mathematics

MDPI

Article

A New Parallel Cuckoo Flower Search Algorithm for Training Multi-Layer Perceptron

Rohit Salgotra [1,2,*], Nitin Mittal [3] and Vikas Mittal [3]

[1] Faculty of Physics and Applied Computer Science, AGH University of Science & Technology, 30-059 Krakow, Poland
[2] MEU Research Unit, Middle East University, Amman 11813, Jordan
[3] University Centre for Research and Development, Chandigarh University, Mohali 140413, India; mittal.nitin84@gmail.com (N.M.); vikas.e14122@cumail.in (V.M.)
* Correspondence: rohits@agh.edu.pl or r.03dec@gmail.com

Abstract: This paper introduces a parallel meta-heuristic algorithm called Cuckoo Flower Search (CFS). This algorithm combines the Flower Pollination Algorithm (FPA) and Cuckoo Search (CS) to train Multi-Layer Perceptron (MLP) models. The algorithm is evaluated on standard benchmark problems and its competitiveness is demonstrated against other state-of-the-art algorithms. Multiple datasets are utilized to assess the performance of CFS for MLP training. The experimental results are compared with various algorithms such as Genetic Algorithm (GA), Grey Wolf Optimization (GWO), Particle Swarm Optimization (PSO), Evolutionary Search (ES), Ant Colony Optimization (ACO), and Population-based Incremental Learning (PBIL). Statistical tests are conducted to validate the superiority of the CFS algorithm in finding global optimum solutions. The results indicate that CFS achieves significantly better outcomes with a higher convergence rate when compared to the other algorithms tested. This highlights the effectiveness of CFS in solving MLP optimization problems and its potential as a competitive algorithm in the field.

Keywords: evolutionary algorithm; neural networks; FNN; multi-layer perceptron; cuckoo flower search

MSC: 90C26; 68U05

Citation: Salgotra, R.; Mittal, N.; Mittal, V. A New Parallel Cuckoo Flower Search Algorithm for Training Multi-Layer Perceptron. *Mathematics* **2023**, *11*, 3080. https://doi.org/10.3390/math11143080

Academic Editors: Thomas Hanne, Zhihua Cui, Gai-Ge Wang, Linqiang Pan and Harish Garg

Received: 21 June 2023
Revised: 7 July 2023
Accepted: 10 July 2023
Published: 12 July 2023

1. Introduction

Over the past decades, artificial intelligence (AI), and particularly machine learning (ML), has paved the way for researchers to study nature and build problem solving models. In particular, studying the phenomena of natural selection, social behavior, and other patterns has led to the rise of evolutionary computing, swarm intelligence, and neural networks (NN). NN are the most significant invention in the arena of soft computing, inspired by neurons present in human brain. The basic NN model was conceptualized by McCulloch and Pitts [1]. There are various types of NNs, including Kohonen self-organizing networks [2], recurrent NN [3], spiking NN [4], feed-forward networks (FNN) [5], and others. Among these NNs, FNN are the simplest, with low computational cost and high performance. FNN receive input from one side and provides output at the other. The FNN is generally unidirectional, with multiple layers in between. If there is only a single layer, the network is called a Single-layer perceptron (SLP) [6]. SLPs are used for solving linear problems. If there are multiple layers, called a multi-layer perceptron (MLP) [1,7], these networks are used to solve non-linear problems.

All NNs have a common feature of learning from experience. Such NNs are called Artificial NN (ANN), and they adapt themselves according to given set of inputs. ANNs can be supervised using an external source for providing feedback [8,9], or they can be unsupervised [10,11], taking the form of a NN that adapts to its own inputs without any

external feedback. Training NNs to achieve the highest possible performance is performed by a trainer. The trainer provides the NN with a set of input samples, modifies it with the structural parameters of the NN, and finally, when the training process is complete, the trainer is omitted and the NN is set as active and is available for use. There are two types of trainers: deterministic and stochastic. Supervised learning to solve problems, brought about with the advent of the Back propagation (BP) algorithm [12] and the gradient search algorithm, are deterministic methods that aim at training through mathematical optimization to achieve maximum performance. These trainers are simple and have higher convergence speed, leading to a global optimum from a single solution. These optimization methods have a problem of becoming in a local optima that is sometimes mistaken as global optima. On the other hand, stochastic training methods use stochastic optimization methods to achieve desired performance. These methods initiate training with a random solution and enhance it to achieve a global optimum. Randomness in stochastic methods provides local optima avoidance but these methods are slower than deterministic methods [13,14]. Stochastic trainers are generally used in literature due to high avoidance of local optimum.

Stochastic trainers can be single-solution or multi-solution. For a single-solution, the NN is constructed by training it with a single random solution and evolving it iteratively until stopping criteria is satisfied. Simulated annealing (SA) [15,16], hill climbing [17], and others [18,19] are examples of single-solution NNs. Multi-solution NNs, on the other hand, are initiated with multiple random solutions and evolve each solution unless the stopping criteria is met. These criteria include Genetic algorithm (GA) [20], Ant colony optimization (ACO) [21], Artificial Bee colony (ABC) [22,23], Particle swarm optimization (PSO) [24,25], Differential evolution (DE) [26], Teacher-learning based optimization (TLBO) [27], Invasive weed algorithm (IWO) [28], ensemble techniques [29], Grey Wolf optimization (GWO) [30], and others. These algorithms have high performance in terms of finding approximate global optimum solutions. This inspires us to develop a new meta-heuristic and apply it efficiently for training NNs.

In this work, a new parallel algorithm based on Cuckoo Search (CS) [31] and Flower Pollination Algorithm (FPA) [32], which we have named Cuckoo Flower Search (CFS), is introduced. The main motivation for this work is the problem of local optima stagnation and premature convergence problems of already existing algorithms. CFS has been tested on standard benchmark functions and compared with state-of-the-art algorithms for establishing its competitiveness. In addition, it has been further tested on FNN-MLP as an application to real world problems. Nineteen benchmark functions have been used to analyze the performance of the proposed algorithm. These benchmark functions consist of unimodal functions, multi-modal functions, and fixed-dimension functions. These problems are highly challenging and any algorithm performing well on these functions is considered to be a good algorithm. A comparison with GWO, CS, FPA, BFP, and others was also conducted. Statistical tests have also been performed to prove the superiority of CFS over other comparable algorithms. The major contributions of the paper are highlighted as:

- To avoid premature convergence and local optima stagnation, best known properties of FPA and CS are added to the proposed algorithm.
- The global and local search phase equations of FPA and CS are optimized for addition in the proposed algorithm.
- Solutions generated by FPA and CS are compared and best among the two is selected as the current best solution. These solutions are further generated over the course of iterations to find the global best solution.
- A greedy selection operation is followed for retaining the best solution over subsequent iterations.
- The proposed algorithm is tested on 19 classical benchmark functions, and Wilcoxon rank-sum test is done to prove the significance of the algorithm statistically.
- Finally, five real-world datasets, including Heart, Breast cancer, Iris, Ballon, and XOR, are optimized using the proposed algorithm.

- The source code of CFS algorithm is available at: https://github.com/rohitsalgotra/CFS (accessed on 20 June 2023).

The rest of the paper is organized as follows: Section 2 describes the preliminary definitions of FNN and MLP. The basics of CS and FPA are detailed in Section 3. Section 4 describes the proposed CFS algorithm. Section 5 presents with the results and discussion. Finally, Section 6 concludes the paper.

2. Feed-Forward Neural Networks and Multi-Layer Perceptron

FNNs are that are unidirectional networks and have a one-way connection between neurons. They contain several parallel layers in which neurons are arranged [33]. The first layer is the input layer and the last last is the output layer. In between these are several other layers that correspond to hidden layers. A three-layer MLP with n input nodes, h hidden nodes, and m number of outputs is shown in Figure 1, showing a simple unidirectional connection between the nodes. The outputs are calculated as in [34]:

- Weighted sum of inputs is given by:

$$s_j = \sum_{i=1}^{n} \left(W_{ij} \cdot W_i\right) - \theta_j,\ j = 1, 2, \ldots h \tag{1}$$

- Outputs of hidden layers are calculated as:

$$s_j = sigmoid\left(s_j\right) = \frac{1}{\left(1 + \exp\left(-s_j\right)\right)},\ j = 1, 2, \ldots h \tag{2}$$

- Final output based on the hidden node outputs is given as:

$$o_k = \sum_{j=1}^{h} \left(W_{jk} \cdot S_j\right) - \theta_k,\ k = 1, 2, \ldots m \tag{3}$$

$$o_k = sigmoid\left(o_k\right) = \frac{1}{\left(1 + \exp\left(-o_k\right)\right)},\ k = 1, 2, \ldots m \tag{4}$$

where W_{ij} and W_{jk} are weight connection of ith node in input layer to jth node in the hidden layer and from jth hidden layer to kth output layer, respectively, θ_j and θ_k are the threshold of jth hidden layer and kth output layer, respectively, and X_i is the ith input layer.

From the above equations, it can be seen that weights and thresholds define the final value of the MLPs. The major concern is finding optimum weights and thresholds (biases) for achieving a balanced relation between input and outputs.

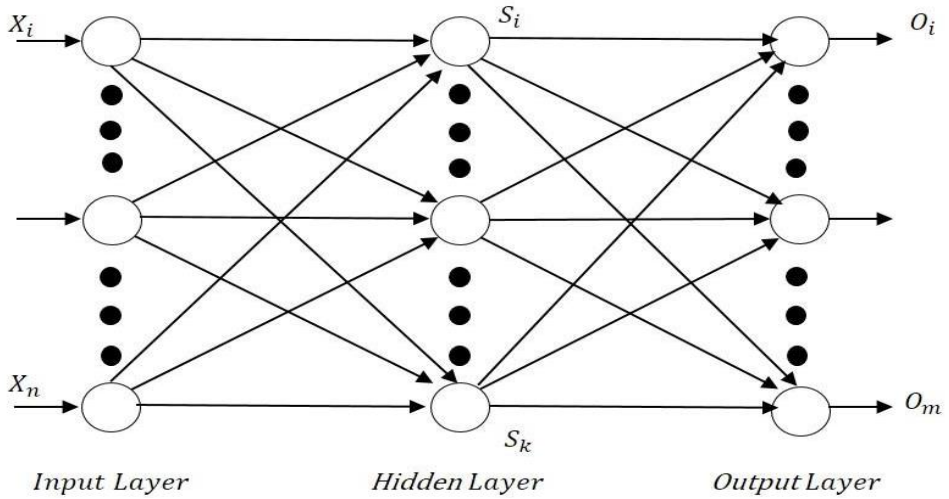

Figure 1. An MLP with one hidden node.

3. Basic Cuckoo Search and Flower Pollination Algorithm

3.1. Cuckoo Search Algorithm

The CS algorithm is inspired by the obligatory brood parasitism behavior of cuckoos [35]. The cuckoos of some species lay their eggs in the nests of host birds, following an obligate brood parasitism. CS is a competitive algorithm among existing algorithms. CS contains components including the selection of best solution and ensuring that this best solution is passed on to the next generation. It employs local random walk to perform the exploitation locally and randomization via Lévy flights to perform the exploration globally. Three rules are established that describe the Cuckoo Search in a simple way. These are explained as follows:

- Each cuckoo lays one egg and dumps it in a random nest;
- The nest with highest fitness will carry over to next generation;
- The host bird discovered the cuckoo's egg with a probability $p_a \in [0, 1]$. A fixed number of host nests are available. Depending on p_a, a new nest is built by the host bird at a new location either by throwing the egg away from the nest or abandoning the nest.

In CS, a solution is an egg that is already present in a nest and a new solution is that egg which is laid by a cuckoo. The not-so-good solutions of nests are replaced by the new and better solutions [35]. More complicated cases arise when multiple eggs are present in each nest. In these cases, the extended form of this algorithm can be used. Based on the above three rules, Equation (5) derives the Levy flight that is performed to produce a new solution x_i^{t+1} for ith cuckoo:

$$x_i^{t+1} = x_i^t + \alpha \bigoplus \text{Lévy } (\lambda) \tag{5}$$

where the previous solution is denoted by x_i^t, $\bigoplus$ is entry wise multiplication, and $\alpha > 0$ is the step size. In most cases, $\alpha = 1$ is used. The above equation is the stochastic equation for random walk. In the case of random walk, the current location draws a path to next status/location and the transition probability of next position. PSO also used this type of entry-wise product.

