8.33E-01	8.33E-01	8.33E-01	0.00E+00	8.33E-01	8.33E-01	8.33E-01	8.33E-01	8.33E-01	8.33E-01	8.33E-01
	avg	1.23E-01	1.40E-01	1.07E-01	2.90E-01	2.74E-01	3.18E-01	5.58E-01	8.20E-02	2.48E-01	3.19E-01	2.89E-01	1.78E-01	2.94E-01	7.00E-02
Ecoli	std	9.90E-02	1.13E-01	1.45E-01	3.50E-02	3.00E-02	1.50E-02	0.00E+00	1.07E-01	7.10E-02	1.00E-02	8.90E-02	8.70E-02	7.00E-03	9.00E-02
	best	2.94E-01	2.94E-01	4.11E-01	4.11E-01	3.23E-01	3.52E-01	5.58E-01	2.64E-01	3.82E-01	3.23E-01	3.23E-01	2.94E-01	3.23E-01	2.64E-01
	avg	9.19E-01	8.15E-01	8.04E-01	8.51E-01	7.86E-01	8.56E-01	7.08E-01	7.98E-01	8.22E-01	8.16E-01	8.51E-01	8.00E-01	8.01E-01	7.48E-01
Flame	std	6.80E-02	1.14E-01	1.06E-01	7.70E-02	9.60E-02	1.15E-01	2.26E-16	1.04E-01	1.02E-01	8.90E-02	9.20E-02	1.15E-01	1.18E-01	1.06E-01
	best	1.00E+00	1.00E+00	1.00E+00	9.58E-01	9.16E-01	1.00E+00	7.08E-01	9.58E-01	1.00E+00	9.58E-01	1.00E+00	1.00E+00	1.00E+00	1.00E+00
	avg	9.00E-02	9.00E-02	9.00E-02	9.00E-02	9.00E-02	2.72E-01	9.00E-02	9.00E-02	9.00E-02	9.00E-02	9.00E-02	9.00E-02	9.00E-02	9.00E-02
Glass	std	1.41E-17	1.41E-17	1.41E-17	1.41E-17	1.41E-17	5.65E-17	1.41E-17	1.41E-17	1.41E-17	1.41E-17	1.41E-17	1.41E-17	1.41E-17	1.41E-17
	best	9.00E-02	9.00E-02	9.00E-02	9.00E-02	9.00E-02	2.72E-01	9.00E-02	9.00E-02	9.00E-02	9.00E-02	9.00E-02	9.00E-02	9.00E-02	9.00E-02
	avg	6.86E-01	6.95E-01	6.62E-01	6.95E-01	6.62E-01	6.97E-01	4.07E-01	7.13E-01	6.64E-01	6.87E-01	6.92E-01	6.80E-01	7.08E-01	6.48E-01
Heart	std	3.40E-02	3.70E-02	6.40E-02	,0.052	5.20E-02	5.20E-02	1.69E-16	6.80E-02	5.90E-02	4.80E-02	4.30E-02	5.30E-02	7.10E-02	8.00E-02
	best	7.40E-01	7.77E-01	8.14E-01	7.77E-01	8.14E-01	8.14E-01	4.07E-01	8.51E-01	7.77E-01	7.40E-01	8.14E-01	7.77E-01	8.88E-01	8.14E-01
	avg	8.11E-01	7.18E-01	7.39E-01	6.03E-01	5.62E-01	6.70E-01	5.83E-01	7.99E-01	5.95E-01	6.41E-01	6.50E-01	6.02E-01	7.12E-01	6.04E-01
Ionosphere	std	5.50E-02	7.40E-02	8.50E-02	7.60E-02	7.40E-02	8.70E-02	0.00E+00	2.90E-02	8.60E-02	7.90E-02	1.03E-01	8.10E-02	6.00E-02	1.00E-01
	best	9.16E-01	6.53E-01	8.88E-01	8.05E-01	6.94E-01	8.61E-01	5.83E-01	8.33E-01	8.33E-01	8.33E-01	8.61E-01	7.50E-01	8.33E-01	7.50E-01
	avg	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	4.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01
Iris	std	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16	1.13E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16
	best	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	4.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01

