IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received August 10, 2020, accepted August 20, 2020. Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2020.3020356

An Improved Grasshopper Optimization Algorithm Based Echo State Network for Predicting Faults in Airplane Engines

ABUBAKAR BALA[1,2], (Member, IEEE), IDRIS ISMAIL[1], ROSDIAZLI IBRAHIM[1], (Senior Member, IEEE), SADIQ M. SAIT[3,4], (Senior Member, IEEE), AND DIEGO OLIVA[5,6], (Member, IEEE)

[1]Electrical and Electronics Engineering Department, Universiti Teknologi PETRONAS, Seri Iskandar 32610, Malaysia
[2]Electrical Engineering Department, Bayero University Kano, Kano 700241, Nigeria
[3]Computer Engineering Department, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia
[4]Center for Communications and IT Research, Research Institute, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia
[5]Departamento de Ciencias Computacionales, Universidad de Guadalajara, CUCEI, Guadalajara 44430, Mexico
[6]IN3—Computer Science Department, Universitat Oberta de Catalunya, 08860 Barcelona, Spain

Corresponding author: Abubakar Bala (abala.ele@buk.edu.ng)

This work was supported by the Universiti Teknologi PETRONAS (UTP), Malaysia under the YUTP and CGS grant. We also acknowledge support from King Fahd University of Petroleum & Minerals, Saudi Arabia.

ABSTRACT In today's age of industrialization, sensor devices installed on equipment generate a vast amount of data. One of the engineers' main jobs is utilizing these data to provide better solutions to industrial problems. This availability of extensive data partly led to the creation of predictive maintenance (PdM). In PdM, existing and previous conditions of devices are used to predict their future behavior for optimal maintenance. Most of these PdM approaches are typical time-series predictions. Machine learning tools like Recurrent Neural Networks (RNNs) are excellent tools for time-series predictions. However, most RNNs suffer from training issues due to the unstable gradient problem. Thus, networks such as the Echo State Network (ESN), were designed to solve them. The ESN solves the gradient problem by training only the output weights using simple linear regression. Despite this ease, the selection of ESN parameters and topology is a considerable design challenge. This problem is often formulated as a typical optimization problem. Metaheuristic algorithms are known to be excellent tools for solving optimization problems. Hence, in this work, we design an improved Grasshopper Optimization Algorithm (GOA) based ESN. The proposed technique uses a new solution representation with a simplified attraction and repulsion mechanisms to enhance performance. Our target application is to predict the Remaining Useful Life (RUL) of turbofan engines. The method outperforms the Cuckoo Search (CS), Differential Evolution (DE), Particle Swarm Optimization (PSO), Binary PSO (BPSO), the original GOA, the classical ESN, deep ESN, and LSTM. We have provided all implemented codes and data at the GitHub repository. https://github.com/bala-221/Airplane-fault-prediction

INDEX TERMS Artificial neural networks, airplanes, echo state network, evolutionary algorithms, grasshopper optimization algorithm (GOA), metaheuristics, maintenance, prediction, remaining useful life (RUL), turbofan engine.

I. INTRODUCTION

Complexities in engineering design and manufacturing, combined with changing operating and environmental conditions, have led to substantial reliability issues in the industry. To overcome these challenges, Prognostics and Health Management (PHM) was invented. PHM's primary aspect is ''prediction,'' which consists of the use of present and past component states of any dynamic system to estimate its future behavior. Engineers can use the prediction results for two purposes. Firstly, it serves as a reliable alarm system to stop machine efficiency degradation, machine malfunctioning,

The associate editor coordinating the review of this manuscript and approving it for publication was Yu Liu.

or even disastrous system failure [1]. Secondly, it can be employed to schedule maintenance appropriately, resulting in a significant reduction in down-times. Moreover, it also has potential maintenance cost savings since there is a shift from preventive to predictive maintenance [2].

Generally, there are two approaches to prognostics: model-based methods and data-driven methods [3]. In model-based methods, we require an understanding of the physics of the system. This information is then used to create mathematical formulae that explain the system's dynamics. We then use the equations to estimate the system's future health. However, for complex dynamic systems, it is often challenging to obtain a correct analytical model of the system, especially if the system is operating in noisy and unpredictable environments.

In contrast, data-driven methods employ the use of machine learning and pattern recognition to create a link between operation signals and the health state of systems. This is then used to detect changes in the system as well as predict future behavior. The classical data-driven techniques for non-linear system prediction are non-deterministic such as projection pursuit [4], multivariate adaptive regression splines [5], and, Autoregressive Integrated Moving Average (ARIMA) models [6]. These methods depend on state patterns of similar previous systems to forecast future states. They are more suitable when the general first principle of a system is not available, or when it is significantly complicated, that finding the accurate analytical model is extremely costly. Data-driven methods have shifted over the years towards the use of adaptive system models like fuzzy neural systems and neural networks [7]. These systems have provided better performance compared to classical prediction methods.

One of the powerful types of neural networks is Recurrent Neural Network (RNN). These networks have cyclic connections within them that loop back previous signals within the network. This architecture makes the RNNs particularly suitable for sequential data prediction. However, most RNNs have complicated training schedules and suffer from unstable gradient problem [8]. Gated networks such as the Long Short Term Memory (LSTMs) and Gated Recurrent Units (GRUs) have been efficiently used to solve the vanishing and exploding gradient problem of RNNs. Despite their extensive and successful application, they are more complex than the ESN in terms of architecture and implementation. The ESN solves the unstable gradient problem by applying the simple least square regression training of the output weight connections. Despite its simplicity, the ESN has been shown to perform as well or even better than LSTM and GRU on multivariate time-series prediction tasks [9]. This work is also compared with the deep LSTM network [10]. In the next section, we discuss the classical ESN.

A. ECHO STATE NETWORK

ESN is a relatively modern RNN, which substitutes a dynamic reservoir for the hidden layer of RNNs. Some of its unique characteristics include:

(i) The randomly generated connection of weights within the reservoir is used as an information processing unit.
(ii) The randomly generated reservoir serves as a feature space in which the inputs are mapped to.
(iii) Only its output weights require training. Other weights are often generated randomly.
(iv) Readout weight connections are often learned through a simple linear regression technique.

A schematic of the classical ESN is shown in Figure 1. It has 3 levels: input, reservoir, and a readout layer. Moreover, it usually has an input bias unit that is excluded here for clarity. General guidelines for designing a good ESN are provided in [11], [25]. They suggested the design of a random, large, and sparsely connected reservoir layer. Thus, the units within the ESN's reservoir could be hundreds to thousands depending on task complexities.