Cuckoos usually search for food using a basic random walk. This is a Markov chain whose updated position is determined by the present location and the transition probability of the following position. The performance of CS is enhanced using Lévy flights [36]. Lévy flight is a random walk measured in step-lengths following a heavy-tailed probability distribution. Ultimately, Levy flight is not a continuous space; it is used to refer to a discrete grid [37–39]. The Levy flight is employed in this study as a result of Levy flight's greater efficiency in exploring the search space. Our algorithm is generated from a Levy distribution with infinite mean and variance.

As the random walk occurs via Lévy flight, the Lévy distribution draws the random step length as:

$$\text{Lévy } \sim \mathrm{u} = t^{-\lambda}, \qquad (1 < \lambda \leq 3) \tag{6}$$

This random walk process is a heavy tail step-length distribution. The Lévy walk achieves new solutions nearer to the best solutions to speed up the local search [36]. Far field randomization should be used to create some of the solutions in order to prevent the system getting stuck in a local optimum. Here, some points are discussed that show that CS is analogous to and competitive with other optimization algorithms. First, as with other GA and PSO algorithms, CS is a population-based algorithm. Second, because of the heavy tailed step length, the large step is possible in CS and the randomization is more efficient. Third, a wide class of optimization problems have adapted to the CS because it tunes fewer parameters when compared to PSO and GA.

3.2. Flower Pollination Algorithm

Flowers are fascinating species. Dating from the Cretaceous period, flowers are estimated to comprise about 80 percent of the total species of plants [40]. About 250,000 species of flowers have been found on earth. The ultimate aim of flowers is to reproduce and this reproduction occurs mainly by pollination. In pollination, pollen is transferred from one flower to other by pollinators. Cross-pollination means that pollination occurs due to pollen from different plants. On the other hand, self-pollination means fertilization of pollen from the same or different flowers of the same plant. Pollinators can be insects, birds, or any other animal. Some flowers do attract only specific kinds of insects for pollination, showing a sort of flower-insect partnership; this is referred to as called flower constancy. Pollinators such as honeybees have been found to develop flower constancy. This property helps pollinators to visit only particular plant species, hence increasing the chances of reproduction for the flower and, in turn, maximizing nectar supply for the pollinator [41].When the pollen is shifted by pollinators such as insects and animals, the process is called biotic pollination (about 90 percent occurs via biotic). Meanwhile, when it occurs via diffusion or wind, the process is called abiotic [42] (this constitutes about 10 percent of pollination). In total, there are about 200,000 varieties of pollinators found on earth. Biotic cross-pollination occurs over long distance and is facilitated by birds, bats, bees, and fireflies, among other animals. This is often referred to as global pollination. Meanwhile, self-pollination is termed as local pollination.

The above characteristics are idealized into four set of rules [43]:

- Global pollination arises via biotic and cross-pollination.
- Local pollination occurs via abiotic and self-pollination.
- Flower constancy, termed as reproduction probability, is proportional to the similarity of two flowers.
- Switch probability p ϵ [0, 1] balances global and local pollination.

When designing the algorithm, it is expected that each plant has only one flower producing only a single pollen gamete. Following this, we can use y_i as a solution equivalent to a flower or a pollen gamete, defining a single objective problem.

The above characteristics have been combined to design an FPA that mainly consists of local and global pollination. In the earlier version, pollination and reproduction of the fittest flower is ensured and the rules are represented mathematically as:

$$y_i^{t+1} = y_i^t + \alpha L(\lambda)\left(R_* - y_i^t\right) \tag{7}$$

where R_* is the current best solution, y_i^t is the potential solution at t iteration, and α is the scaling factor to control the Lévy flight-based step size $L(\lambda)$. Lévy flight is expressed as:

$$L \sim \left\{ \frac{\lambda \Gamma\,(\lambda)\sin(\pi\lambda/2)}{\pi} \frac{1}{s^{1+\lambda}}, \quad (s \gg s_0 > 0) \right. \tag{8}$$

where $\Gamma(\lambda)$ is the standard gamma function.

The local pollination rule can be mathematically represented as:

$$y_i^{t+1} = y_i^t + \epsilon\left(y_j^t - y_k^t\right) \tag{9}$$

where y_j^t and y_k^t are pollens from diverse flowers of the same plants. In the confined space, flower constancy corresponds to a local random walk, and is selected from a uniform distribution ϵ in [0, 1].

4. Cuckoo Flower Search Algorithm

4.1. Algorithm Definition

The CFS algorithm is proposed as a hybrid version of the CS and FPA algorithms. Both these algorithms work in coordination to attain a global optimum solution. The

main idea is to generate the current best solution for both cuckoos and flower pollinators. After finding this solution, both are compared, the best solution is considered, and the process is repeated. The solution after first evaluation is fed back to the cuckoos and flower pollinators. This procedure is continued until the termination criteria are met. The final solution is the most appropriate solution to the problem under discussion. There are three phases to the proposed CFS algorithm:

Initialization

This is the first phase of the CFS algorithm, in which the population is randomly initialized. The solution is initialized according to Equation (10) and operates as a potential solution to the problem under examination, starting with an initial population of N cuckoos and flower pollinators (termed as CF).

$$CF_{i,j} = CF_{min,j} + a_b * \left(CF_{min,j} - CF_{max,j}\right) \tag{10}$$

where $i \epsilon \{1, \ldots .CF\}$, $j \epsilon \{1, \ldots ..D\}$, $CF_{i,j}$ is the ith solution in the jth dimension, D is the dimension or number of variables in the problem being studied, $CF_{min,j}$, $CF_{max,j}$ are the lower and upper bounds, respectively, and a_b is randomly generated number between [0, 1]. Here, the population initialized in Equation (10) is same for both cuckoos and flower pollinators. The fitness of the solution is estimated for objective function after initialization, and the best solution attained is treated as the initial best for all cuckoos (C_F) and flower pollinators (F_C).

- Solution generation

After the initial step, two new solutions are generated: one solution is inspired by cuckoo brood parasitism and the other from flower pollinators. The main concern here is to follow exploration and exploitation in a well-defined fashion. In cuckoos, exploration is achieved by randomization via Lévy flights. Local random walk is used to achieve exploitation. The new solution x_i^{t+1} is generated as per Equation (6) and its fitness is evaluated for the optimization problem being tested. The solution x_i^{t+1} obtained in this manner is compared with the C_F, and the best (C_F^{best}) among them is retained.

In the case of flower pollinators, exploration and exploitation is balanced by local pollination and global pollination, respectively. Equations (7) and (9), based on random probability in the search domain [0, 1], often called the switch probability, are used extensively to find the second new solution (y_i^{t+1}). This solution y_i^{t+1} and the initial best (F_C) solution are also compared, resulting in another best solution F_C^{best} among them.

- Final evaluation

After comparing the best fit solutions obtained by cuckoos (C_F^{best}) and flower pollinators (F_C^{best}), the best solution attained is the final optimum solution. For both cuckoos and flower pollinators, the solution generated at the last stage is set as the initial best (C_F and F_C, respectively) Unless and until the termination requirements are met, the same procedure is followed. The final solution obtained in this manner is the most appropriate solution. It is also worth noting that cuckoos and flower pollinators are both seeking the most appropriate solutions in parallel. If a cuckoo-produced solution becomes trapped in the local optimum and is unable to deliver the global optimal, flower pollinators assist it in exiting the local trap and achieving the global optimum, and vice versa. This characteristic increases the likelihood of the CFS algorithm reaching the global optimum solution.

Both CS and FPA are good algorithms in terms of finding global optimum solution, but the real problem is their inconsistency in finding the best fit individual every time the algorithm is run. This inconsistency is due to the problem of getting stuck in local optima while moving toward global optima. As a result, a better solution is required to move the algorithm closer to the global optimum. If the CS algorithm becomes stuck, FPA moves it towards the global optimum, and vice versa. As a result, both Cuckoos and Flower Pollinators collaborate analytically to obtain a global optimum. Figure 2 shows the

flow code for the CFS algorithm. The pseudocode of the proposed algorithm is given in Algorithm 1.

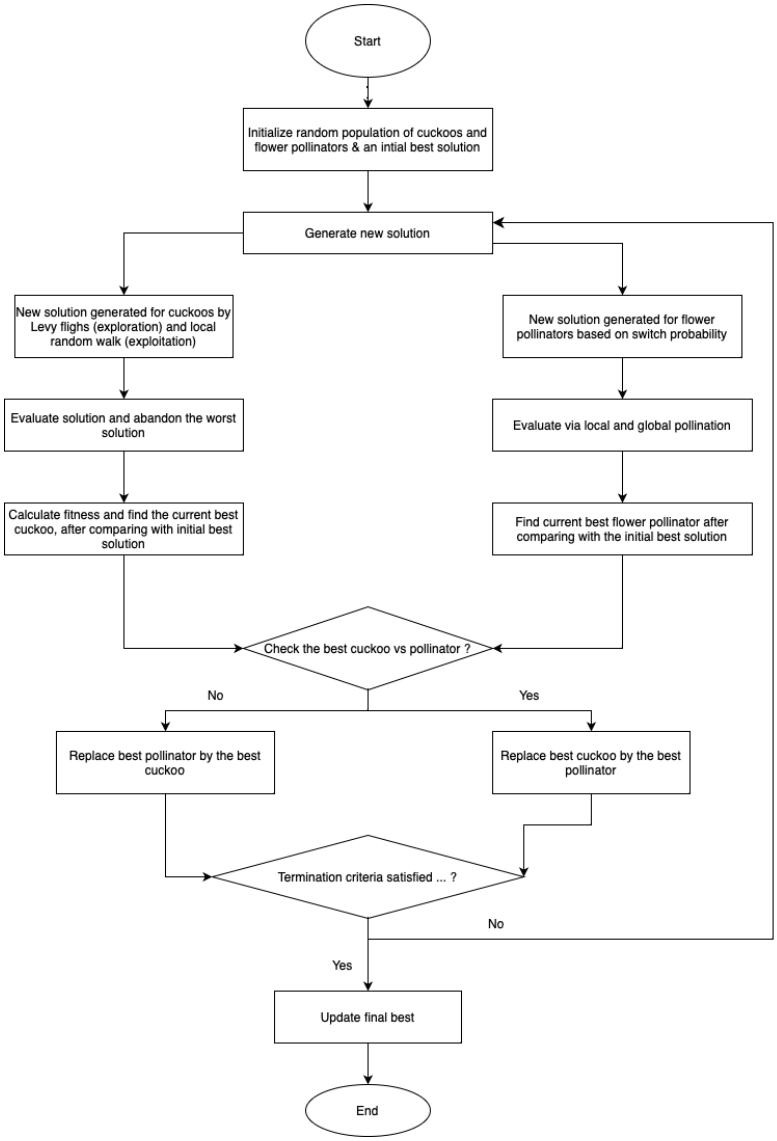

Figure 2. Flow-code for CFS algorithm.

Algorithm 1: Pseudocode of CFS algorithm

begin:
1. **Initialize:** α, β_0, γ, **maximum iterations**
2. **Define Population, objective function** $f(x)$
3. **While (***t* **<** *maximum iterations***)**
   - **For i = 1 to n**
     - **For j = 1 to n**
       - **Evaluate new solution using CS inspired equation;**
       - **Evaluate new solution using FPA inspired equation;**
     - **Find the best among the two using greedy selection;**
     - **End for** *j*
   - **End for** *i*
4. **Update current best.**
5. **End while**
6. **Find final best**

end.

4.2. CFS-MLP Trainer

When training a MLP, the first step is to formulate a problem [44] and find values of weights and biases with the highest accuracy/classification/statistical results. These should be found by a trainer. This important step is achieved by training an MLP using meta-heuristic algorithms. There are three methods for training MLPs using meta-heuristic algorithms [45]:

1. To find combination of weights and biases of MLP for achieving the minimum error using meta-heuristic algorithms. In this approach, proper values of weights are found without changing the basic architecture of the heuristic algorithm. It has simple a encoding phase and a difficult decoding phase, and so is often used for simple NNs.
2. To find proper architecture for an MLP using heuristic algorithms. In this method, the architecture varies and it can be achieved by varying the connections between hidden nodes, layers, and neurons, as proposed in [46]. This method has a simple decoding phase but, due to complexity in the encoding phase, it is used for complex structures.
3. To tune the gradient-based learning algorithm parameters using a heuristic approach. This method has been used to train FNNs using EAs [47] and others, such as GA [48], using a combination of methods to tune FNN. In this method, the decoding and encoding processes are very complicated and hence the structure becomes very complex.