Data-Sets		MROM	GROM	BAT	CS	DE	FFA	GA	HHO	JAYA	MFO	PSO	SCA	SSA	WOA
	avg	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	4.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01
Iris2D	std	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16	1.13E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16
	best	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	4.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01
	avg	9.78E-01	9.39E-01	9.63E-01	9.47E-01	9.42E-01	9.57E-01	3.42E-01	9.50E-01	9.40E-01	9.43E-01	9.67E-01	9.37E-01	9.57E-01	9.45E-01
Jain	std	2.10E-02	2.40E-02	2.80E-02	3.40E-02	3.50E-02	2.50E-02	1.69E-16	2.70E-02	3.40E-02	2.00E-02	1.90E-02	3.80E-02	1.70E-02	2.50E-02
	best	1.00E+00	9.73E-01	1.00E+00	1.00E+00	1.00E+00	1.00E+00	3.42E-01	1.00E+00	1.00E+00	9.73E-01	1.00E+00	1.00E+00	9.73E-01	9.73E-01
	avg	3.71E-01	3.71E-01	3.70E-01	3.71E-01	3.71E-01	3.71E-01	0.00E+00	3.71E-01	3.71E-01	3.71E-01	3.71E-01	3.71E-01	3.71E-01	3.71E-01
Liver	std	1.13E-16	1.13E-16	5.22E-03	1.13E-16	1.13E-16	1.13E-16	0.00E+00	1.13E-16	1.13E-16	1.13E-16	1.13E-16	1.13E-16	1.13E-16	1.13E-16
	best	3.71E-01	3.71E-01	3.71E-01	3.71E-01	3.71E-01	3.71E-01	0.00E+00	3.71E-01	3.71E-01	3.71E-01	3.71E-01	3.71E-01	3.71E-01	3.71E-01
	avg	9.48E-01	9.27E-01	9.24E-01	9.42E-01	9.25E-01	9.53E-01	4.39E-01	9.29E-01	9.45E-01	9.61E-01	9.68E-01	9.22E-01	9.51E-01	9.27E-01
Moons	std	2.50E-02	1.70E-02	2.50E-02	1.70E-02	1.80E-02	2.30E-02	1.29E-17	1.50E-02	1.70E-02	1.30E-02	1.20E-02	1.50E-02	1.30E-02	1.70E-02
	best	9.93E-01	9.66E-01	9.66E-01	9.73E-01	9.73E-01	9.93E-01	4.40E-01	9.66E-01	9.86E-01	9.86E-01	9.86E-01	9.60E-01	9.80E-01	9.66E-01
	avg	6.12E-01	6.05E-01	6.05E-01	6.03E-01	6.05E-01	6.32E-01	5.91E-01	6.02E-01	6.06E-01	6.03E-01	6.12E-01	6.05E-01	5.96E-01	5.94E-01
Mouse	std	3.10E-02	1.40E-02	1.40E-02	2.70E-02	3.10E-02	1.40E-02	1.13E-16	2.10E-02	2.10E-02	1.40E-02	1.10E-02	3.90E-02	1.00E-02	2.80E-02
	best	6.93E-01	6.32E-01	6.32E-01	6.53E-01	6.32E-01	6.53E-01	5.91E-01	6.32E-01	6.53E-01	6.32E-01	6.32E-01	6.93E-01	6.32E-01	6.12E-01
	avg	4.32E-01	4.33E-01	4.28E-01	4.31E-01	4.40E-01	4.25E-01	2.66E-01	4.26E-01	4.30E-01	4.34E-01	4.34E-01	4.26E-01	4.30E-01	4.28E-01
Pathbased	std	1.00E-02	3.40E-02	2.00E-02	1.20E-02	2.50E-02	1.40E-02	0.00E+00	2.50E-02	2.00E-02	6.00E-03	6.00E-03	2.60E-02	1.00E-02	1.44E-01
	best	4.66E-01	4.66E-01	4.66E-01	4.66E-01	5.00E-01	4.33E-01	2.66E-01	4.33E-01	4.66E-01	4.66E-01	4.66E-01	4.66E-01	4.33E-01	4.33E-01
	avg	5.96E-01	5.80E-01	5.98E-01	5.85E-01	5.82E-01	5.84E-01	4.28E-01	5.87E-01	5.80E-01	5.85E-01	5.84E-01	5.98E-01	5.82E-01	5.85E-01
Seeds	std	3.20E-02	3.80E-02	2.70E-02	3.10E-02	4.40E-02	2.10E-02	1.13E-16	3.10E-02	3.60E-02	2.20E-02	2.40E-02	3.20E-02	2.40E-02	4.10E-02
	best	6.19E-01	6.19E-01	6.66E-01	6.19E-01	6.66E-01	6.19E-01	4.28E-01	6.19E-01	6.19E-01	6.19E-01	6.19E-01	6.19E-01	6.19E-01	6.19E-01
	avg	5.40E-01	5.60E-01	6.23E-01	5.14E-01	5.58E-01	5.76E-01	6.40E-01	5.88E-01	5.33E-01	5.39E-01	5.38E-01	5.35E-01	5.44E-01	6.07E-01