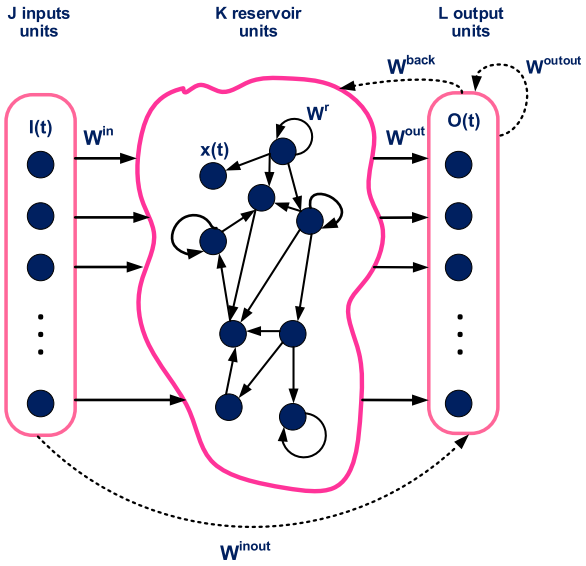


FIGURE 1. The ESN layout with dotted lines denoting optional weight connections [26].

The connection weights are mostly obtained via a uniform distribution with symmetry around zero. In this work, we use a discrete ESN with J inputs. These J input neurons receive inputs of $\mathbf{I}(t) = [\mathbf{I_1}(t), \mathbf{I_2}(t), \ldots \mathbf{I_J}(t)]^{\mathsf{T}}$ at each time interval (t). Additionally, it has K reservoir units depicted as: $\mathbf{x}(t) = [\mathbf{x_1}(\mathbf{t}), \mathbf{x_2}(\mathbf{t}), \ldots \mathbf{x_K}(\mathbf{t})]^{\mathsf{T}}$. Finally, it has L output units represented as $[\mathcal{O}_\mathbf{1}(t), \mathcal{O}_\mathbf{2}(t), \ldots \mathcal{O}_\mathbf{L}(t)]^{\mathsf{T}}$ that provide the prediction results. From Figure 1, the reservoir update equation is:

$$\mathbf{x}(t) = f(W^{in}\mathbf{I}(\mathbf{t}) + W^{r}\mathbf{x}(t-1) + W^{back}\mathcal{O}(t-1)) \quad (1)$$

Additionally, the output equation is:

$$\mathcal{O}(t) = g(W^{inout}\mathbf{I}(t) + W^{out}\mathbf{x}(t) + W^{outout}\mathcal{O}(t-1)) \quad (2)$$

where f and g are the input and output activation functions (AFs) respectively. While $W^{in}(J \times K)$ is the input weight matrix connections from the inputs to the reservoir, and $W^{r}(K \times K)$ is the reservoir connection weights. Moreover,

$W^{back}(L \times K)$ is the optional feedback weights from the output back to the reservoir. Additionally, other voluntary connections are: weights between inputs and output units $W^{inout}(J \times L)$, output cycle weights $W^{outout}(L \times L)$ and bias to output weights $W^{biasout}(1 \times L)$. As already stated, only the output weights require training, and all remaining weight connections are usually generated randomly and often stay constant throughout the ESN training. If W^{outGen} denotes all weight connections to the output, then (2) can be transformed to:

$$\mathcal{O}(t) = g\left(W^{outGen}\left[\mathbf{I}(t); \mathbf{x}(t); \mathcal{O}(t-1)\right]\right) \tag{3}$$

with [;;;] representing matrix concatenation. The main goal of the training is to find the W^{outGen}. If we assume that the g in (3) is invertible, then the equation may be formulated as:

$$Y = W^{outGen}\left[\mathbf{I}(t); \mathbf{x}(t); \mathcal{O}(t-1)\right] \tag{4}$$

with $Y = g^{-1}\mathcal{O}(t)$. We calculate the output weight (W^{outGen}) by employing the least square linear regression technique, which minimizes the variance between the target value and the network's actual output.

One important feature of the echo state network is the echo state property (ESP) [27]. ESP posits that the impact of previous states and initial inputs on future states should decrease over time and not continue or grow. The ESP is often ensured in practical applications if reservoir weight is scaled so that its spectral radius (SR), which is the highest absolute eigenvalue of W^r, is set to a magnitude of less than 1. However, several works have proven that since the inputs to the ESN also affect its ESP, it is possible to attain ESP even if SR is greater than one for certain inputs.

Summary of ESN training:

Main Inputs: An input/output sequence:
$[\mathbf{I}(1), \mathcal{O}(1); \mathbf{I}(2), \mathcal{O}(2); \ldots\ldots; \mathbf{I}(T), \mathcal{O}(T)]$
Main Output: Learned output weights (W^{outGen})

Step 1: The number of reservoir units K, spectral radius of the reservoir (SR), and the washout time (T_o) are initialized. The washout time is the number of training samples in which the reservoir activations are not collected but are discarded. It is required so that the impact of initialized random weights is reduced. After the initial parameters are set, the weights are randomly generated while ensuring ESP.

Step 2: Collection of reservoir activations:

- Setting ESN's zero states. i.e., $\mathcal{O}(0) = 0$ & $\mathbf{x}(0) = 0$, which are used to find $\mathbf{x}(1)$.
- Computing the reservoir states $\mathbf{x}(t)$ of the T time steps after washout using (1).
- The reservoir sates $\mathbf{x}(t)$, input $\mathbf{I}(t)$ and output $\mathcal{O}(t-1)$ for $t = T_o, \ldots, T$ are collected in a matrix called S with size $(J + K + L) \times (T - T_o + 1)$.

Step 3: The output weights are then computed by using the following definition from (4).

$$W^{outGen} = (S^{-1}Y_{desired})^{\intercal} \tag{5}$$

with $Y_{desired}$ denoting the desired output of the training data. The Moore-Penrose pseudoinverse method is used to find inverse of (5). This marks the end of the training, and the ESN is now ready for prediction by applying inputs and forecasting the outputs using (1) and (2). Moreover, as a means to reduce over-fitting, regularization may be applied to the output as:

$$W^{outGen} = (SS' + reg.I)^{-1}(S'Y_{desired}) \tag{6}$$

where reg represents the regularization term and I an identity matrix.