In the present work, the CFS algorithm is proposed and applied to train an MLP using the first method. The weights and biases for the CFS algorithm are given in the form of a vector as follows:

$$C = (W, \theta) = (W_{1,1},\ W_{1,2}, \ldots,\ W_{n,n} | \theta_1,\ \theta_2,\ \ldots,\ \theta_h) \tag{11}$$

where n is the number of nodes, W_{ij} is the weight connection between ith and jth node, and θ_j is the bias of jth hidden node. After setting the initial variables, the fitness function is to be designed using CFS algorithm. This is achieved by defining a common metric for evaluation of the MLP and is called Mean Square Error (MSE). The MSE is used to calculate the difference between the desired output and the value obtained from MLP. The performance of MLP is based upon the average MSE values of all training samples and is given by:

$$MSE = \sum_{k=1}^{s} \frac{\sum_{i=1}^{m} (o_i^k - d_i^k)^2}{s} \tag{12}$$

where s is training samples count, m is the number of outputs, and o_i^k and d_i^k are the actual output and desired output of the ith input for the kth training sample, respectively. Based on MSE, the final objective function can be formulated as:

$$Minimize : F(C) = MSE \tag{13}$$

The overall process of using the CFS algorithm delivers MLP with weights as well as biases and, in turn, receives average MSE for all training samples. The CFS algorithm updates the weights and biases iteratively in order to achieve minimized average MSE. The best MSE is obtained from the last iteration of the algorithm. Since weights and biases find the best MSE in the MLP, there is a greater chance of improvement in the MLP structure at each iteration. Thus, the CFS algorithm converges toward a better solution than initial random solution.

5. Result and Discussion

This section presents the details on the applicability of the proposed CFS algorithm for classical benchmark problems and real-world optimization of FNN-MLP. We have used 19 benchmark functions, consisting of unimodal, multi-modal, and fixed dimension problems. For the optimization of real-world FNN-MLP, five highly challenging datasets have been used. More details on applicability are presented in the consecutive subsections.

For performance analysis, the simulations are performed on MATLAB, using a Windows 10 × 64, Intel Core i3 processor, with 8 GB RAM.

5.1. Benchmark Problems

To check the effectiveness of the CFS algorithm, it was tested on nineteen well known benchmark problems. The Wilcoxon rank-sum tests were performed to test the validity of results statistically. This non-parametric test is used to detect the static significance of any algorithm. Differences between two pairs of populations were analyzed and compared. The test returns a *p*-value determining the significance level of two algorithms. This value should be less than 0.05 for an algorithm to be statistically efficient [49]. The proposed algorithm is compared with ABC [50], Firefly Algorithm [51], FPA, CS, and Bat Flower Pollinator [52] algorithms. The parameter setting to test each algorithm for benchmark problems is shown in Table 1.

Table 1. Parameter settings for various algorithms.

Algorithm	Parameters	Values
FA	Number of fireflies	20
	Alpha (α)	0.5
	Beta (β)	0.2
	Gamma (γ)	1
	Stopping Criteria	200 Iterations
ABC	Swarm Size	20
	Limit	100
	Stopping Criteria	200 Iterations
FPA	Population Size	20
	Probability Switch	0.8
	Stopping Criteria	200 Iterations
CS	Population Size	20
	Discovery Rate of alien egg	0.25
	Maximum number of iterations	200
	Stopping Criteria	Max Iteration.
BFP	Population size	20
	Probability Switch	0.8
	Alpha (α)	0.5
	Stopping Criteria	200 Iterations
CFS	Population size	20
	Probability switch	0.8
	Discovery rate of alien egg (p_a)	0.25
	Stopping Criteria	200 Iterations

5.1.1. Unimodal Functions

There is no local solution for unimodal functions; they have a single global solution. These benchmark functions are useful for evaluating the convergence characteristics of heuristic optimization techniques. The CFS algorithm was applied to four unimodal benchmark problems with three dimension sets (30, 50, and 100), as given in Table 2. The CFS algorithm was compared to the ABC, FA, FPA, CS, and BFP algorithms (see Tables 3–8). For the 30 (Table 3) and 50 (Table 5) dimension (D) problems, with the function f_1, the FPA algorithm has a better mean and best value but CS and CFS are found to give the best values of standard deviation. For function f_2, the FA is found to be better, with a highly competitive result for the CFS algorithm. for f_3, the CFS algorithm provides better results, and for f_4, the ABC and CFS algorithms are both competitive when compared to rest of the algorithms. For 100 D (Table 7), the CFS algorithm performs better for f_2 and f_3. for f_1, FPA is better, and for f_4, ABC is better. The BFP algorithm is better for none of the functions.

The rank-sum tests from Tables 4, 6 and 8 acknowledge the superior performance of the CFS algorithm. The convergence characteristics are shown in Figure 3.

Table 2. Description of Unimodal Test functions.

Unimodal Test Problems	Objective Function	Search Range	Optimum Value	D
Schwefel function	$f_1(x) = \sum_{i=1}^{D} [x_i sin(\sqrt{\lvert x_i \rvert})]$	$[-500, 500]$	$-418.9829 \times D$	30, 50, 100
Sphere function	$f_2(x) = \sum_{i=1}^{D} x_i^2$	$[-100, 100]$	0	30, 50, 100
Elliptic function	$f_3(x) = \sum_{i=1}^{D} (10^6)^{\frac{i-1}{D-1}} x_i^2$	$[-100, 100]$	0	30, 50, 100
Scaffer function	$f_4(x) = [\frac{1}{n-1}\sqrt{s_i}.(\sin(50.0 s_i^{\frac{1}{5}}) + 1)]^2 s_i$ $= \sqrt{x_i^2 + x_{i+1}^2}$	$[-100, 100]$	0	30, 50, 100

Table 3. Results comparison for unimodal functions (30 Dimension).

Objective Function	Algorithm	Best	Worst	Mean	Standard Deviation
$f_1(x)$	CFS	-1.16×10^4	-1.03×10^4	-1.08×10^4	$\mathbf{3.47 \times 10^2}$
	FA	-4.85×10^3	-2.53×10^3	-3.78×10^3	6.61×10^2
	ABC	-9.65×10^3	-7.67×10^3	-8.68×10^3	4.93×10^2
	FPA	-6.36×10^{19}	-4.73×10^{15}	-3.72×10^{18}	1.41×10^{19}
	CS	-7.34×10^3	-6.49×10^3	-6.94×10^3	$\mathbf{2.30 \times 10^2}$
	BFP	-5.19×10^{10}	-2.08×10^3	-2.76×10^9	1.15×10^{10}
$f_2(x)$	CFS	1.0666	2.9397	2.0917	0.4731
	FA	**0.0282**	**0.0818**	**0.0567**	**0.0137**
	ABC	1.09×10^4	2.31×10^4	1.56×10^4	3.27×10^3
	FPA	9.52×10^3	2.28×10^4	1.53×10^4	3.20×10^3
	CS	2.93×10^2	1.23×10^3	8.07×10^2	2.45×10^2
	BFP	3.49×10^4	7.41×10^4	6.00×10^4	1.27×10^4
$f_3(x)$	CFS	$\mathbf{9.73 \times 10^3}$	$\mathbf{3.56 \times 10^4}$	$\mathbf{2.08 \times 10^4}$	$\mathbf{6.88 \times 10^3}$
	FA	1.95×10^6	1.66×10^7	6.96×10^6	4.06×10^6
	ABC	6.75×10^6	5.16×10^8	1.04×10^8	1.17×10^8
	FPA	1.60×10^8	5.09×10^8	2.81×10^8	8.27×10^7
	CS	9.32×10^5	5.87×10^6	2.31×10^6	1.13×10^6
	BFP	9.86×10^8	4.43×10^9	2.73×10^9	7.60×10^8
$f_4(x)$	CFS	**0**	6.43×10^{-14}	1.40×10^{-14}	1.74×10^{-14}
	FA	3.61×10^{-10}	0.0298	0.0066	0.0091
	ABC	**0**	**0**	**0**	**0**
	FPA	1.28×10^{-5}	0.0029	5.12×10^{-4}	7.13×10^{-4}
	CS	1.82×10^{-8}	5.84×10^{-5}	1.05×10^{-5}	1.73×10^{-5}
	BFP	4.67×10^{-2}	4.75×10^{-1}	3.25×10^{-1}	1.51×10^{-1}

Bold values in the table correspond to the best algorithmic values.

Table 4. P-test values of simulated algorithms for unimodal functions (30 Dimension).

Objective Function	FA	FPA	CS	ABC	CFS
$f_1(x)$	6.79×10^{-8}	**NA**	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}
$f_2(x)$	**NA**	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}
$f_3(x)$	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	**NA**
$f_4(x)$	8.00×10^{-9}	8.00×10^{-9}	8.00×10^{-9}	**NA**	**NA**

Table 5. Results comparison for unimodal functions (50 Dimension).

Objective Function	Algorithm	Best	Worst	Mean	Standard Deviation
$f_1(x)$	CFS	-1.66×10^4	-1.53×10^4	-1.60×10^4	$\mathbf{4.10 \times 10^2}$
	FA	-9.18×10^3	-3.88×10^3	-6.19×10^3	1.59×10^3
	ABC	-1.36×10^4	-1.11×10^4	-1.23×10^4	7.09×10^2
	FPA	$\mathbf{-1.18 \times 10^{20}}$	$\mathbf{-9.35 \times 10^{15}}$	$\mathbf{-7.75 \times 10^{18}}$	2.69×10^{19}
	CS	-1.08×10^4	-9.62×10^3	-1.00×10^4	$\mathbf{3.52 \times 10^2}$
	BFP	-6.32×10^{11}	-1.22×10^3	-3.31×10^{10}	1.41×10^{11}
$f_2(x)$	CFS	4.5385	11.9049	9.2753	4.5385
	FA	**0.1062**	**0.2069**	**0.1578**	**0.0303**
	ABC	5.83×10^3	1.81×10^4	1.37×10^4	3.21×10^3
	FPA	1.46×10^4	4.84×10^4	3.03×10^4	8.91×10^3
	CS	2.09×10^3	5.50×10^3	3.83×10^3	8.82×10^2
	BFP	9.06×10^4	1.43×10^5	1.18×10^5	1.63×10^4
$f_3(x)$	CFS	$\mathbf{1.14 \times 10^4}$	$\mathbf{3.03 \times 10^4}$	$\mathbf{1.95 \times 10^4}$	$\mathbf{5.71 \times 10^3}$
	FA	2.80×10^6	1.34×10^7	6.66×10^6	2.99×10^6
	ABC	2.91×10^7	1.14×10^9	5.12×10^8	3.16×10^9
	FPA	1.16×10^8	4.53×10^8	2.76×10^8	1.04×10^8
	CS	1.17×10^6	4.58×10^6	2.42×10^6	8.49×10^5
	BFP	1.71×10^9	4.60×10^9	2.70×10^9	8.34×10^8
$f_4(x)$	CFS	**0**	7.88×10^{-14}	1.63×10^{-14}	2.32×10^{-14}
	FA	9.36×10^{-10}	0.0336	0.0082	0.0106
	ABC	**0**	**0**	**0**	**0**
	FPA	2.38×10^{-5}	0.0069	0.0012	0.0019
	CS	2.37×10^{-8}	2.39×10^{-4}	3.22×10^{-5}	7.18×10^{-5}
	BFP	2.19×10^{-2}	4.86×10^{-1}	3.33×10^{-1}	1.36×10^{-1}

Bold values in the table correspond to the best algorithmic values.

Table 6. P-test values of simulated algorithms for unimodal functions (50 Dimension).