Smiley	std	5.10E-02	7.30E-02	4.60E-02	6.80E-02	1.00E-01	7.50E-02	1.13E-16	6.60E-02	9.60E-02	5.00E-02	6.20E-02	1.05E-01	5.50E-02	7.70E-02
	best	6.40E-01	6.40E-01	6.40E-01	7.40E-01	6.40E-01	7.80E-01	6.40E-01	6.60E-01	6.40E-01	6.40E-01	6.40E-01	6.40E-01	6.40E-01	7.80E-01
	avg	7.66E-01	7.31E-01	7.84E-01	7.26E-01	7.04E-01	7.34E-01	3.80E-01	8.58E-01	7.04E-01	7.55E-01	7.31E-01	7.04E-01	8.01E-01	6.57E-01
Sonar	std	6.70E-02	1.00E-01	7.90E-02	6.60E-02	7.00E-02	7.90E-02	1.13E-16	5.20E-02	5.50E-02	6.80E-02	7.00E-02	9.10E-02	8.50E-02	7.90E-02
	best	8.57E-01	9.04E-01	9.04E-01	8.57E-01	8.57E-01	9.04E-01	3.80E-01	9.52E-01	8.57E-01	8.57E-01	8.57E-01	9.52E-01	9.52E-01	8.57E-01
	avg	6.90E-01	6.54E-01	6.59E-01	6.80E-01	6.70E-01	6.85E-01	3.53E-01	6.66E-01	6.67E-01	6.62E-01	6.87E-01	6.70E-01	6.89E-01	6.54E-01
Varied	std	7.00E-03	2.70E-02	5.80E-02	1.30E-02	1.80E-02	1.40E-02	5.65E-17	2.30E-02	1.70E-02	1.90E-02	1.10E-02	1.80E-02	8.00E-03	2.60E-02
	best	7.00E-01	7.00E-01	6.93E-01	7.00E-01	6.93E-01	7.00E-01	3.53E-01	7.00E-01	7.00E-01	7.00E-01	7.00E-01	6.93E-01	7.00E-01	7.00E-01
	avg	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	4.00E-01	8.00E-01	7.95E-01	8.00E-01	8.00E-01	7.64E-01	8.00E-01	7.97E-01
Vary-Density	std	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16	2.26E-16	1.60E-02	2.26E-16	2.26E-16	3.30E-02	2.26E-16	1.20E-02
	best	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	4.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01	8.00E-01
	avg	8.06E-01	7.90E-01	7.84E-01	8.04E-01	7.55E-01	7.99E-01	7.41E-01	7.94E-01	7.69E-01	7.73E-01	7.95E-01	7.65E-01	8.08E-01	7.54E-01
Vertebral2	std	3.20E-02	6.40E-02	3.90E-02	4.70E-02	5.20E-02	4.50E-02	0.00E+00	4.50E-02	4.90E-02	6.60E-02	4.60E-02	4.40E-02	4.00E-02	5.80E-02
	best	9.03E-01	8.70E-01	8.70E-01	9.03E-01	8.38E-01	9.03E-01	7.41E-01	8.70E-01	8.38E-01	8.70E-01	8.38E-01	8.70E-01	8.70E-01	8.70E-01
	avg	2.59E-01	2.51E-01	2.53E-01	2.56E-01	2.48E-01	2.61E-01	1.93E-01	2.48E-01	2.51E-01	2.53E-01	2.56E-01	2.52E-01	2.54E-01	2.52E-01
Vertebral3	std	1.50E-02	3.80E-02	2.90E-02	3.10E-02	3.50E-02	4.30E-02	5.65E-17	3.00E-02	3.30E-02	2.50E-02	2.70E-02	3.10E-02	3.00E-02	3.20E-02
	best	2.90E-01	3.54E-01	3.22E-01	3.22E-01	3.22E-01	3.54E-01	1.93E-01	2.90E-01	3.22E-01	2.90E-01	3.22E-01	3.22E-01	2.90E-01	2.90E-01
	avg	9.36E-01	9.33E-01	9.17E-01	9.36E-01	9.23E-01	9.44E-01	2.98E-01	9.41E-01	9.28E-01	9.32E-01	9.33E-01	9.21E-01	9.33E-01	9.14E-01
Wdbc	std	1.50E-02	2.50E-02	2.60E-02	2.40E-02	2.50E-02	2.60E-02	1.13E-16	1.90E-02	1.40E-02	2.30E-02	4.30E-02	2.50E-02	1.50E-02	4.30E-02
	best	9.82E-01	9.82E-01	9.64E-01	9.82E-01	9.64E-01	9.82E-01	2.98E-01	9.82E-01	9.64E-01	9.82E-01	9.64E-01	9.64E-01	9.64E-01	9.82E-01
	avg	4.03E-01	3.92E-01	3.96E-01	4.05E-01	4.01E-01	4.29E-01	2.22E-01	3.92E-01	3.96E-01	4.53E-01	4.37E-01	4.62E-01	4.11E-01	3.96E-01
Wine	std	3.50E-02	2.00E-02	2.40E-02	3.90E-02	3.70E-02	5.60E-02	2.82E-17	1.40E-02	3.10E-02	5.60E-02	5.20E-02	8.60E-02	4.20E-02	1.90E-02
	best	5.00E-01	5.00E-01	4.44E-01	5.00E-01	5.55E-01	5.55E-01	2.22E-01	4.44E-01	5.00E-01	6.11E-01	5.55E-01	6.11E-01	5.55E-01	4.44E-01