Since the ESN inception, different types have been proposed [26]. However, ESNs with leaky integrator neurons in their reservoir called Leaky Integrator ESNs (LI-ESN) have been proven to give impressive performance [28]. The leaky neurons perform a leaky integration of its activation from previous time steps. Thus, (1) is transformed to:

$$\mathbf{x}(t) = (1-a)\mathbf{x}(t-1) + af\left(W^{in}\mathbf{I}(t) + W^{r}\mathbf{x}(t-1) + W^{back}\mathcal{O}(t-1)\right) \tag{7}$$

with a as the leakage/decay rate with value often in the range [0, 1]. This value is set so that a neuron neither retains nor leaks more activations than it had. The leakage rate has been shown to control the ''velocity'' of the reservoir dynamics. Throughout this article, we use LI-ESN.

Despite its easy training, ESN parameter and topology selection is a significant challenge. Initial studies have provided guides on ESN designs [11]. However, these suggestions depend on the target application and are often fuzzy. Thus, more recently, many works have employed metaheuristics techniques to select the ESN's design parameters [12], [29].

In this work, we develop a grasshopper optimization algorithm (GOA) based method to optimize the ESN [13]. This algorithm has proven to have excellent performance compared with Particle Swarm Optimization (PSO), Differential Evolution (DE), and Genetic Algorithm (GA) on selected mathematical benchmark functions. We test our technique on fault prediction of turbofan engines. The main contributions of the work include.

- Development of a new solution representation for ESN optimization.
- Design of an improved Grasshopper Optimization Algorithm (GOA) with simplified attraction and repulsion schedules.
- Testing the optimized ESN on the remaining useful life (RUL) prediction of turbofan engines.

The remaining portions of the paper are as follows: Section II discuses a review of works related to this article. In Section III, the Grasshopper Optimization Algorithm is introduced. Section IV, outlines the methodology of our research. The RUL prediction of the turbofan engine is explained in Section V. While Section VI discusses the results of our study. In Section VII, we discuss some of the challenges of ESN. Finally, Section VIII gives concluding remarks.

II. RELATED WORK

This section briefly discusses some selected works that use metaheuristics (MAs) to optimize the ESN. For a general review, we refer the reader to Bala *et al.* [12]. One of the pioneering works in the application of MA in ESN optimization is Ferreira and Ludermir [14]. They employed the famous GA to tune the ESN. In their further work [15], they proposed a new technique called RCDESIGN, that employs GA to select the appropriate ESN parameters and topology.

Additionally, Ma *et al.* [16] developed a deep ESN for time-series forecasting. The GA was used to tune the values of leaky rate, spectral radii, and reservoir sizes of the stack of ESNs within the deep ESN. Moreover, a GA optimized double reservoir ESN (DRESN) was developed by Zhong *et al.* [17]. The technique was better than the PSO optimized ESN for turbofan engine RUL estimation.

The PSO has also been successfully used to fine-tune the ESN. For instance, Wang and Yan [18] proposed a binary PSO (BPSO) for ESN optimization. They define the output weights connections as a feature binary selection problem. Subsequently, the BPSO was used to select appropriate output connections. In [19], a PSO based tuning of ESN was developed. The method selects a fraction of the reservoir weights and then tunes their values using PSO. They highlighted that the technique is less mathematical since the spectral radius does not have to be computed. Chouikhi *et al.* [20] extended the work in [19]. Here, together with the fraction of reservoir weights, a fraction of input weights and feedback weights are also chosen for the PSO optimization. The method outperformed the one in [19].

Additionally, Amaya and Alvares [21] proposed an ESN tuning based on an artificial bee colony algorithm. The method was tested on the RUL prediction of turbofan engines and was better than the classical ESN. Moreover, cuckoo search optimization was used to optimize the ESN [22], [23]. The method was found to outclass classical ESN and other methods. In recent research, Wang *et al.* [24] used the DE algorithm for ESN tuning. The method was found to outperform the GA optimized ESN on the prediction of electricity consumption. In the next section, we present the classical GOA algorithm.

III. THE GRASSHOPPER OPTIMIZATION ALGORITHM

The Grasshopper Optimization Algorithm (GOA) is a swarm-based metaheuristic developed in 2016 [30]. It copies the natural swarming culture of grasshoppers and locusts [31]. Even though grasshoppers can be found as isolated individuals, they are mostly organized in one of the fascinating swarms available in nature. One exciting thing about these swarms is that they can exist in both the larva and adult phases of the grasshopper development. The larva swarm is characterized by slow and little steps of movements. In contrast, adult swarms consist of long steps of movement with abrupt jumps. The authors mimicked this behavior to fit the exploitation and exploration techniques required to form most metaheuristics. Thus, the exploration is like the adult swarm movement, and the exploitation copies the larva swarms [30]. They modeled the location of a grasshopper within the swarm as [32]:

$$X_i = S_i + G_i + A_i \tag{8}$$

where X_i represents the location of the i^{th} grasshopper. S_i is the social relationship of grasshopper i with other grasshoppers within the swarm. A_i is the wind advection affecting the i^{th} grasshopper. Finally, G_i is the gravitational pull on grasshopper i. The main engine of the GOA is the social interaction between the grasshoppers (S_i) given as [30]:

$$S_i = \sum_{j=1, j\neq i}^{M} \mathcal{Z}(d_{ij})\hat{d}_{ij} \tag{9}$$

where d_{ij} is the distance between grasshopper i and j given as: $d_{ij} = |x_j - x_i|$. The $\mathcal{Z}$ is a function representing the power of social forces on grasshopper i. While $\hat{d}_{i,j} = \dfrac{X_j - X_i}{d_{ij}}$ is a unit vector from the i^{th} to the j^{th} grasshopper. The $\mathcal{Z}$ function is given as [32]:

$$\mathcal{Z}(d) = f\exp(-d/l) - \exp(-d) \tag{10}$$

with f denoting the attraction strength and l, the attractive length scale. The $\mathcal{Z}$ function elicits forces of attraction and repulsion between the grasshoppers.

A plot of the function is shown in Figure 2. One can observe from the figure that when the distance is in the range [0, 2.079], the grasshoppers/agents repel each other to avoid impact. However, there is no attraction nor repulsion when the difference is exactly at 2.079, and agents are said to be in the comfort zone. As the distance extends beyond 2.079, the function keeps increasing till the distance reaches a value of around 4. This range [2.079, 4] is called the *attraction phases*. Here, the grasshoppers cooperate to reach the food source. Different values of f and l would give variant zones of repulsion, comfort, and attraction. However, for the GOA, the values of $f = 0.5$ and $l = 1.5$ are used. Despite the excellent modeling of the different zones by (10), it gives a value of almost zero when the distance goes beyond 10, (Figure 2). Thus, to solve this issue, the distances between the agents are projected to the range [1, 4].