Objective Function	FA	FPA	CS	ABC	CFS
$f_1(x)$	6.79×10^{-8}	**NA**	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}
$f_2(x)$	**NA**	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}
$f_3(x)$	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{v8}	**NA**
$f_4(x)$	8.00×10^{-9}	8.00×10^{-9}	8.00×10^{-9}	**NA**	**NA**

Table 7. Results comparison for unimodal functions (100 Dimension).

Objective Function	Algorithm	Best	Worst	Mean	Standard Deviation
$f_1(x)$	CFS	-2.78×10^4	-2.36×10^4	-2.60×10^4	$\mathbf{1.07 \times 10^3}$
	FA	-1.54×10^4	-5.88×10^3	-9.40×10^3	3.05×10^3
	ABC	-2.28×10^4	-1.73×10^4	-1.98×10^4	1.45×10^3
	FPA	$\mathbf{-1.63 \times 10^{19}}$	$\mathbf{-1.24 \times 10^{16}}$	$\mathbf{-1.50 \times 10^{18}}$	3.85×10^{18}
	CS	-1.05×10^4	-9.41×10^3	-1.00×10^4	$\mathbf{2.79 \times 10^2}$
	BFP	-5.75×10^8	-4.56×10^3	-5.87×10^7	1.73×10^8
$f_2(x)$	CFS	32.0745	$\mathbf{1.01 \times 10^2}$	69.1336	**19.4049**
	FA	**14.1504**	1.69×10^2	**55.1173**	40.4882
	ABC	5.70×10^3	1.88×10^4	1.23×10^4	3.88×10^3
	FPA	3.03×10^4	9.66×10^4	5.99×10^4	1.93×10^4
	CS	1.38×10^4	2.45×10^4	1.69×10^4	2.59×10^3
	BFP	1.61×10^5	3.16×10^5	2.53×10^5	4.57×10^4

Table 7. *Cont.*

Objective Function	Algorithm	Best	Worst	Mean	Standard Deviation
$f_3(x)$	CFS	$\mathbf{6.22 \times 10^3}$	$\mathbf{3.48 \times 10^4}$	$\mathbf{2.11 \times 10^4}$	$\mathbf{6.55 \times 10^3}$
	FA	1.89×10^6	1.10×10^7	5.29×10^6	2.83×10^6
	ABC	2.57×10^8	1.69×10^9	1.03×10^9	3.57×10^8
	FPA	1.71×10^8	4.66×10^8	3.22×10^8	8.91×10^7
	CS	1.26×10^6	6.12×10^6	2.69×10^6	1.06×10^6
	BFP	1.92×10^9	4.22×10^9	2.87×10^8	2.87×10^9
$f_4(x)$	CFS	$\mathbf{2.22 \times 10^{-16}}$	2.83×10^{-13}	2.49×10^{-14}	$\mathbf{6.35 \times 10^{-14}}$
	FA	1.36×10^{-11}	0.0667	0.0121	0.0164
	ABC	**0**	**0**	**0**	**0**
	FPA	1.34×10^{-6}	0.0028	4.55×10^{-4}	7.03×10^{-4}
	CS	1.80×10^{-8}	6.88×10^{-5}	1.43×10^{-5}	2.11×10^{-5}
	BFP	3.10×10^{-2}	4.92×10^{-1}	3.30×10^{-1}	1.42×10^{-1}

Bold values in the table correspond to the best algorithmic values.

Table 8. P-test values of simulated algorithms for unimodal functions (100 Dimension).

Objective Function	FA	FPA	CS	ABC	CFS
$f_1(x)$	6.79×10^{-8}	**NA**	6.79×10^{-8}	6.79×10^{-08}	6.79×10^{-8}
$f_2(x)$	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-08}	**NA**
$f_3(x)$	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-08}	**NA**
$f_4(x)$	8.00×10^{-9}	8.00×10^{-9}	8.00×10^{-9}	**NA**	7.97×10^{-9}

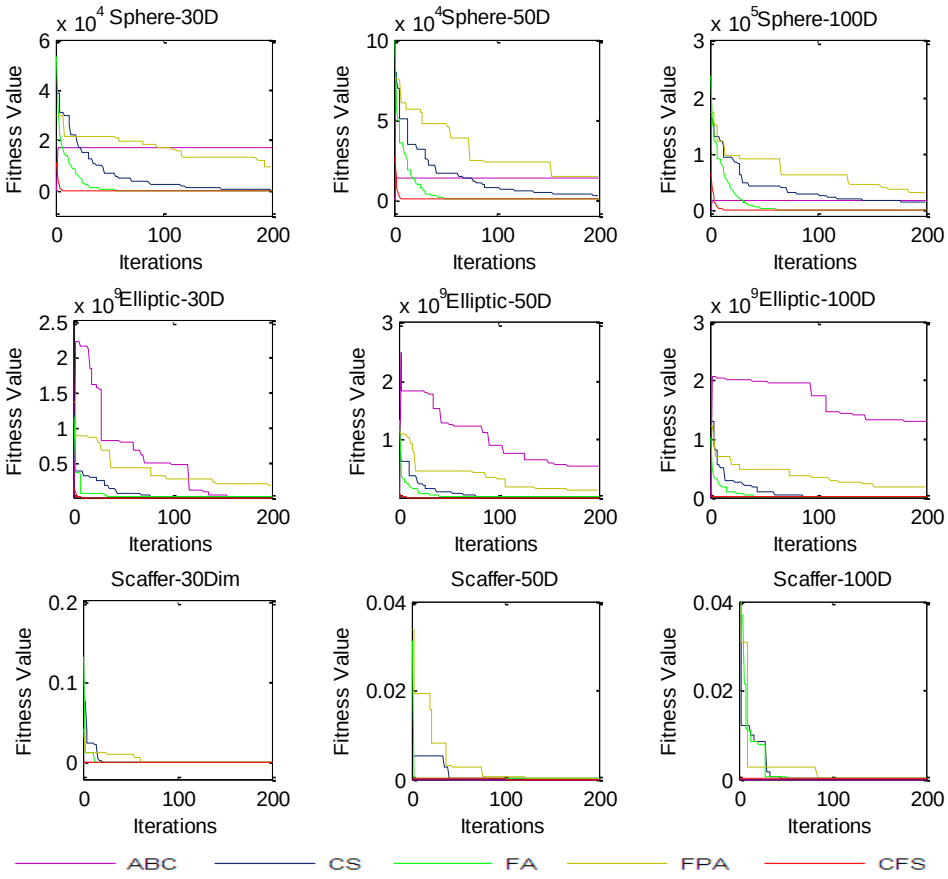

Figure 3. Convergence curves for unimodal functions.

5.1.2. Multimodal Functions

Multimodal benchmark functions feature several local minima that grow in number exponentially with dimension. As such, they are good for testing an algorithm's ability to avoid local minima. The CFS algorithm has been applied to six multimodal benchmark problems, with three-dimension sets (30, 50, and 100), as shown in Table 9. The algorithm has been compared to the ABC, FA, FPA, BFP, and CS algorithms. For 30 D and 50 D problems, with the f_5, f_6, and f_7 functions, the CFS algorithm performs better, while only the FA algorithm performs better than CFS with the f_8, f_9, and f_{10} functions, as shown in Tables 10–13, respectively. For 100 D (Tables 14 and 15), the CFS algorithm performs better for the f_5, f_6, f_7, and f_9 functions. for f_8 and f_{10}, the FA algorithm performs better. The rank-sum tests from Tables 11, 13 and 15 shows that the performance of the CFS algorithm is better statistically. The convergence characteristics are shown in Figure 4.

Table 9. Description of multimodal test problems.

Multimodal Test Problems	Objective Function	Search Range	Optimum Value	D
Rastrigin function	$f_5(x) = 10D + \sum_{i=1}^{D}\left[x_i^2 - 10\cos(2\pi x_i)\right]$	$[-5.12, 5.12]$	0	30, 50, 100
Weierstrass function	$f_6(x) = \sum_{i=1}^{D}\sum_{k=0}^{kmax}\left[a^k\cos\left(2\pi b^k(x_i+0.5)\right)\right] - D\sum_{k=0}^{kmax}\left[a^k\cos\left(2\pi b^k\cdot 0.5\right)\right]$; where $a = 0.5$, $b = 3$, $kmax = 20$	$[-0.5, 0.5]$	0	30, 50, 100
Griewank	$f_7 = \frac{1}{4000}\sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600, 600]$	0	30, 50, 100
Penalized 1 Function	$f_8 = \frac{\pi}{n}\{10\sin(\pi y_1) + \sum_{i=1}^{n-1}(y_i-1)2[1+10\sin2(\pi y_{i+1}) + (y_n-1)2\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$; $y_i = 1 + \frac{x_{i+1}}{4}$; $u(x_i, a, k, m) = \begin{cases} k(x_i-a)^m x_i > a \\ 0 \quad -a < x_i < a \\ k(-x_i-a)^m \quad x_i < -a \end{cases}$	$[-50, 50]$	0	30, 50, 100
Penalized 2 function	$f_9 = 0.1\{(3\pi x_1) + \sum_{i=1}^{n}(x_i-1)^2[1+sin^2(3\pi x_i+1)] + (x_n-1)^2[1+sin^2(2\pi x_n)]\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$; $u(x_i, a, k, m) = \begin{cases} k(x_i-a)^m x_i > a \\ 0 \quad -a < x_i < a \\ k(-x_i-a)^m \quad x_i < -a \end{cases}$	$[-50, 50]$	0	30, 50, 100
Ackley function	$f_{10}(x) = -20\exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}) - \exp(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)) + 20 + e$	$[-100, 100]$	0	30, 50, 100

Table 10. Results comparison for multimodal functions (30 Dimension).

Objective Function	Algorithm	Best	Worst	Mean	Standard Deviation
$f_5(x)$	CFS	$\mathbf{7.53 \times 10^{-13}}$	$\mathbf{3.59 \times 10^{-9}}$	$\mathbf{7.46 \times 10^{-10}}$	$\mathbf{9.59 \times 10^{-10}}$
	FA	2.07×10^{-9}	0.339	0.0226	0.0765
	ABC	7.24×10^{2}	4.17×10^{3}	2.06×10^{3}	9.69×10^{2}
	FPA	0.0017	0.2469	0.0595	0.063
	CS	8.00×10^{2}	1.86×10^{3}	1.16×10^{3}	3.00×10^{2}
	BFP	4.00×10^{-3}	1.70×10^{1}	8.13×10	3.16×10
$f_6(x)$	CFS	**1.9004**	**2.6108**	**2.3049**	**0.2186**
	FA	13.279	21.3048	16.8613	1.8416
	ABC	11.1346	19.6264	15.5206	2.4429
	FPA	35.7517	39.4474	37.5697	1.2557
	CS	16.6273	23.8924	19.9146	1.9027
	BFP	42.4496	50.6708	46.8925	2.1102

Table 10. *Cont.*

Objective Function	Algorithm	Best	Worst	Mean	Standard Deviation
$f_7(x)$	CFS	$\mathbf{1.31 \times 10^{-13}}$	$\mathbf{8.61 \times 10^{-11}}$	$\mathbf{2.18 \times 10^{-11}}$	$\mathbf{2.44 \times 10^{-11}}$
	FA	2.25×10^{-7}	1.48×10^{-5}	4.01×10^{-6}	3.54×10^{-6}
	ABC	3.5295	72.961	28.0289	16.7278
	FPA	1.50×10^{-4}	0.0874	0.0161	0.0231
	CS	4.5803	18.3641	8.8762	3.1545
	BFP	7.5279	1.65×10^{2}	7.69×10^{1}	4.51×10^{1}
$f_8(x)$	CFS	0.187	4.4737	0.5345	0.9324
	FA	**0.0013**	**0.133**	**0.0167**	**0.0287**
	ABC	8.04×10^{6}	1.68×10^{8}	6.93×10^{7}	4.39×10^{7}
	FPA	5.42×10^{5}	3.37×10^{7}	8.71×10^{6}	8.63×10^{6}
	CS	9.846	81.0669	24.5174	15.5959
	BFP	3.63×10^{8}	9.02×00^{8}	5.82×10^{8}	1.64×10^{8}
$f_9(x)$	CFS	**0.0086**	**0.0873**	**0.0286**	**0.0015**
	FA	**0.0059**	**0.0619**	**0.0119**	**0.0031**
	ABC	2.41×10^{7}	3.19×10^{8}	1.49×10^{8}	8.95×10^{7}
	FPA	1.38×10^{7}	1.09×10^{8}	4.99×10^{7}	2.45×10^{7}
	CS	51.2308	1.83×10^{5}	3.13×10^{4}	5.18×10^{4}
	BFP	4.59×10^{8}	1.65×10^{9}	1.13×10^{9}	3.41×10^{8}
$f_{10}(x)$	CFS	**0.5159**	**0.879**	**0.6915**	**0.0111**
	FA	**0.1462**	**0.469**	**0.261**	**0.0745**
	ABC	6.0052	14.9839	11.0513	2.6699
	FPA	14.0678	19.0195	17.4088	1.0373
	CS	10.0393	17.1319	13.0073	1.6329
	BFP	19.8877	20.849	20.5093	0.2645

Bold values in the table correspond to the best algorithmic values.