Additionally, the GROMSCA algorithm is robust against the problem of premature convergence and is able to achieve a high accuracy even in the presence of noisy data.

In conclusion, this study has demonstrated the effectiveness of the GROMSCA algorithm as a FNN trainer and has provided a novel approach for training FNNs. In future work, it would be beneficial to explore the use of the GROMSCA algorithm in other types of neural networks such as Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs). Additionally, the GROMSCA algorithm can be extended to other optimization problems in machine learning, such as feature selection and parameter tuning.

References

[1] J. K. Basu, D. Bhattacharyya, T.-h. Kim, Use of artificial neural network in pattern recognition, International journal of software engineering and its applications 4 (2).

[2] D. W. Otter, J. R. Medina, J. K. Kalita, A survey of the usages of deep learning for natural language processing, IEEE transactions on neural networks and learning systems 32 (2) (2020) 604–624.

[3] B. C. Kamble, Speech recognition using artificial neural network–a review, Int. J. Comput. Commun. Instrum. Eng 3 (1) (2016) 61–64.

[4] A. Khemphila, V. Boonjing, Heart disease classification using neural network and feature selection, in: 2011 21st International Conference on Systems Engineering, IEEE, 2011, pp. 406–409.

[5] J. George, A. Cyril, B. I. Koshy, L. Mary, Exploring sound signature for vehicle detection and classification using ann, International Journal on Soft Computing 4 (2) (2013) 29.

[6] S. Kaymak, A. Helwan, D. Uzun, Breast cancer image classification using artificial neural networks, Procedia computer science 120 (2017) 126–131.

[7] J. S. Almeida, Predictive non-linear modeling of complex data by artificial neural networks, Current opinion in biotechnology 13 (1) (2002) 72–76.

[8] M. Ghiassi, S. Nangoy, A dynamic artificial neural network model for forecasting nonlinear processes, Computers & Industrial Engineering 57 (1) (2009) 287–297.

[9] A. Tealab, H. Hefny, A. Badr, Forecasting of nonlinear time series using ann, Future Computing and Informatics Journal 2 (1) (2017) 39–47.

[10] D. Svozil, V. Kvasnicka, J. Pospichal, Introduction to multi-layer feed-forward neural networks, Chemometrics and intelligent laboratory systems 39 (1) (1997) 43–62.

[11] G. Bebis, M. Georgiopoulos, Feed-forward neural networks, Ieee Potentials 13 (4) (1994) 27–31.

[12] N. Momo, Abdullah, J. Uddin, Speech recognition using feed forward neural network and principle component analysis, in: Advances in Signal Processing and Intelligent Recognition Systems: Proceedings of Third International Symposium on Signal Processing and Intelligent Recognition Systems (SIRS-2017), September 13-16, 2017, Manipal, India, Springer, 2018, pp. 228–239.

[13] M. F. Alghifari, T. S. Gunawan, M. Kartiwi, Speech emotion recognition using deep feedforward neural network, Indonesian Journal of Electrical Engineering and Computer Science 10 (2) (2018) 554–561.

[14] M. Awais, M. J. Iqbal, I. Ahmad, M. O. Alassafi, R. Alghamdi, M. Basheri, M. Waqas, Real-time surveillance through face recognition using hog and feedforward neural networks, IEEE Access 7 (2019) 121236–121244.

[15] A. Eleyan, H. Demirel, Face recognition system based on pca and feedforward neural networks, in: International Work-Conference on Artificial Neural Networks, Springer, 2005, pp. 935–942.