A schematic of the different agent movement zone is further given in Figure 3. From the figure, when the distance of a grasshopper and the target (a grasshopper at food source) is less than the comfort zone, the repulsion force on the grasshopper is greater than that of attraction. While, when the distance is at the comfort zone, there is an equal force of repulsion and attraction. As the grasshopper travels outside the comfort zone, the forces of attraction become more significant than the forces of repulsion.

Equation (8) is slightly modified. The gravity force is dropped. Moreover, the direction of wind is assumed to be towards the target. Thus, the new position of a grasshopper i

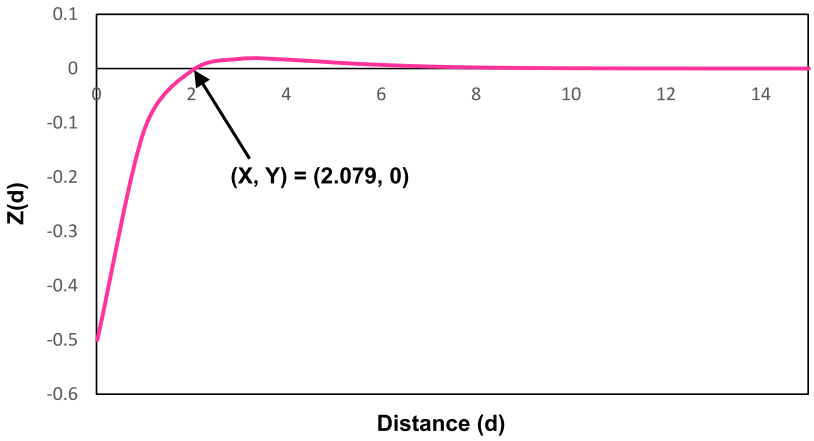


FIGURE 2. The $\mathcal{Z}$ function with $l = 1.5$ & $f = 0.5$ [13].

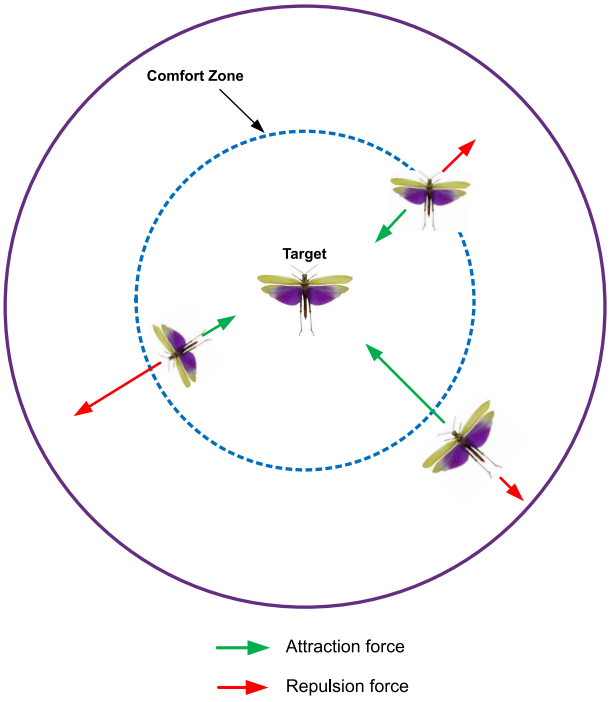


FIGURE 3. The forces of attraction and repulsion acting on a grasshopper [13].

is modeled as [30]:

$$X_i^d = c\left(\sum_{j=1, j\neq i}^{N} c\left(\frac{uB_d - lB_d}{2}\right)\mathcal{Z}\left(|x_j^d - x_i^d|\right)\frac{x_j - x_i}{d_{ij}}\right) + T_d \tag{11}$$

where lB_d and uB_d as the lower bound and upper bound values on the d^{th} dimension accordingly. While, T_d is the value of the d^{th} dimension in the target (best solution met). Moreover, c is a reducing coefficient set to minimize the repulsion, comfort, and attraction zones with iterations. Its value is computed as [30]:

$$c = c_{max} - l\left(\frac{c_{max} - c_{min}}{maxIter}\right) \tag{12}$$

with *maxIter* denoting the maximum iteration number and l the present iteration number. The constants are: $c_{min} = 0.00001$, $c_{max} = 1$.

The basic GOA is shown in Algorithm 1. It commences by initializing the algorithm constants like: the maximum iteration (*maxIter*), number of grasshoppers in the swarm (m), c_{max} and c_{min}. Next, the initial random population/swarm is generated. Subsequently, the fitness/cost of each agent within the population is found by using an evaluation function (line 3). The best agent within the swarm is set as target (T) in line 4. The main GOA iteration starts at line 5. Within the loop, the value of c is updated by using (12). Then, for each search agent, its distance from other agents is found and normalized into the range [1,4]. In line 9, the position of the present agent is updated by (11). If the new position extends beyond boundary conditions, they are brought back (line 10). The fitness of each new position is found. Then the best new position of the search agent is compared with that of the target. If it is better than the target, it replaces it. The algorithm keeps iterating till the maximum iteration count is reached and the target is returned.

IV. IMPROVED GOA ALGORITHM

The methods developed in this research are discussed here. Inspired by the grasshopper optimization algorithm (GOA) we develop a new hybrid version. We take motivation from many other metaheuristics to come up with this improved one.

The procedure is shown in Algorithm 2. It proceeds as follows: firstly, algorithm parameters such as maximum generation (*maxGen*), size of the population, and the number of children (*numChildren*) are set. Then, the population is initialized, the cost of each agent is found, and the best agent is saved. In line 7, we begin the main loop of the algorithm which will be repeated *maxGen* times. In this loop, the first

Algorithm 1 The GOA Algorithm [13]

1: Fix algorithm parameters
2: Fix the grasshopper population X_i $\{i = 1, 2, \ldots, m\}$
3: Find the cost of each grasshopper
4: $T \leftarrow$ best grasshopper
5: **for** $l = 1$ **to** *maxIter* **do**
6: Update c using (12)
7: **for each** X_i in the swarm **do**
8: Normalize the distances between the swarm agents in [1,4]
9: Update the position of the current agent using (11)
10: If new position goes out of search boundary, return it back
11: Find the fitness of new positions
12: **end for**
13: Update T if better solution is found
14: **end for**
15: Return the T as best agent.

task is to divide the population into two groups. The best 25% form the best group and the remaining 75% make up the bottom group. Subsequently, in lines 10-28, we create children from the initial population. In this for-loop, we perform a selection process like that of the genetic algorithm (GA). For the first parent called hopperOne, we run a *rand* function which generates a random number evenly distributed from 0 to 1. If the number generated is less than 0.7, we select the first parent (hopperOne) from a random position in the top group (line 12). Otherwise, we choose hopperOne from a random point in the bottom group in line 15. This would make the first parent more likely to come from the best group. However, for the second parent (hopperTwo), we reverse the case above, thus, making the second parent more probable to come from the bottom group lines 13 and 16.