Table 11. P-test values of various algorithms for multimodal functions (30 Dimension).

Objective Function	FA	FPA	CS	ABC	BFP	CFS
$f_5(x)$	1.23×10^{-7}	1.23×10^{-7}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	**NA**
$f_6(x)$	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	**NA**
$f_7(x)$	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	**NA**
$f_8(x)$	**NA**	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}
$f_9(x)$	**NA**	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	$\mathbf{7.57 \times 10^{-4}}$
$f_{10}(x)$	**NA**	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	**NA**

Table 12. Results comparison for multimodal functions (50 Dimension).

Objective Function	Algorithm	Best	Worst	Mean	Standard Deviation
$f_5(x)$	CFS	$\mathbf{7.88 \times 10^{-10}}$	$\mathbf{3.30 \times 10^{-9}}$	$\mathbf{7.39 \times 10^{-10}}$	$\mathbf{9.49 \times 10^{-10}}$
	FA	1.85 × 10−09	0.1989	0.0109	0.0444
	ABC	7.08 × 1003	2.62 × 1004	1.74 × 1004	6.25 × 1003
	FPA	0.0083	0.323	0.1085	0.1053
	CS	3.08 × 1003	5.85 × 1003	4.34 × 1003	8.13 × 1002
	BFP	1.51 × 1000	1.98 × 1001	1.09 × 1001	5.08 × 1000
$f_6(x)$	CFS	**4.4288**	**6.3445**	**5.6194**	**0.5645**
	FA	28.8712	39.2514	33.3655	3.1566
	ABC	32.506	46.6375	38.085	3.2999
	FPA	66.9832	74.2188	70.8866	2.0856
	CS	33.8792	46.2211	39.4868	3.1594
	BFP	68.1743	89.0548	80.9581	5.9281

Table 12. *Cont.*

Objective Function	Algorithm	Best	Worst	Mean	Standard Deviation
$f_7(x)$	CFS	$\mathbf{9.55 \times 10^{-14}}$	$\mathbf{3.91 \times 10^{-10}}$	$\mathbf{4.20 \times 10^{-11}}$	$\mathbf{9.14 \times 10^{-11}}$
	FA	1.31×10^{-9}	0.0038	1.91×10^{-4}	8.44×10^{-4}
	ABC	45.2242	3.06×10^{2}	1.84×10^{2}	67.741
	FPA	0.0017	0.0634	0.0144	0.0174
	CS	21.4091	67.0806	38.1323	9.8239
	BFP	1.5237	1.77×10^{2}	8.07×10^{1}	6.33×10^{1}
$f_8(x)$	CFS	1.4169	11.1903	4.2718	2.5936
	FA	**0.0061**	**2.0945**	**0.4204**	**0.5428**
	ABC	8.05×10^{6}	1.07×18	5.58×10^{7}	2.92×10^{7}
	FPA	4.61×10^{5}	9.02×100^{7}	2.58×10^{7}	2.07×10^{7}
	CS	65.7835	8.56×10^{5}	6.28×10^{4}	1.90×10^{5}
	BFP	2.73×10^{8}	1.33×10^{9}	9.55×10^{8}	3.23×10^{8}
$f_9(x)$	CFS	0.0809	1.446	0.4249	**0.0559**
	FA	**0.0075**	**0.3539**	**0.0444**	**0.0817**
	ABC	8.40×10^{7}	2.33×10^{8}	1.52×10^{8}	4.62×10^{7}
	FPA	3.80×10^{7}	3.59×10^{8}	1.24×10^{8}	7.85×10^{7}
	CS	1.47×10^{5}	1.09×10^{6}	5.76×10^{5}	2.86×10^{5}
	BFP	4.59×10^{8}	1.65×10^{9}	1.13×10^{9}	3.41×10^{9}
$f_{10}(x)$	CFS	2.3662	18.0276	9.7138	4.4605
	FA	**0.3074**	**0.8458**	**0.3074**	**0.1439**
	ABC	10.5978	18.3258	15.8443	1.9489
	FPA	16.287	18.9147	17.6333	0.7004
	CS	10.3972	17.8387	13.9978	1.9681
	BFP	19.5794	20.8868	20.6361	0.3188

Bold values in the table correspond to the best algorithmic values.

Table 13. P-test values of various algorithms for multimodal functions (50 Dimension).

Objective Function	FA	FPA	CS	ABC	BFP	CFS
$f_5(x)$	1.06×10^{-7}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	**NA**
$f_6(x)$	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	**NA**
$f_7(x)$	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	**NA**
$f_8(x)$	**NA**	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}
$f_9(x)$	**NA**	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	9.12×10^{-7}
$f_{10}(x)$	**NA**	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}

Table 14. Results comparison for multimodal functions (100 Dimension).

Objective Function	Algorithm	Best	Worst	Mean	Standard Deviation
$f_5(x)$	CFS	$\mathbf{3.21 \times 10^{-10}}$	$\mathbf{2.21 \times 10^{-9}}$	$\mathbf{7.12 \times 10^{-10}}$	$\mathbf{7.73 \times 10^{-10}}$
	FA	1.49×10^{-8}	1.91×10^{-6}	3.98×10^{-7}	5.25×10^{-7}
	ABC	8.09×10^{4}	1.34×10^{5}	1.05×10^{5}	1.44×10^{4}
	FPA	0.0033	0.2715	0.0813	0.0755
	CS	1.22×10^{4}	2.03×10^{4}	1.65×10^{4}	2.37×10^{3}
	BFP	2.10×10^{-3}	1.49×10^{1}	8.27×10^{00}	4.11×10^{00}
$f_6(x)$	CFS	**16.4301**	**22.9819**	**18.3616**	**1.4303**
	FA	67.1686	81.6839	74.2381	4.1565
	ABC	1.03×10^{2}	1.25×10^{2}	1.16×10^{2}	6.363
	FPA	1.23×10^{2}	1.62×10^{2}	1.53×10^{2}	9.3169
	CS	81.7266	96.5011	89.2076	4.8649
	BFP	1.47×10^{2}	1.86×10^{2}	1.72×10^{2}	1.03×10^{1}

Table 14. *Cont.*

Objective Function	Algorithm	Best	Worst	Mean	Standard Deviation
$f_7(x)$	CFS	$\mathbf{9.20 \times 10^{-14}}$	$\mathbf{3.20 \times 10^{-10}}$	$\mathbf{4.46 \times 10^{-11}}$	$\mathbf{7.82 \times 10^{-11}}$
	FA	1.11×10^{-6}	6.13×10^{-6}	1.83×10^{-6}	1.65×10^{-6}
	ABC	6.68×10^{2}	1.08×10^{3}	8.83×10^{2}	1.15×10^{2}
	FPA	0.003	0.071	0.022	0.0212
	CS	1.06×10^{2}	1.91×10^{2}	1.41×10^{2}	22.8388
	BFP	1.0051	1.60×10^{2}	5.98×10^{1}	4.05×10^{1}
$f_8(x)$	CFS	18.9008	1.86×10^{2}	50.7091	35.0491
	FA	**9.0521**	**48.0274**	**28.2233**	**10.0771**
	ABC	3.42×10^{6}	7.72×10^{7}	3.44×10^{7}	1.94×10^{7}
	FPA	3.35×10^{7}	1.78×10^{7}	8.73×10^{7}	4.64×10^{7}
	CS	2.67×10^{4}	4.32×10^{6}	7.91×10^{5}	9.75×10^{5}
	BFP	7.01×10^{8}	3.61×10^{9}	2.49×10^{9}	8.69×10^{8}
$f_9(x)$	CFS	**1.7343**	**5.7693**	**3.5451**	**1.1998**
	FA	2.3704	9.5091	4.5693	1.5113
	ABC	3.89×10^{7}	2.01×10^{8}	1.07×10^{8}	4.86×10^{8}
	FPA	5.12×10^{7}	7.27×10^{8}	3.17×10^{8}	2.03×10^{8}
	CS	2.86×10^{6}	2.04×10^{7}	7.63×10^{6}	5.15×10^{6}
	BFP	1.64×10^{9}	6.02×10^{9}	4.75×10^{9}	1.40×10^{9}
$f_{10}(x)$	CFS	4.5948	19.4525	12.4022	3.9196
	FA	**1.3412**	**3.5494**	**2.7283**	**0.5434**
	ABC	18.2683	19.7601	19.1378	0.3621
	FPA	16.6528	19.686	18.2081	0.8273
	CS	13.5684	17.8497	15.6218	1.4927
	BFP	20.0389	21.0637	20.7694	0.2923

Bold values in the table correspond to the best algorithmic values.

Table 15. P-test values of various algorithms for multimodal functions (100 Dimension).

Objective Function	FA	FPA	CS	ABC		CFS
$f_5(x)$	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	**NA**
$f_6(x)$	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	**NA**
$f_7(x)$	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	**NA**
$f_8(x)$	**NA**	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}
$f_9(x)$	0.0439	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	**NA**
$f_{10}(x)$	**NA**	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}

5.1.3. Fixed Dimension Functions

Fixed dimension benchmark functions have finite dimensional space. The CFS algorithm has been applied to nine benchmark functions, as shown in Table 16, and the results have been compared to the ABC, CS, FA, BFP, and FPA algorithms. It can be seen from Tables 17 and 18 that the CFS algorithm performs better than the other algorithms for all the test problems. For functions $f_{14}, f_{15}, f_{16}, f_{17}$, and f_{18}, the ABC, FA, FPA, and CS algorithms are not able to achieve the global optimum. For the remaining algorithms, even if the global optimum is met, the CFS algorithm shows superior consistency because of its better standard deviation. The CFS algorithm's results are also statistically better, as shown in Table 18. The convergence curves are shown in Figure 5.