[16] J. G. Kuschewski, S. Hui, S. H. Zak, Application of feedforward neural networks to dynamical system identification and control, IEEE transactions on control systems technology 1 (1) (1993) 37–49.

[17] P. An, W. M. Moon, Reservoir characterization using feedforward neural networks, SEG Technical Program Expanded Abstracts 1993 (1993) 258–262.

[18] L. Ma, K. Khorasani, Facial expression recognition using constructive feedforward neural networks, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 34 (3) (2004) 1588–1595.

[19] R. Hecht-Nielsen, Theory of the backpropagation neural network, in: Neural networks for perception, Elsevier, 1992, pp. 65–93.

[20] S. Ruder, An overview of gradient descent optimization algorithms, arXiv preprint arXiv:1609.04747.

[21] D. Rumerhart, Learning internal representation by error propagation, Parallel distributed processing 1 (1986) 318–362.

[22] K. Hussain, M. N. Mohd Salleh, S. Cheng, Y. Shi, Metaheuristic research: a comprehensive survey, Artificial intelligence review 52 (2019) 2191–2233.

[23] M. Abdel-Basset, L. Abdel-Fatah, A. K. Sangaiah, Metaheuristic algorithms: A comprehensive review, Computational intelligence for multimedia big data on the cloud with engineering applications (2018) 185–231.

[24] X.-S. Yang, Nature-inspired metaheuristic algorithms, Luniver press, 2010.

[25] A. Anand, H. Batra, S. K. Syal, Dynamic neighborhood-based grey wolf optimizer with dimension learning-based hunting and lévy flight, in: 2023 Intelligent Methods, Systems, and Applications (IMSA), 2023, pp. 622–627. doi:10.1109/IMSA58542.2023.10217457.

[26] J. Kennedy, Swarm intelligence, in: Handbook of nature-inspired and innovative computing: integrating classical models with emerging technologies, Springer, 2006, pp. 187–219.

[27] D. Karaboga, B. Akay, A comparative study of artificial bee colony algorithm, Applied mathematics and computation 214 (1) (2009) 108–132.

[28] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95-international conference on neural networks, Vol. 4, IEEE, 1995, pp. 1942–1948.

[29] R. Mendes, P. Cortez, M. Rocha, J. Neves, Particle swarms for feedforward neural network training, in: Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290), Vol. 2, IEEE, 2002, pp. 1895–1899.

[30] A. Biswas, K. Mishra, S. Tiwari, A. Misra, Physics-inspired optimization algorithms: a survey, Journal of Optimization 2013.

[31] E. Rashedi, H. Nezamabadi-Pour, S. Saryazdi, Gsa: a gravitational search algorithm, Information sciences 179 (13) (2009) 2232–2248.