Next, we find the absolute difference between the two parents (*diff*). For each of the genes in *diff* that is less than 3, we perform a repulsion. The repulsion is like the case where grasshoppers have come too close in flight, and need to repel each other to avoid collision. However, if the gene difference is greater than or equal to 7, we perform an attraction showing the case where the two grasshoppers are too far apart and may miss the source of food. Thus, they need to attract each other. The procedures depicting attraction and repulsion are given in Algorithm 3 and 4 respectively.

In the attraction Algorithm 3, for each of the genes were K_i is greater than 7, we calculate a ratio term as in line 2. We then add this ratio to the inferior parent's gene as in lines 4 or 6 and we return the child.

In contrast, Algorithm 4 shows the repulsion procedure and it proceeds as follows. For each gene where K_i is less than or equal to 3, we generate a *val* term. Next, we run a *randi*(2) function which gives a value of 1 or 2, so we randomly add this *val* term to the gene of parent one or two and then return the child.

We then return to Algorithm 2 line 25, where other genes of the child are obtained by randomly choosing from either

Algorithm 2 The Hybrid GOA Algorithm

1: Start algorithm parameters
2: Start the grasshopper swarm X_i $\{i = 1, 2, \ldots, m\}$
3: Find the cost of each grasshopper
4: $T \leftarrow$ best grasshopper
5: $goldenRatio \leftarrow 1.62$
6: $A \leftarrow 1$
7: **for** $gen = 1$ **to** *maxGen* **do**
8: $\alpha = A/(\sqrt{gen})$
9: Divide the population into top and bottom group
10: **for** $X_i = 1$ **to** *numChildren* **do**
11: **if** $rand < 0.7$ **then**
12: Select random top agent as hopperOne
13: Select random bottom agent as hopperTwo
14: **else**
15: Select random bottom agent as hopperOne
16: Select random top agent as hopperTwo
17: **end if**
18: diff = $abs(hopperOne - hopperTwo)$
19: **for each** K_i gene in diff **do**
20: **if** $K_i < 3$ **then**
21: Simulate repulsion
22: **else if** $K_i > 7$ **then**
23: Simulate attraction
24: **end if**
25: Get other genes by randomly choosing from hopperOne or hopperTwo
26: **end for**
27: Find the cost of new child and save it
28: **end for**
29: Merge parents and children and sort them by cost
30: 50% of next population come from best merge
31: Other 50% are chosen randomly from the remaining merged population
32: Update T, if better solution is found
33: **end for**
34: Return the T as best agent.

the gene of parent one or two. The new child is validated for out of bounds and then saved. This similar approach is done for all children. They are then merged with their parents and sorted. We then proceed to create the next generation. 50% of the subsequent generation of the population is selected from the fittest of the merged population as in line 30. While the other 50% are selected randomly from the remaining merged population. The best agent is updated if a better solution is found. The outer loop keeps repeating till the *maxGen* iterations are reached and we report the final best agent.

A. SOLUTION REPRESENTATION

Six parameters of the ESN are optimized: reservoir size, reservoir connectivity, leaky rate, spectral radius, regularization term, and, input scaling. However, the reader should note that for the input scaling (scaling of the input weight connections), all the 14 inputs (in the Commercial

Algorithm 3 The Attraction

1: **for each** K_i gene in diff where $K_i \geq 7$ **do**
2: $ratio = (K_i/goldenRatio)$
3: **if** Cost(hopperOne) > Cost(hopperTwo) **then**
4: child[i] = ratio + hopperOne[i]
5: **else**
6: child[i] = ratio + hopperTwo[i]
7: **end if**
8: **end for**
9: Return the child

Algorithm 4 The Repulsion

1: **for each** K_i gene in diff where $K_i \leq 3$ **do**
2: $val = 0.2 * (1 - rand)^{-\alpha}$
3: **if** $randi(2) == 1$ **then**
4: child[i] = val + hopperOne[i]
5: **else**
6: child[i] = val + hopperTwo[i]
7: **end if**
8: **end for**
9: Return the child

Modular Aero-Propulsion System Simulation data) and the bias connections are scaled differently. Thus, making the total number of parameters as 20. Moreover, each parameter range is divided into 10 parts as follows:

resSize = [100, 300, 450, 600, 750, 900, 1050, 1200, 1350, 1500]

resCon = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

spectRad = [0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0]

leaky = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

regTerm = [1E-9, 1E-8, 1E-7, 1E-6, 1E-5, 1E-4, 1E-3, 1E-2, 1E-1, 1.0]

inScaling = [0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0]

A typical solution/agent can be {1, 4, 6, 7, 8, 9, 10, 4, 8, 9, 2, 4, 5, 6, 7, 8, 1, 2, 1, 4}. We then use these values to index into the arrays of the parameters. Thus, this solution would be transformed into reservoir size = 100, reservoir connectivity = 0.3, spectral radius = 1.2, leaky rate = 0.7, etc. The usefulness of this unique solution representation may not be apparent here, until we discuss how we perform the repulsion and attraction in Section IV-D.

B. INITIAL POPULATION

To obtain the initial population, we select a random integer from 1 to 10 for each agent's gene. This is then repeated to get each initial member of the population.

C. OBJECTIVE FUNCTION

We adopt the mean squared error (MSE) of the validation data as cost function. Thus, for each agent, we set aside a part of the train data for validation. Then, we select parameters and train the ESN. The MSE of the trained ESN on the validation data is taken as the cost of the agent. The MSE cost function is shown in (13). For the input scaling, each column of the input weight matrix is normalized by dividing it with its absolute highest singular value and then scaled to the parameter. While, for the spectral radius, the reservoir matrix is normalized by dividing it by the absolute highest eigenvalue, before it is scaled to the spectral radius.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\mathcal{O}_i - \hat{\mathcal{O}}_i)^2 \tag{13}$$

where $\mathcal{O}$ and $\hat{\mathcal{O}}$ respectively as the desired and actual output.