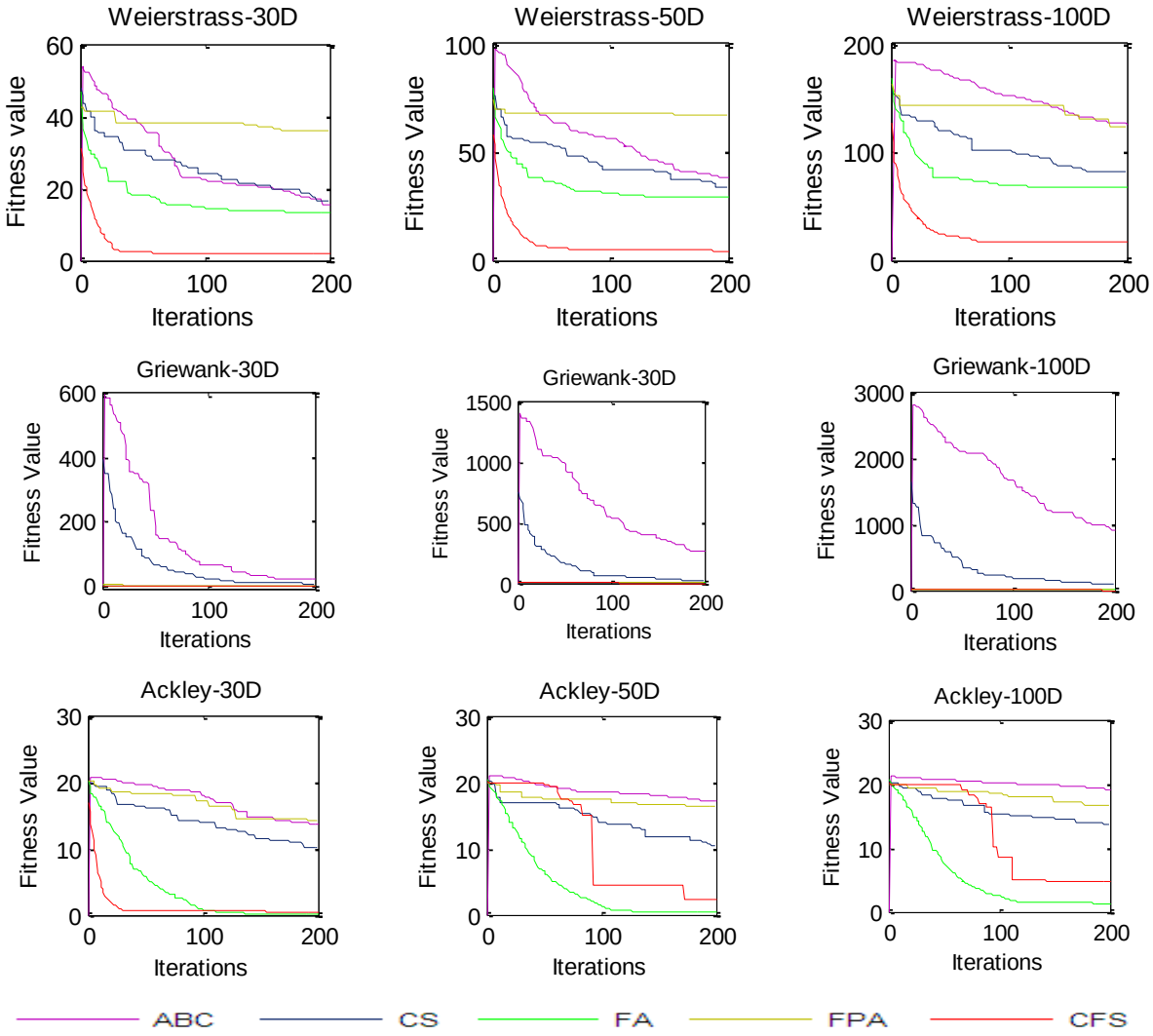

Figure 4. Convergence curves for multimodal functions.

Table 16. Description of fixed dimension test functions.

Fixed Dimension Test Problems	Objective Function	Search Range	Optimum Value	D
Branin RCOS Function	$f_{11}(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)cosx_1 + 10$	$x_1 \epsilon [-5, 10]$, $x_2 \epsilon [0, 15]$	0.397887	2
Six Hump Camel function	$f_{12}(x) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right)x_1^2 + x_1x_2 + \left(-4 + 4x_2^2\right)x_2^2$	$[-5, 5]$	-1.0316	2
Goldstein & Price function	$f_{13}(x) = (1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3{x_1}^2 - 14$ $x_2 + 6x_1x_2 + 3{x_2}^2))\,(30 + (2x_1 - 3x_2)^2(18 - 32x_1$ $+12{x_1}^2 + 48x_2 - 36x_1x_2 + 27{x_2}^2))$	$[-2, 2]$	3	2
Hartmann function 3	$f_{14}(x) = -\sum_{i=1}^{4} \alpha_i \exp[-\sum_{j=1}^{3} A_{ij}(x_j - P_{ij})^2]$	$[0, 1]$	-3.86278	3
Hartmann function 6	$f_{15}(x) = -\sum_{i=1}^{4} \alpha_i \exp[-\sum_{j=1}^{6} A_{ij}(x_j - P_{ij})^2]$	$[0, 1]$	-3.32237	6
Shekel 5	$f_{16}(x) = -\sum_{j=1}^{5} [\sum_{i=1}^{4}((x_i - C_{ij})^2 + \beta_j)^{-1}]$	$[0, 10]$	-10.1532	4
Shekel 7	$f_{17}(x) = -\sum_{j=1}^{7} [\sum_{i=1}^{4}((x_i - C_{ij})^2 + \beta_j)^{-1}]$	$[0, 10]$	-10.4029	4
Shekel 10	$f_{18}(x) = -\sum_{j=1}^{10} [\sum_{i=1}^{4}((x_i - C_{ij})^2 + \beta_j)^{-1}]$	$[0, 10]$	-10.5364	4
Easom function	$f_{19}(x) = -cosx_1 cosx_2 e^{(-(x_1-\pi)^2-(x_2-\pi)^2)}$	$[-10, 10]$	-1	2

Table 17. Results comparison for fixed dimension functions.

Objective Function	Algorithm	Best	Worst	Mean	Standard Deviation
$f_{11}(x)$	CFS	**0.3979**	**0.3979**	**0.3979**	$\mathbf{2.19 \times 10^{-11}}$
	FA	0.3979	0.3979	0.3979	1.30×10^{-8}
	ABC	0	0	0	0
	FPA	0.3979	0.3983	0.398	9.64×10^{-5}
	CS	0.3979	0.3979	0.3979	5.32×10^{-8}
	BFP	0.4416	5.3576	3.0721	1.63×10^{00}
$f_{12}(x)$	CFS	**−1.0316**	**−1.0316**	**−1.0316**	$\mathbf{1.66 \times 10^{-10}}$
	FA	−1.3016	−1.3015	−1.0316	3.47×10^{-5}
	ABC	−1.3016	−1.0250	−1.0310	0.0015
	FPA	−1.3016	−1.3016	−1.3016	1.24×10^{-5}
	CS	−1.3016	−1.0316	−1.3016	$\mathbf{8.22 \times 10^{-11}}$
	BFP	−0.9884	4.4587	0.1862	1.50×10^{00}
$f_{13}(x)$	CFS	**3**	**3**	**3**	$\mathbf{1.54 \times 10^{-12}}$
	FA	3	3	3	1.51×10^{-7}
	ABC	3.0004	3.0531	3.0107	0.0148
	FPA	3	3.0015	3.0004	4.73×10^{-4}
	CS	3	3	3	9.86×10^{-9}
	BFP	3.3525	98.258	47.458	3.46×10^{1}
$f_{14}(x)$	CFS	**−3.8628**	**−3.8628**	**−3.8628**	$\mathbf{6.56 \times 10^{-12}}$
	FA	−3.8628	−2.1968	−3.3064	0.6077
	ABC	−3.8628	−3.8621	−3.8626	2.17×10^{-4}
	FPA	−3.8325	−1.5171	−3.3253	0.6709
	CS	−3.8628	−3.8628	−3.8628	1.13×10^{-8}
	BFP	−0.5359	−3.25E−6	−0.0814	1.65×10^{-1}
$f_{15}(x)$	CFS	**−3.3224**	**−3.3224**	**−3.3224**	$\mathbf{3.74 \times 10^{-7}}$
	FA	−3.3224	−3.0639	−3.2469	9.32×10^{-2}
	ABC	−3.3223	−3.1954	−3.2461	0.059
	FPA	−3.2275	−2.9663	−3.1345	0.0702
	CS	−3.3223	−3.3140	−3.3201	0.0028
	BFP	−2.6298	−0.7595	−1.6330	0.5693
$f_{16}(x)$	CFS	**−10.1532**	**−10.1532**	**−10.1532**	$\mathbf{5.74 \times 10^{-5}}$
	FA	−5.0552	−5.0552	−5.0552	1.03×10^{-8}
	ABC	−10.1486	−2.6075	−5.5322	3.4454
	FPA	−5.0546	−5.0419	−5.0513	0.0033
	CS	−10.0826	−9.3309	−10.0826	0.1799
	BFP	−3.9584	−1.2893	−2.3915	0.8119
$f_{17}(x)$	CFS	**−10.4029**	**−10.4029**	**−10.4029**	$\mathbf{1.33 \times 10^{-4}}$
	FA	−5.0877	−5.0877	−5.0877	9.13×10^{-9}
	ABC	−10.5359	−2.4206	−5.3332	3.1817
	FPA	−5.0864	−5.0771	−5.0837	0.0025
	CS	−10.5358	−73868	−10.3006	0.6974
	BFP	−4.4980	−1.6336	−2.6498	0.8739
$f_{18}(x)$	CFS	**−10.5364**	**−10.5364**	**−10.5364**	$\mathbf{1.87 \times 10^{-6}}$
	FA	−5.1285	−5.1285	−5.1285	9.16×10^{-9}
	ABC	−10.4895	−1.8556	−4.6289	3.0032
	FPA	−5.1279	−5.1185	−5.1244	0.0028
	CS	−10.5357	−9.8686	−10.4320	0.1724
	BFP	−4.3369	−1.6523	−2.5939	0.7823
$f_{19}(x)$	CFS	**−1**	**−1.0000**	**−1.0000**	$\mathbf{6.07 \times 10^{-14}}$
	FA	−1.0000	−1.0000	−1.0000	1.26×10^{-8}
	ABC	−1.0000	−0.9886	−0.9977	0.0029
	FPA	−1.0000	−0.9998	−0.9999	6.79×10^{-5}
	CS	−1.0000	−1.0000	−1.0000	5.30×10^{-10}
	BFP	−0.5894	-2.18×10^{-13}	−0.0574	1.51×10^{-1}

Bold values in the table correspond to the best algorithmic values.

Table 18. P-test values of various algorithms for fixed dimension functions.

Objective Function	FA	FPA	CS	ABC	BFP	CFS
$f_{11}(x)$	7.89×10^{-8}	7.89×10^{-8}	9.17×10^{-8}	8.00×10^{-9}	6.79×10^{-8}	**NA**
$f_{12}(x)$	6.79×10^{-8}	6.79×10^{-8}	0.0679	6.79×10^{-8}	6.79×10^{-8}	**NA**
$f_{13}(x)$	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	**NA**
$f_{14}(x)$	6.79×10^{-8}	6.79×10^{-8}	7.89×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	**NA**
$f_{15}(x)$	0.1895	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	**NA**
$f_{16}(x)$	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	**NA**
$f_{17}(x)$	6.79×10^{-8}	6.79×10^{-8}	0.0012	1.60×10^{-4}	6.79×10^{-8}	**NA**
$f_{18}(x)$	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	**NA**
$f_{19}(x)$	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	6.79×10^{-8}	**NA**

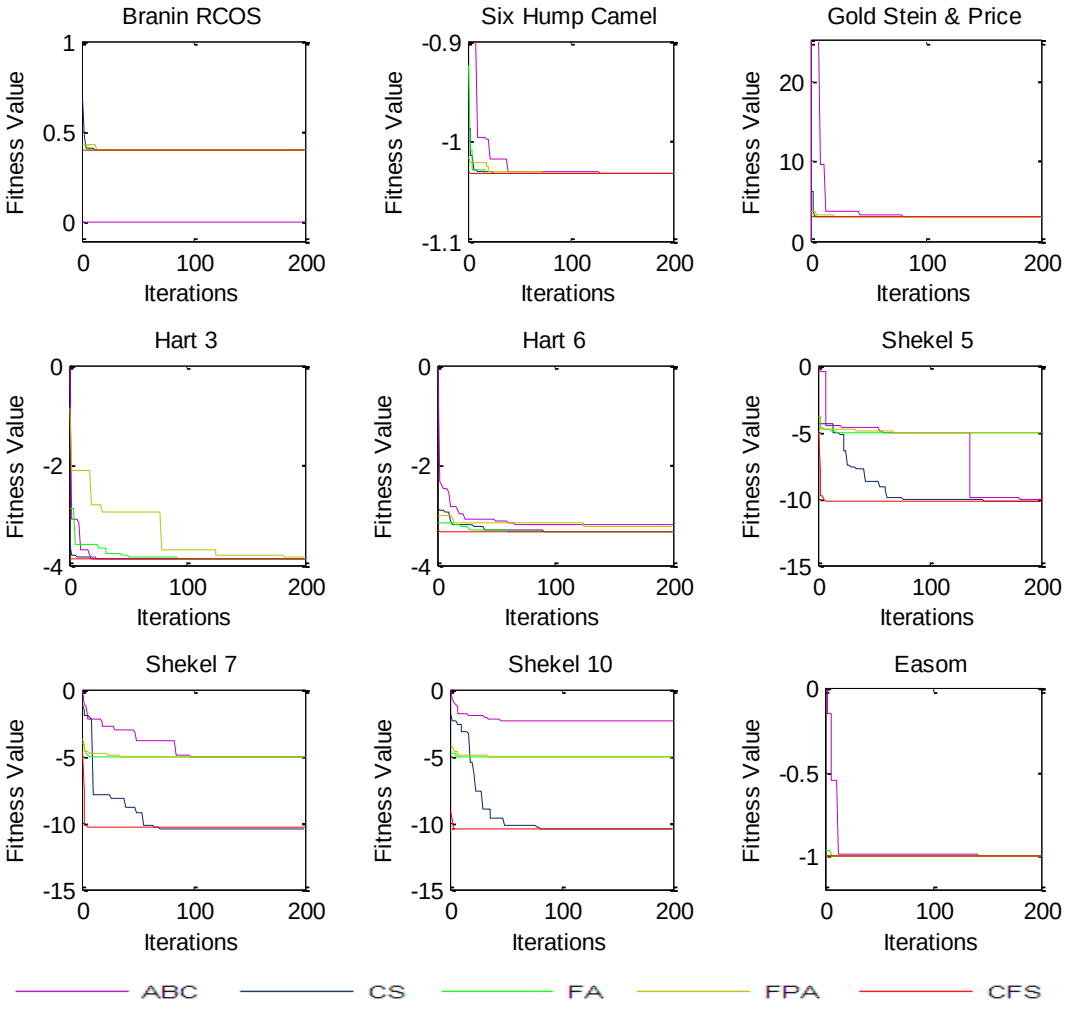

Figure 5. Convergence curves for fixed dimension problems.