[32] V. K. Bohat, K. Arya, An effective gbest-guided gravitational search algorithm for real-parameter optimization and its application in training of feedforward neural networks, Knowledge-Based Systems 143 (2018) 192–207.
[33] P. Ghannadi, S. S. Kourehli, Multiverse optimizer for structural damage detection: Numerical study and experimental validation, The Structural Design of Tall and Special Buildings 29 (13) (2020) e1777.
[34] H. Shareef, A. A. Ibrahim, A. H. Mutlag, Lightning search algorithm, Applied Soft Computing 36 (2015) 315–333.
[35] C. A. Coello Coello, A comprehensive survey of evolutionary-based multiobjective optimization techniques, Knowledge and Information systems 1 (3) (1999) 269–308.
[36] S. Mirjalili, S. Mirjalili, Genetic algorithm, Evolutionary Algorithms and Neural Networks: Theory and Applications (2019) 43–55.
[37] S. Das, P. N. Suganthan, Differential evolution: A survey of the state-of-the-art, IEEE transactions on evolutionary computation 15 (1) (2010) 4–31.
[38] C. A. C. Coello, R. L. Becerra, Evolutionary multiobjective optimization using a cultural algorithm, in: Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706), IEEE, 2003, pp. 6–13.
[39] Y. Shi, Brain storm optimization algorithm, in: Advances in Swarm Intelligence: Second International Conference, ICSI 2011, Chongqing, China, June 12-15, 2011, Proceedings, Part I 2, Springer, 2011, pp. 303–309.
[40] E. Tanyildizi, G. Demir, Golden sine algorithm: A novel math-inspired algorithm., Advances in Electrical & Computer Engineering 17 (2).
[41] A. F. Nematollahi, A. Rahiminejad, B. Vahidi, A novel meta-heuristic optimization method based on golden ratio in nature, Soft Computing 24 (2020) 1117–1151.
[42] S. Chattopadhyay, A. Dey, P. K. Singh, Z. W. Geem, R. Sarkar, Covid-19 detection by optimizing deep residual features with improved clustering-based golden ratio optimizer, Diagnostics 11 (2) (2021) 315.
[43] K. Nusair, F. Alasali, Optimal power flow management system for a power network with stochastic renewable energy resources using golden ratio optimization method, Energies 13 (14) (2020) 3671.
[44] A. Dey, S. Chattopadhyay, P. K. Singh, A. Ahmadian, M. Ferrara, R. Sarkar, A hybrid meta-heuristic feature selection method using golden ratio and equilibrium optimization algorithms for speech emotion recognition, IEEE Access 8 (2020) 200953–200970.
[45] S. Mirjalili, Sca: a sine cosine algorithm for solving optimization problems, Knowledge-based systems 96 (2016) 120–133.
[46] Min Han, Jianchao Fan, Jun Wang, A Dynamic Feedforward Neural Network Based on Gaussian Particle Swarm Optimization and its Application for Predictive Control, IEEE Transactions on Neural Networks 22 (9) (2011) 1457–1468. doi:10.1109/TNN.2011.2162341.
URL http://ieeexplore.ieee.org/document/5963724/
[47] S. McLoone, M. Brown, G. Irwin, A. Lightbody, A hybrid linear/nonlinear training algorithm for feedforward neural networks, IEEE Transactions on Neural Networks 9 (4) (1998) 669–684. doi:10.1109/72.701180.
URL http://ieeexplore.ieee.org/document/701180/
[48] Y. Xue, T. Tang, A. X. Liu, Large-Scale Feedforward Neural Network Optimization by a Self-Adaptive Strategy and Parameter Based Particle Swarm Optimization, IEEE Access 7 (2019) 52473–52483. doi:10.1109/ACCESS.2019.2911530.
URL https://ieeexplore.ieee.org/document/8692438/
[49] J.-R. Zhang, J. Zhang, T.-M. Lok, M. R. Lyu, A hybrid particle swarm optimization–back-propagation algorithm for feedforward neural network training, Applied Mathematics and Computation 185 (2) (2007) 1026–1037. doi:10.1016/j.amc.2006.07.025.
URL https://linkinghub.elsevier.com/retrieve/pii/S0096300306008277
[50] K. Socha, C. Blum, An ant colony optimization algorithm for continuous optimization: application to feed-forward neural network training, Neural Computing and Applications 16 (3) (2007) 235–247. doi:10.1007/s00521-007-0084-z.
URL http://link.springer.com/10.1007/s00521-007-0084-z
[51] E. Pashaei, E. Pashaei, Training Feedforward Neural Network Using Enhanced Black Hole Algorithm: A Case Study on COVID-19 Related ACE2 Gene Expression Classification, Arabian Journal for Science and Engineering 46 (4) (2021) 3807–3828. doi:10.1007/s13369-020-05217-8.
URL http://link.springer.com/10.1007/s13369-020-05217-8
[52] L. K. Li, S. Shao, K.-F. C. Yiu, A new optimization algorithm for single hidden layer feedforward neural networks, Applied Soft Computing 13 (5) (2013) 2857–2862. doi:10.1016/j.asoc.2012.04.034.
URL https://linkinghub.elsevier.com/retrieve/pii/S156849461200230X
[53] Y.-D. Zhang, L. Wu, WEIGHTS OPTIMIZATION OF NEURAL NETWORK VIA IMPROVED BCO APPROACH, Progress In Electromagnetics Research 83 (2008) 185–198. doi:10.2528/PIER08051403.
URL http://www.jpier.org/PIER/pier.php?paper=08051403
[54] T.-K. Dao, J. Yu, T.-T. Nguyen, T.-G. Ngo, A Hybrid Improved MVO and FNN for Identifying Collected Data Failure in Cluster Heads in WSN, IEEE Access 8 (2020) 124311–124322. doi:10.1109/ACCESS.2020.3005247.
URL https://ieeexplore.ieee.org/document/9130710/
[55] I. Benmessahel, K. Xie, M. Chellal, T. Semong, A new evolutionary neural networks based on intrusion detection systems using locust swarm optimization, Evolutionary Intelligence 12 (2) (2019) 131–146. doi:10.1007/s12065-019-00199-5.
URL http://link.springer.com/10.1007/s12065-019-00199-5
[56] J.-S. Wang, S. Han, Feed-forward neural network soft-sensor modeling of flotation process based on particle swarm optimization and gravitational search algorithm, Computational Intelligence and Neuroscience 2015.
[57] T. Sağ, Z. Abdullah Jalil Jalil, Vortex search optimization algorithm for training of feed-forward neural network, International Journal of Machine Learning and Cybernetics 12 (5) (2021) 1517–1544.
[58] H. Taud, J. Mas, Multilayer perceptron (mlp), Geomatic approaches for modeling land change scenarios (2018) 451–455.
[59] V. Kůrková, Kolmogorov's theorem and multilayer neural networks, Neural Networks 5 (3) (1992) 501–506. doi:https://doi.org/10.1016/0893-6080(92)90012-8.
URL https://www.sciencedirect.com/science/article/pii/0893608092900128
[60] A. Al Bataineh, D. Kaur, S. M. J. Jalali, Multi-layer perceptron training optimization using nature inspired computing, IEEE Access 10 (2022) 36963–36977.