D. CHILD CREATION

Assuming the procedure from lines 11 to 17 of Algorithm 2 gave us hopperOne as [10, 3, 10, 6, 3, 5, 7, 10, 9, 4, 5, 10, 8, 3, 5, 9, 6, 2, 5, 5] and hopperTwo as [8, 2, 7, 9, 6, 8, 2, 8, 3, 9, 8, 8, 4, 1, 7, 5, 7, 4, 2, 9]. We then proceed to find *diff* on line 18. The value of *diff* is [2, 1, 3, 3, 3, 3, 5, 2, 6, 5, 3, 2, 4, 2, 2, 4, 1, 2, 3, 4]. Since there is no value of *diff* that is greater than 7, we do not have an attraction in this case. We then proceed to perform repulsion for each value of *diff* that is ≤3. We generate the *val* term and add it to either gene of hopperOne or hopperTwo. The resulting child after repulsion is: [8.33, 2.20, 7.24, 9.29, 6.33, 9.33, 0, 8.44, 0, 0, 8.32, 8.25, 0, 1.23, 7.21, 0, 7.27, 4.23, 2.31, 0]. Subsequently, the zero genes of the child are taken randomly from either hopperOne or hopperTwo. The consequent child is: [8.33, 2.20, 7.24, 9.29, 6.33, 9.33, 7, 8.44, 9, 4, 8.32, 8.25, 8, 1.23, 7.21, 9, 7.27, 4.23, 2.31, 5]. Next, we round-off these values and fix out of bounds by setting the value of any gene above 10 as 10. The final child is [8, 2, 7, 9, 6, 9, 7, 8, 9, 4, 8, 8, 8, 1, 7, 9, 7, 4, 2, 5] and its cost is found as 705.01.

E. STOPPING CRITERION

The maximum iteration count which is set as 100 is the stopping condition adopted here.

V. TURBOFAN ENGINE DEGRADATION PREDICTION

This section explains the turbofan engine degradation prediction. Figure 4 shows a pictorial view of the prediction procedure. The main element of the method is the turbofan engine (shown in Figure 5). It is responsible for providing the driving thrust to most modern airplanes. Its main parts include a fan, low-pressure compressor (LPC), high-pressure compressor (HPC), low-pressure turbine (LPT), and high-pressure turbine (HPT).

The engine degradation dataset is obtained from the prognostic data center of the National Aeronautics and Space Administration (NASA) of the United States [33]. It is simulated with the NASA-developed Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) program. To produce the C-MAPSS data, identical turbofan engines were driven in different operating conditions, and then faults of varying magnitudes were inserted into each engine unit until the system failed. These faults were injected by varying the inputs of the C-MAPSS software. These inputs are presented in Table 1. As the inputs (faults) and operating

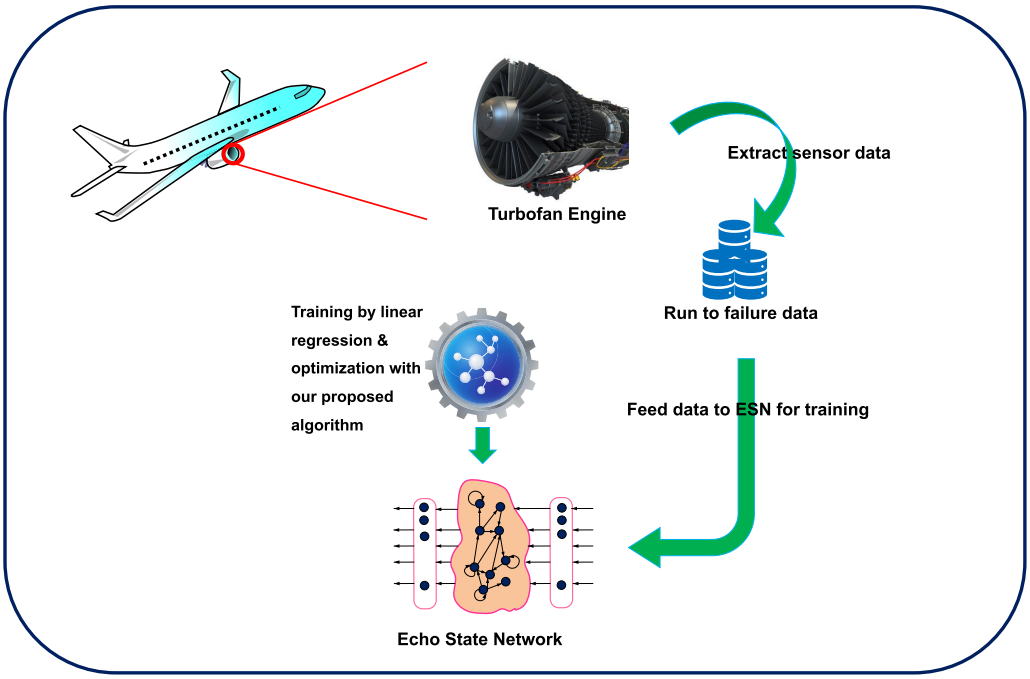


FIGURE 4. Firstly, we extract run-to-failure sensor signals from the turbofan engine. The data is then preprocessed (Filtering & Normalization), and fed to the echo state network (ESN) for training. In the linear regression training process, our proposed algorithm is used to select the right parameters for the network. After training, the ESN is now ready for deployment. In this mode, it can provide the predicted remaining useful life of an engine when it receives a certain sensor signal.

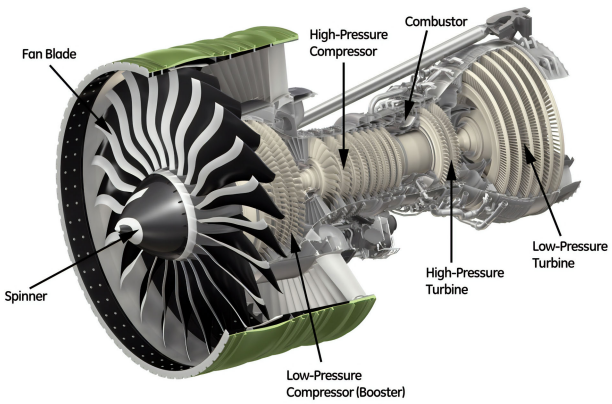


FIGURE 5. A turbofan engine with its different components [34].

conditions are varied, sensor output signals shown in Table 2 are measured at a regular frequency. The engine keeps running until it fails to meet the healthy state criterion determined by the health index parameters given at the bottom of Table 2. These health index parameters are represented by margins. First, they are normalized to the range [0, 1], where 1 indicates a healthy machine, and 0 denotes an engine with stall margin reduced to a defined limit (which represents an unhealthy state). The limits are set as 15% for LPC, HPC, and fan stall margins, and, to 2% for the EGT margin. The health index of an engine is taken as the minimum of HPC, LPC, fan, and EGT margins. The varying operational conditions and the injected faults are the sole cause of engine failure. More details on this can be found in [33].