5.2. FNN–MLP Datasets

The proposed CFS algorithm was used to train FNN-MLP datasets. The standard benchmark FNN–MLP data sets were obtained from the Machine Learning Repository of the University of California, Irvine [53]. The datasets used are: Breast Cancer, XOR, Balloon, Iris, and Heart. The results are compared with the PSO, GA, ES, ACO, GWO, PLIB [30,54–57] algorithms, and the Whale Optimization Algorithm (WOA) [58] and the Moth Flame Optimization (MFO) [59] for verification. The optimization parameter settings for the CFS algorithm are presented in Table 19. Table 20 details the specifications of the datasets used for comparison. The simplest dataset is the 3-bit XOR; it has three

attributes and eight training/test samples. for the Balloon dataset, there are four attributes and 16 training/test samples. For the Iris dataset, there are 150 training/test samples with four attributes. For the Breast Cancer dataset, the highest number of 100 test samples, 599 training samples, and nine attributes are used. for the Heart dataset, there are 80 training samples, 187 test samples, and 22 attributes. The number of classes for each dataset is two, except for Iris, which is set to three. These datasets are highly complex sets of problems and are employed to test the performance of the CFS algorithm. The total number of runs for the CFS algorithm is set to 10; this is the same as used in study [30]. The number of function evaluations for the XOR and Balloon datasets is 50 × 250 = 12,500 for all the algorithms. For the Iris, Breast Cancer, and Heart datasets, the number of function evaluations is 20 × 250 = 5000 for the CFS algorithm and 200 × 250 = 50,000 for the rest. The results are presented in the form of an average of 10 runs and their standard deviation is obtained from the best MSEs in the last iteration of the CFS algorithm. The best results are those with the lowest average and standard deviation, ultimately indicating the better performance of the proposed approach [60,61].

$$X^t = \frac{(x-a)\times(d-c)}{(b-a)} + c \tag{14}$$

Table 19. Parameters for algorithms.

Algorithm	Parameters	Value
CFS	Population size	50 for XOR and Balloon; 20 for the rest
	Probability switch	0.8
	Discovery rate of alien egg (p_a)	0.25
	Maximum number of iterations	250

Table 20. Classification datasets.

Classification Datasets	Attributes Count	Training Samples Count	Test Samples Count	Number of Classes
3-bit XOR	3	8	8 as training samples	2
Balloon	4	16	16 as training samples	2
Iris	4	150	150 as training samples	3
Breast Cancer	9	599	100	2
Heart	22	80	187	2

The number of hidden nodes for N number of inputs of datasets is kept constant and is given by $2 \times N + 1$. The structure for each MLP is given in Table 21.

Table 21. MLP structure for each dataset.

Classification Datasets	Attributes Count	MLP Structure
3-bit XOR	3	3−7−1
Balloon	4	4−9−1
Iris	4	4−9−3
Breast Cancer	9	9−19−1
Heart	22	22−45−1

5.2.1. XOR Dataset

This dataset returns XOR of input as output. It has three inputs, eight training/test samples, and one output. This dataset has a dimension of 36 with range of [−10, 10], with an MLP structure of 3−7−1. The results, in term of average and standard deviation, are given in Table 22. It can be seen in Table 4 that the performance evaluation of the CFS-MLP algorithm is far better than all other algorithms tested.

Table 22. Comparison results of CFS-MLP for XOR dataset.

Algorithm	Average	Standard Deviation
CFS−MLP	$\mathbf{9.687 \times 10^{-12}}$	$\mathbf{2.520 \times 10^{-11}}$
GWO−MLP	9.410×10^{-3}	2.950×10^{-1}
PSO−MLP	8.405×10^{-2}	3.594×10^{-2}
GA−MLP	1.810×10^{-4}	4.130×10^{-4}
ACO−MLP	1.803×10^{-1}	2.526×10^{-2}
ES−MLP	1.187×10^{-1}	1.157×10^{-2}
PBIL−MLP	3.022×10^{-2}	3.966×10^{-2}
WOA−MLP	8.420×10^{-2}	5.140×10^{-2}
MFO−MLP	5.298×10^{-6}	1.038×10^{-5}

Bold values in the table correspond to the best algorithmic values.

5.2.2. Balloon Dataset

The Balloon dataset has a dimension of 55, with range of [−10, 10]. This dataset has 18 training/test samples, with four attributes and two classes, with an MLP structure of 4−9−1. The results are given in Table 23. The results show that the CFS algorithm gives far higher average and standard deviation values when compared to the GWO, PSO, GA, ACO, ES, PBIL, WOA, and MFO algorithms.

Table 23. Comparison results of CFS-MLP for the Balloon dataset.

Algorithm	Average	Standard Deviation
CFS−MLP	$\mathbf{1.19 \times 10^{-41}}$	$\mathbf{1.90 \times 10^{-41}}$
GWO−MLP	9.38×10^{-15}	2.81×10^{-14}
PSO−MLP	0.000585	0.000749
GA−MLP	5.08×10^{-24}	1.06E−23
ACO−MLP	0.004854	0.00776
ES−MLP	0.019055	0.17026
PBIL−MLP	2.49×10^{-5}	5.27×10^{-5}
WOA−MLP	4.88×10^{-6}	1.41×10^{-5}
MFO−MLP	1.85×10^{-15}	6.18×10^{-15}

Bold values in the table correspond to the best algorithmic values.

5.2.3. Iris Dataset

The Iris dataset has 75 variables to be optimized in the range of [−10, 10]. It has 150 training/test samples, with four attributes and two classes. The MLP structure of 4−9−3 is utilized to solve this dataset. The results are shown in Table 24. For the Iris dataset, the results of the CFS algorithm are competitive with GWO in terms of average; for standard deviation, the results of the CFS algorithm are superior.

Table 24. Comparison results of CFS-MLP for the Iris dataset.

Algorithm	Average	Standard Deviation
CFS−MLP	0.06673	$\mathbf{5.31 \times 10^{-4}}$
GWO−MLP	**0.0229**	0.0032
PSO−MLP	0.22868	0.057235
GA−MLP	0.089912	0.123638
ACO−MLP	0.405979	0.053775
ES−MLP	0.31434	0.052142
PBIL−MLP	0.116067	0.036355
WOA−MLP	0.734134	0.051808
MFO−MLP	0.667957	0.003467

Bold values in the table correspond to the best algorithmic values.

5.2.4. Breast Cancer Dataset

This is a challenging dataset, with 100 test samples, 599 training samples, nine attributes, and two classes. It has 209 dimensions, with an MLP structure of 9−19−1. The outcomes of this dataset are given in Table 25. The results show that the CFS algorithm is far superior than the PSO, GA, ACO, PBIL, WOA, and MFO algorithms. When compared to GWO, they are highly competitive.

Table 25. Comparison results of CFS-MLP for the Breast Cancer dataset.

Algorithm	Average	Standard Deviation
CFS−MLP	**0.0018**	$\mathbf{2.83 \times 10^{-4}}$
GWO−MLP	**0.0012**	$\mathbf{7.44 \times 10^{-5}}$
PSO−MLP	0.034881	0.002472
GA−MLP	0.003026	0.0015
ACO−MLP	0.01351	0.002137
ES−MLP	0.04032	0.00247
PBIL−MLP	0.032009	0.003065
WOA−MLP	0.006243	0.003128
MFO−MLP	0.004038	0.003041

Bold values in the table correspond to the best algorithmic values.

5.2.5. Heart Dataset

This is the last dataset used in this paper and was solved with an MLP structure of 22−45−1. It has 187 test samples, 80 training samples, 22 attributes, and two classes. The results are shown in Table 26. The Heart dataset is a very challenging dataset, with a 1081 dimension. The CFS algorithm performs better for this dataset when compared to others.

Table 26. Comparison results of CFS-MLP for the Heart dataset.

Algorithm	Average	Standard Deviation
CFS−MLP	**0.0686**	**0.0067**
GWO−MLP	0.1226	0.0077
PSO−MLP	0.188568	0.008939
GA−MLP	0.093047	0.02246
ACO−MLP	0.22843	0.004979
ES−MLP	0.192473	0.015174
PBIL−MLP	0.154096	0.018204
WOA−MLP	0.179664	0.052152
MFO−MLP	0.08321	0.02062

Bold values in the table correspond to the best algorithmic values.

5.3. Discussion of Results

The results comparison of the CFS algorithm with the ABC, CS, FA, FPA, and BFP algorithms show that, for test function, the CFS algorithm delivers very competitive results for unimodal and multimodal benchmark problems. For fixed dimension problems, no algorithm among ABC, CS, FA, FPA, and BFP are comparable. This occurs as a result of the inability of these algorithms to emerge from local minima. The ABC algorithm has the problem of becoming stuck in local minima, while the CS and FPA algorithms are inconsistent due to their inability to emerge from local minima and are, hence, inconsistent. In its initial stage, the FA algorithm has slower convergence because of random distribution and as a result of its insufficiency in exploring ability. At the last stage, fireflies gather around the optimal solution but, due to random motion and attractiveness, there can be flight mistake and hence the solution converges very slowly.

The results of the CFS-MLP clearly show that, for the XOR and Balloon datasets, there are same number of function evaluations and that the results are bothbetter and significant.

For the Iris, Breast Cancer, and Heart datasets, the minimum number of function evaluations for CFS algorithms is 5000, while for others it is 50,000. Hence, it can be said that the CFS algorithm is able to achieve a more significant result with fewer function evaluations. This proves the superiority of the CFS algorithm over the GWO, PSO, GA, ACO, ES, PBIL, WOA, and MFO algorithms.

In the CFS algorithm, there are two search agents: cuckoos and flower pollinators. When cuckoos are not able to find the optimal solution, flower pollinators help them; in turn, the flower pollinators are helped by cuckoos when stuck in a local optimum. Therefore, two solutions (one from cuckoos and the other from flower pollinators) are generated. The final solution is the best among the two. This helps the CFS algorithm to achieve faster convergence and consistency in finding the optimal solution.

6. Conclusions

In this work, a new CFS algorithm was proposed for MLP training. The algorithm was first tested over 19 standard benchmark functions and their results were statistically compared with the ABC, CS, FA, FPA, and BFP algorithms. The results demonstrate that the CFS algorithm perform significantly better, with higher consistency, in avoiding local minima when compared to the ABC, CS, FA, FPA, and BFP algorithms. The CFS algorithm was then used to train MLPs; five datasets were used. The results of the CFS-MLP were compared, in terms of average and standard deviation, with the GWO, PSO, GA, ACO, ES, PBIL, WOA, and MFO algorithms. The experimental results again proved the superiority of the CFS algorithm for MLPs.