[61] S. R. Dubey, S. K. Singh, B. B. Chaudhuri, Activation functions in deep learning: A comprehensive survey and benchmark, Neurocomputing.
[62] A. F. Agarap, Deep learning using rectified linear units (relu), arXiv preprint arXiv:1803.08375.
[63] A. Maniatopoulos, N. Mitianoudis, Learnable leaky relu (lelelu): An alternative accuracy-optimized activation function, Information 12 (12) (2021) 513.
[64] A. F. Kamaruzaman, A. M. Zain, S. M. Yusuf, A. Udin, Levy flight algorithm for optimization problems-a literature review, Applied mechanics and materials 421 (2013) 496–501.
[65] R. Jensi, G. W. Jiji, An enhanced particle swarm optimization with levy flight for global optimization, Applied Soft Computing 43 (2016) 248–261.
[66] T. Hassanzadeh, H. Vojodi, A. M. E. Moghadam, A multilevel thresholding approach based on levy-flight firefly algorithm, in: 2011 7th Iranian Conference on Machine Vision and Image Processing, IEEE, 2011, pp. 1–5.
[67] L. Wu, J. Wu, T. Wang, Enhancing grasshopper optimization algorithm (goa) with levy flight for engineering applications, Scientific Reports 13 (1) (2023) 124.

Appendix A. Details of benchmark Functions

Table A.3: CEC 2014 benchmark functions (summary)

Type	Function ID	Function	Global minima
Unimodal	f_{101}	Rotated high conditioned elliptic function	100
	f_{102}	Rotated bent cigar function	200
	f_{103}	Rotated discus function	300
Multimodal	f_{104}	Shifted and rotated Rosenbrock function	400
	f_{105}	Shifted and rotated Ackley's function	500
	f_{106}	Shifted and rotated Weierstrass function	600
	f_{107}	Shifted and rotated Griewank's function	700
	f_{108}	Shifted Rastrigin function	800
	f_{109}	Shifted and rotated Rastrigin function	900
	f_{110}	Shifted Schwefel's function	1000
	f_{111}	Shifted and rotated Schwefel's function	1100
	f_{112}	Shifted and rotated Katsuura function	1200
	f_{113}	Shifted and rotated HappyCat function	1300
	f_{114}	Shifted and rotated HGbat function	1400
	f_{115}	Shifted and rotated Expanded Griewank's plus Rosenbrock's function	1500
	f_{116}	Shifted and rotated Expanded Scaffer's F6 function	1600
Hybrid	f_{117}	Hybrid function 1 (f_{109}, f_{108}, f_{101})	1700
	f_{118}	Hybrid function 2 (f_{102}, f_{112}, f_{108})	1800
	f_{119}	Hybrid function 3 (f_{107}, f_{106}, f_{104}, f_{114})	1900
	f_{120}	Hybrid function 4 (f_{112}, f_{103}, f_{113}, f_{108})	2000
	f_{121}	Hybrid function 5 (f_{114}, f_{112}, f_{104}, f_{109}, f_{101})	2100
	f_{122}	Hybrid function 6 (f_{110}, f_{111}, f_{113}, f_{109}, f_{105})	2200
Composition	f_{123}	Composition function 1 (f_{104}, f_{101}, f_{102}, f_{103}, f_{101})	2300
	f_{124}	Composition function 2 (f_{110}, f_{109}, f_{114})	2400
	f_{124}	Composition function 3 (f_{111}, f_{109}, f_{101})	2500
	f_{126}	Composition function 4 (f_{111}, f_{113}, f_1, f_{106}, f_{107})	2600
	f_{127}	Composition function 5 (f_{114}, f_{109}, f_{111}, f_{106}, f_{101})	2700
	f_{128}	Composition function 6 (f_{115}, f_{113}, f_{111}, f_{116}, f_{101})	2800
	f_{129}	Composition function 7 (f_{117}, f_{118}, f_{119})	2900
	f_{130}	Composition function 8 (f_{120}, f_{121}, f_{122})	3000