The data has some interesting properties and is an outstanding research benchmark for modern prognostic approaches. For instance, it includes a multi-dimensional output from a dynamic system (the turbofan engine), and the simulation incorporated measurement and process noise. This dual-state noise inclusion creates a delicate noise feature often associated with real data.

As shown in Table 3, the data consists of the following columns: engine identifier, cycle number, 3 indices that define operational settings, and 21 run-to-failure signals (defined as s1-s21). The operational settings are different combinations of Mach number (0-0.84), altitude (0-42K ft.), and throttle resolver angle (20-100).

The C-MAPSS data is partitioned into four parts, as depicted in Table 4. The table also shows the operational modes and the type of injected faults for each of the data. The fault types are HPC and fan degradation. Moreover, each data part is additionally subdivided into a training and a testing set. However, signals of the test set are intentionally truncated several periods before failure. Hence, the prediction job is to find the test set's RUL. To verify the found RUL, the real RUL is provided within the datasets for comparison. Since this study is a comparison of different optimization techniques of the ESN, we make use of only the training samples of the C-MAPSS data and then further divide the same (train set) into train/validation/test data.

TABLE 1. Input to C-MAPSS [33].

S/N	Name	Sign
1	LPC pressure-ratio modifier	LPC_PR_mod
2	LPC efficiency modifier	LPC_eff_mod
3	LPC flow modifier	LPC_flow_mod
4	HPC pressure-ratio modifier	HPC_PR_mod
5	HPC efficiency modifier	HPC_eff_mod
6	HPC flow modifier	HPC_flow_mod
7	Fan pressure-ratio modifier	fan_PR_mod
8	Fan efficiency modifier	fan_eff_mod
9	Fan flow modifier	fan_flow_mod
10	HPT flow modifier	HPT_flow_mod
11	HPT efficiency modifier	HPT_eff_mod
12	LPT flow modifier	LPT_flow_mod
13	LPT efficiency modifier	LPT_eff_mod
14	Fuel flow	W_f

A. RUL APPROXIMATION

The subsequent job is to estimate each machine's RUL from the dataset. An excellent approximation is to use the time steps before failure as the engine's RUL. So that if the engine fails at time cycle 234, for example, then at the start of the signal, the RUL will be set as 234, and the RUL at failure will be set to 0. This is a good estimate [35]. However, as discussed in [36], for most cases, the deterioration in the engine does not start to indicate prominence until some time cycles have passed. They highlighted from experiments they conducted that it will be unwise to find the RUL of the engine before it begins to show any sign of wear. This will be like finding a new engine's RUL. So most authors commence the RUL forecast only after a certain time cycle has passed [37]. To achieve this, the RUL is held constant for the initial step until a certain time has elapsed. The value of this time is set to 125 in [35], [36], [38]. Figure 6 shows an example of the RUL plot. It compares the old RUL estimate with the new RUL having a constant value at the beginning of each machine run.

B. SIGNAL PREPROCESSING

Like most machine learning processes, particularly those with noisy data, the input signal may require preparation before being added to the network. This preprocessing may involve noise reduction, collection of features, and normalization. It was observed that some signal values of the C-MAPSS stayed unchanged during run-to-failure simulation from the 21 sensor signals. Since they do not reflect system failure, we remove such signals. We select Fourteen signals from the current 21 sensor signals. These are: (2, 3, 4, 7, 8, 9, 11, 12, 13, 14, 15, 17, 20, 21) as done in [36], [37], [39]. We also performed normalization to fit the data in the span $[-1, 1]$ of our *tanh* squash function. The equation for normalization is:

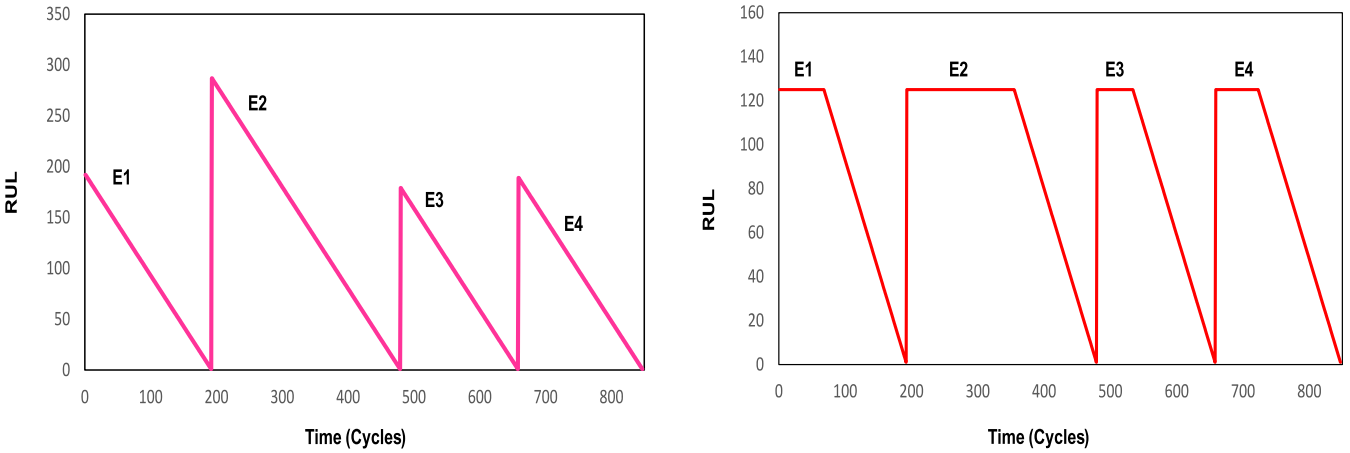
$$d_{i,j}^{norm} = \frac{2\left(d_{i,j} - d_j^{min}\right)}{d_j^{max} - d_j^{min}} - 1, \quad \forall i, j, \tag{14}$$

TABLE 2. Output sensor signals of C-MAPSS [33].