Despite this, the proposed algorithm has certain drawbacks. Because of the stochastic nature of the algorithm, the algorithm is inefficient for several kinds of problems. As the CFS algorithm uses two general equations for finding new solutions, the computational complexity of the algorithm increases. Thus, it is imperative to find new and prospective solutions to deal with the complexity problem. As a future direction of study, the parameters of the algorithm can be exploited to find the best set of parameters. In addition, the CFS algorithm can be applied to various other domains, including clustering, antenna array synthesis, feature selection, medical imaging, segmentation, and others. Apart from that, the proposed algorithm can be extended to multi-objective, hyper-parameters, and other dimensions.

Author Contributions: Conceptualization, R.S.; methodology, R.S.; software, N.M.; validation, R.S., N.M. and V.M.; formal analysis, V.M.; investigation, R.S.; resources, N.M.; data curation, V.M.; writing—original draft preparation, R.S.; writing—review and editing, R.S, and N.M.; visualization, R.S.; supervision, R.S.; project administration, R.S. and N.M.; All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data will be made available on request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. McCulloch, W.S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **1943**, *5*, 115–133. [CrossRef]
2. Kohonen, T. The self-organizing map. *Proc. IEEE* **1990**, *78*, 1464–1480. [CrossRef]
3. Dorffner, G. Neural networks for time series processing. *Neural Netw. World* **1996**.
4. Ghosh-Dastidar, S.; Adeli, H. Spiking neural networks. *Int. J. Neural Syst.* **2009**, *19*, 295–308. [CrossRef] [PubMed]
5. Bebis, G.; Georgiopoulos, M. Feed-forward neural networks. *IEEE Potentials* **1994**, *13*, 27–31. [CrossRef]
6. Rosenblatt, F. *The Perceptron, A Perceiving and Recognizing Automaton Project Para*; Cornell Aeronautical Laboratory: Buffalo, NY, USA, 1957.
7. Werbos, P. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Ph.D. Thesis, Harvard University, Cambridge, MA, USA, 1974.

8. Reed, R.D.; Marks, R.J. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*; MIT Press: Cambridge, MA, USA, 1998.
9. Caruana, R.; Niculescu-Mizil, A. An empirical comparison of supervised learning algorithms. In Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, PA, USA, 25–29 June 2006; pp. 161–168.
10. Hinton, G.E.; Sejnowski, T.J. *Unsupervised Learning: Foundations of Neural Computation*; MIT Press: Cambridge, MA, USA, 1999.
11. Wang, D. *Unsupervised Learning: Foundations of Neural Computation*; MIT Press: Cambridge, MA, USA, 2001; p. 101.
12. Hertz, J. *Introduction to the Theory of Neural Computation. Basic Books 1*; Taylor Francis: Abingdon, UK, 1991.
13. Wang, G.-G.; Guo, L.; Gandomi, A.H.; Hao, G.-S.; Wang, H. Chaotic krill herd algorithm. *Inf. Sci.* **2014**, *274*, 17–34. [CrossRef]
14. Wang, G.-G.; Gandomi, A.H.; Alavi, A.H.; Hao, G.-S. Hybrid krill herd algorithm with differential evolution for global numerical optimization. *Neural Comput. Appl.* **2013**, *25*, 297–308. [CrossRef]
15. Van Laarhoven, P.J.; Aarts, E.H. *Simulated Annealing*; Springer: Berlin/Heidelberg, Germany, 1987.
16. Szu, H.; Hartley, R. Fast simulated annealing. *Phys. Lett. A* **1987**, *122*, 157–162. [CrossRef]
17. Mitchell, M.; Holland, J.H.; Forrest, S. When will a genetic algorithm outperform hill climbing? *NIPS* **1993**, 51–58.
18. Sanju, P. Enhancing Intrusion Detection in IoT Systems: A Hybrid Metaheuristics-Deep Learning Approach with Ensemble of Recurrent Neural Networks. *J. Eng. Res.* 2023; *in press*.
19. Mirjalili, S.; Mohd Hashim, S.Z.; Moradian Sardroudi, H. Training feedforward neural networks using hybrid particle swarm optimization and gravitational search algorithm. *Appl. Math. Comput.* **2012**, *218*, 11125–11137. [CrossRef]
20. Whitley, D.; Starkweather, T.; Bogart, C. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Comput.* **1990**, *14*, 347–361. [CrossRef]
21. Shokouhifar, A.; Shokouhifar, M.; Sabbaghian, M.; Soltanian-Zadeh, H. Swarm intelligence empowered three-stage ensemble deep learning for arm volume measurement in patients with lymphedema. *Biomed. Signal Process. Control.* **2023**, *85*, 105027. [CrossRef]
22. Socha, K.; Blum, C. An ant colony optimization algorithm for continuous optimization: Application to feed-forward neural network training. *Neural Comput. Appl.* **2007**, *16*, 235–247. [CrossRef]
23. Ozturk, C.; Karaboga, D. Hybrid Artificial Bee Colony algorithm for neural network training. In Proceedings of the 2011 IEEE Congress on, Evolutionary Computation (CEC), New Orleans, LA, USA, 5–8 June 2011; pp. 84–88.
24. Mendes, R.; Cortez, P.; Rocha, M.; Neves, J. Particle swarms for feed forward neural network training. In Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290), Honolulu, HI, USA, 12–17 May 2002.
25. Gudise, V.G.; Venayagamoorthy, G.K. Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In Proceedings of the Swarm Intelligence Symposium, SIS'03, Indianapolis, IN, USA, 26 April 2003; pp. 110–117.
26. Ilonen, J.; Kamarainen, J.-K.; Lampinen, J. Differential evolution training algorithm for feed-forward neural networks. *Neural Process. Lett.* **2003**, *17*, 93–105. [CrossRef]
27. Uzlu, E.; Kankal, M.; Akpınar, A.; Dede, T. Estimates of energy consumption in Turkey using neural networks with the teaching–learning-based optimization algorithm. *Energy* **2014**, *75*, 295–303. [CrossRef]
28. Moallem, P.; Razmjooy, N. A multi-layer perceptron neural network trained by invasive weed optimization for potato color image segmentation. *Trends Appl. Sci. Res.* **2012**, *7*, 445–455. [CrossRef]
29. Darekar, R.V.; Chavan, M.; Sharanyaa, S.; Ranjan, N.M. A hybrid meta-heuristic ensemble based classification technique speech emotion recognition. *Adv. Eng. Softw.* **2023**, *180*, 103412. [CrossRef]
30. Mirjalili, S. How effective is the Grey Wolf Optimizer in training multi-layer perceptrons. *Appl. Intell.* **2015**, *43*, 150–161. [CrossRef]
31. Yang, X.-S.; Deb, S. Engineering optimization by cuckoo search. *Int. J. Math. Model. Numer. Optim.* **2010**, *1*, 330–343.
32. Yang, X.-S. Flower Pollination Algorithm for Global Optimization. In Proceedings of the 11th International Conference, UCNC 2012, Orléan, France, 3–7 September 2012; Volume 7445, pp. 240–249. [CrossRef]
33. Fine, T.L. *Feedforward Neural Network Methodology*; Springer: Berlin/Heidelberg, Germany, 1999.
34. Mirjalili, S.; Sadiq, A.S. Magnetic optimization algorithm for training multi-layer perceptron. In Proceedings of the Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference, Xi'an, China, 27–29 May 2011; pp. 42–46.
35. Payne, R.B.; Sorenson, M.D.; Klitz, K. *The Cuckoos*; Oxford University Press: Oxford, UK, 2005.
36. Barthelemy, P.; Bertolotti, J.; Wiersma, D.S. A Lévy flight for light. *Nature* **2008**, *453*, 495–498. [CrossRef]
37. Yang, X.-S.; Deb, S. Cuckoo Search via Levy Flights'. In Proceedings of the 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC), Coimbatore, India, 9–11 December 2009; IEEE Publications: Piscataway, NJ, USA, 2009.
38. Brown, C.; Liebovitch, L.S.; Glendon, R. Lévy Flights in Dobe Ju/'hoansi Foraging Patterns. *Human Ecol.* **2007**, *35*, 129–138. [CrossRef]
39. Pavlyukevich, I. Cooling down Lévy flights. *J. Phys. A Math. Theory* **2007**, *40*, 12299–12313. [CrossRef]
40. Walker, M. How Flowers Conquered the World, BBC Earth News, 10 July 2009. Available online: http://news.bbc.co.uk/earth/hi/earth_news/newsid_8143000/8143095.stm (accessed on 1 January 2019).
41. Waser, N.M. Flower constancy: Definition, cause and measurement. *Am. Nat.* **1986**, *127*, 596–603. [CrossRef]
42. Glover, B.J. *Understanding Flowers and Flowering: An Integrated Approach*; Oxford University Press: Oxford, UK, 2007.
43. Xin-She, Y.; Karamanoglu, M.; He, X. Flower pollination algorithm: A novel approach for multiobjective optimization. *Eng. Optim.* **2014**, *46*, 1222–1237.

44. Belew, R.K.; McInerney, J.; Schraudolph, N.N. *Evolving Networks: Using the Genetic Algorithm with Connectionist Learning*; Cognitive Computer Science Research Group: La Jolla, CA, USA, 1990.
45. Smizuta, T.; Sato, D.; Lao, M.; Ikeda, T. Shimizu, Structure design of neural networks using genetic algorithms. *Complex Syst.* **2001**, *13*, 161–176.
46. Yu, J.; Wang, S.; Xi, L. Evolving artificial neural networks using an improved PSO and DPSO. *Neurocomputing* **2008**, *71*, 1054–1060. [CrossRef]
47. Leung, F.H.; Lam, H.; Ling, S.; Tam, P.K.S. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Trans. Neural Netw.* **2003**, *14*, 79–88. [CrossRef]
48. Montana, D.J.; Davis, L. Training Feedforward Neural Networks Using Genetic Algorithms. *IJCAI* **1989**, *89*, 762–767.
49. Derrac, J.; García, S.; Molina, D.; Herrera, F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm Evolut. Comput.* **2011**, *1*, 3–18. [CrossRef]
50. Karaboga, D. *An Idea Based on Honey Bee Swarm for Numerical Optimization*; Technical Report TR-06; Erciyes University, Engineering Faculty, Computer Engineering Department: Kayseri, Turkey, 2005.
51. Yang, X.S. Firefly algorithms for multimodal optimization. In *Stochastic Algorithms: Foundations and Applications*; Lecture Notes in Computer Sciences; SAGA: Chicago, IL, USA, 2009; Volume 5792, pp. 169–178.
52. Urvinder, S.; Salgotra, R. Synthesis of linear antenna array using flower pollination algorithm. *Neural Comput. Appl.* **2016**, *29*, 435–445.
53. Blake, C.; Merz, C.J. *{UCI} Repository of Machine Learning Databases*; UCI: Aigle, Switzerland, 1998.
54. Beyer, H.-G.; Schwefel, H.-P. Evolution strategies—A comprehensive introduction. *Nat. Comput.* **2002**, *1*, 3–52. [CrossRef]
55. Yao, X.; Liu, Y.; Lin, G. Evolutionary programming made faster. *Evol. Comput. IEEE Trans.* **1999**, *3*, 82–102.
56. Yao, X.; Liu, Y. Fast evolution strategies. In Proceedings of the Evolutionary Programming VI, Indianapolis, IN, USA, 13–16 April 1997; pp. 149–161.
57. Baluja, S. *Population-Based Incremental Learning: A Method for Integrating Genetic Search-Based Function Optimization and Competitive Learning*; DTIC Document; Carnegie Mellon University: Pittsburgh, PA, USA, 1994.
58. Seyedali, M.; Lewis, A. The whale optimization algorithm. *Adv. Eng. Softw.* **2016**, *95*, 51–67.
59. Seyedali, M. Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. *Knowl. Based Syst.* **2015**, *89*, 228–249.
60. Zhou, Y.; Niu, Y.; Luo, Q.; Jiang, M. Teaching learning-based whale optimization algorithm for multi-layer perceptron neural network training. *Math. Biosci. Eng.* **2020**, *17*, 5987–6025. [CrossRef]
61. Chong, H.Y.; Yap, H.J.; Tan, S.C.; Yap, K.S.; Wong, S.Y. Advances of metaheuristic algorithms in training neural networks for industrial applications. *Soft Comput.* **2021**, *25*, 11209–11233. [CrossRef]