Table A.4: List of first group of benchmark functions

Function	Mathematical Representation	Dim	Global Minima	Range
f_1	$f_1(X) = \sum_{i=1}^{Dim} x_i^2$	30	0	$(-100, 100)$
f_2	$f_2(X) = \sum_{i=1}^{Dim} \lvert x_i \rvert + \prod_{i=1}^{Dim} \lvert x_i \rvert$	30	0	$(-10, 10)$
f_3	$f_3(X) = \sum_{i=1}^{Dim} \left(\sum_{i=1}^{Dim} x_i \right)^2$	30	0	$(-100, 100)$
f_4	$f_4(X) = \max_i \{ \lvert x_i \rvert, 1 \leq i \leq D \}$	30	0	$(-100, 100)$
f_5	$f_5(X) = \sum_{i=1}^{Dim-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	30	0	$(-30, 30)$
f_6	$f_6(X) = \sum_{i=1}^{Dim} (\lfloor x_i + 0.5 \rfloor)^2$	30	0	$(-100, 100)$
f_7	$f_7(X) = \sum_{i=1}^{Dim} i x_i + rand[0, 1)$	30	0	$(-1.28, 1.28)$
f_8	$f_8(X) = -\sum_{i=1}^{Dim} (x_i sin(\sqrt{\lvert x_i \rvert}))$	30	-12569.5	$(-500, 500)$
f_9	$f_9(X) = \sum_{i=1}^{Dim} [x_i^2 - 10cos(2\pi x_i) + 10]$	30	0	$(-5.12, 5.12)$
f_{10}	$f_{10}(X) = -20exp\left(-0.2 \sqrt{\frac{1}{D} \sum_{i=1}^{Dim} x_i^2} \right)$	30	0	$(-32, 32)$
f_{11}	$f_{11}(X) = \frac{1}{4000} \sum_{i=1}^{Dim} x_i^2 - \prod_{i=1}^{Dim} cos(\frac{x_i}{\sqrt{i}}) + 1$	30	0	$(-600, 600)$
f_{12}	$f_{12}(X) = \frac{\pi}{Dim} \left\{ sin^2(3\pi y_1) + \sum_{i=1}^{Dim-1} (y_i - 1)^2 [1 + sin^2(3\pi y_{i+1})] \right.$	30	0	$(-50, 50)$
	$\left. + (y_{Dim} - 1)^2 \right\} + \sum_{i=1}^{Dim} u(x_i, 10, 100, 4)$			
f_{13}	$f_{13}(X) = 0.1 \left\{ sin^2(3\pi x_1) + \sum_{i=1}^{Dim-1} (x_i - 1)^2 [1 + sin^2(3\pi x_{i+1})] \right.$	30	0	$(-50, 50)$
	$\left. + (x_{Dim} - 1)^2 [1 + sin^2(3\pi x_{Dim})] \right\} + \sum_{i=1}^{Dim} u(x_i, 5, 100, 4)$			
f_{14}	$f_{14}(X) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6} \right]^{-1}$	2	1	$(-65.536, 65.536)$
f_{15}	$f_{15}(X) = \sum_{i=1}^{11} \left[a_i - \frac{x_1 (b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4} \right]^2$	4	0.0003	$(-5, 5)$
f_{16}	$f_{16}(X) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1 x_2 - 4x_2^2 + 4x_2^4$	2	-1.0316	$(-5, 5)$
f_{17}	$f_{17}(X) = \left(x_2 - \frac{5.1}{4\pi^2} x_1 + \frac{5}{\pi} x_1 - 6 \right)^2 + 10 \left(1 - \frac{1}{8\pi} \right) cos x_1 + 10$	2	0.398	$(-5, 10) \& (0, 15)$
f_{18}	$f_{18}(X) = \left[1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1 x_2 + 3x_2^2) \right]$	2	3	$(-2, 2)$
	$\times \left[30 + (2x_1 - 3x_2)^2 \times (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1 x_2 + 27x_2^2) \right]$			
f_{19}	$f_{19}(X) = -\sum_{i=1}^{4} exp\left[-\sum_{j=1}^{3} a_{ij} (x_j - p_{ij})^2 \right]$	3	-3.86	$(0, 1)$
f_{20}	$f_{20}(X) = -\sum_{i=1}^{4} exp\left[-\sum_{i=1}^{6} a_{ij} (x_j - p_{ij})^2 \right]$	6	-3.32	$(0, 1)$