S/N	Name	Sign
1	Total pressure at HPC outlet	P30
2	Total pressure in bypass-duct	P15
3	Pressure at fan inlet	P2
4	Total temperature at LPT outlet	T50
5	Total temperature at HPC outlet	T30
6	Total temperature at LPC outlet	T24
7	Total temperature at fan inlet	T2
8	Physical core speed	Nc
9	Physical fan speed	Nf
10	LPT coolant bleed	W32
11	HPT coolant bleed	W31
12	Demanded corrected fan speed	PCNfR_dmd
13	Demanded fan speed	Nf_dmd
14	Bleed Enthalpy	htBleed
15	Burner fuel-air ratio	farB
16	Bypass Ratio	BPR
17	Corrected core speed	NRc
18	Corrected fan speed	NRf
19	Static pressure at HPC outlet	Ps30
20	Ratio of fuel flow to Ps30	phi
21	Engine pressure ratio (P50/P2)	epr
	Health Index Parameters	
1	HPC stall margin	SmHPC
2	LPC stall margin	SmLPC
3	Fan stall margin	SmFan
4	Total temperature at HPT outlet	T48 (EGT)

TABLE 3. C-MAPSS Data set overview.

Column #	Data fields	Type of data	Details
1	ID	Integer	Turbofan engine identity
2	Cycles	Integer	Cycle time
3	Setting1	Double	Operational sett. 1
4	Setting2	Double	Operational sett. 2
5	Setting3	Double	Operational sett. 3
6	s1	Double	Sensor signal 1
7	s2	Double	Sensor signal 2
...	...	...	...
26	s21	Double	Sensor signal 21



(a) Former RUL for four engines.

(b) New RUL of the same four engines.

FIGURE 6. A graph showing series of the old RUL approximation of four engines in (a) and the transformed estimate in (b) with a constant RUL of 125 at the beginning. With E1, E2, E3, and E4 denoting turbofan engines 1 to 4. Note that vertical lines on the graph are not part of the plot but serve as a partition between the RUL of the engines.

TABLE 4. Characteristics of the turbofan engine data.

Dataset	FD001	FD002	FD003	FD004
# of engines for training	100	260	100	249
# of engines for testing	100	259	100	248
# of operating modes	1	6	1	6
# of faults modes	1	1	2	2
# of training samples	20,631	53,759	24,720	61,249
# of testing samples	13,096	33,991	16,596	41,214

with $d_{i,j}$ denotes the i^{th} data of signal j and $d_{i,j}^{norm}$ is its normalized form. d_j^{min} and d_j^{max} represent the maximum and minimum values of the original sensor data of sensor j accordingly. After normalization, a Gaussian filter is then used to smooth the signals.

VI. RESULTS AND DISCUSSIONS

This section outlines the study's research findings. We employ the LI-ESN in all simulations, and we assumed that all ESNs had no feedback weights. Also, a washout time of 300 cycles was used. Moreover, both the input and reservoir weight values were randomly initialized between $[-1, 1]$ of our *tanh* squashing function. Additionally, the codes were run on MATLAB R2018b on an OptiPlex 7060 Dell PC with Intel® Core™ i5-8500 CPU @ 3.00GHz (6 CPUs) running on Windows 10 Pro 64-bit (10.0, Build 18362).

A. MACKEY-GLASS SERIES

Here, the deepESN [40], [41] is applied to predict the next time-series data on the Mackey-Glass (MG) synthetic time-series. This experiment's main aim is to compare two versions of the deepESNs with the shallow (classical) ESN and determine which is the best. The deepESN is a type of ESN with many layers. Both ESNs used to predict the next time step of the Mackey-Glass (MG) time-series. The series was developed in 1977 by Michael Mackey and Lean Glass to explain physiological control systems. The sequence is known for its nonlinear random characteristics. The series with the values of constants assigned to their most common numbers are given in Equation (15). This work uses the value of τ as 17. Figure 7 shows a typical graph of the trajectory of this series.

$$y(t+1) = \frac{0.2y(t-\tau)}{1+y(t-\tau)^{10}} - 0.1y(t) \tag{15}$$

The training length was set to 14000-time steps and the test length as 6000. Moreover, we use reservoir connectivity of 0.01 and a spectral radius of 1.25 for each network. Additionally, we set the reservoir size to 500 for the shallow ESN. While for the deep ESNs, the reservoir size was set to the total reservoir size divided by the number of layers. For example, for a five-layered network, each reservoir would have a size of 100, i.e., 500/5. We implement two types of deepESNs. The first (deepESNv1), has the same reservoir and interlayer weights for all the layers. In contrast, the second (deepESNv2) has distinct reservoir weights and interlayer weights for each layer. Each ESN was trained and tested twenty times, and the average, standard deviation, best and worst means squared error (MSE) is reported. Additionally, the average run times of each network are also reported. Initially, we run the deepESNs with different layers (3-20), and we found that the layer number of 3 was the best because the results deteriorated as the number of layers increased beyond 3. Table 5 shows the results of the experiments. From the table, the shallowESN outperformed the deepESNs. This shows that for this prediction task, there is no advantage of layering. Additionally, the deepESNv2 with different reservoir and inter-layer weight was better than the deepESNv1 with a

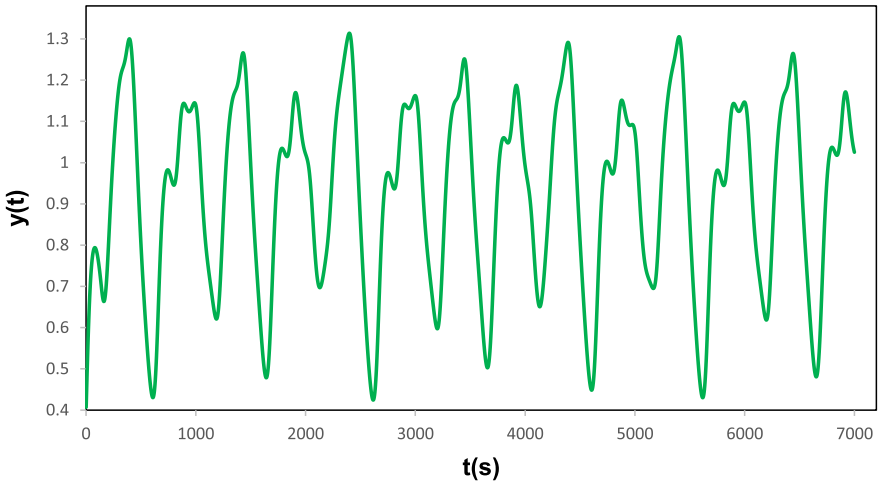


FIGURE 7. A typical plot of the Mackey-Glass series.

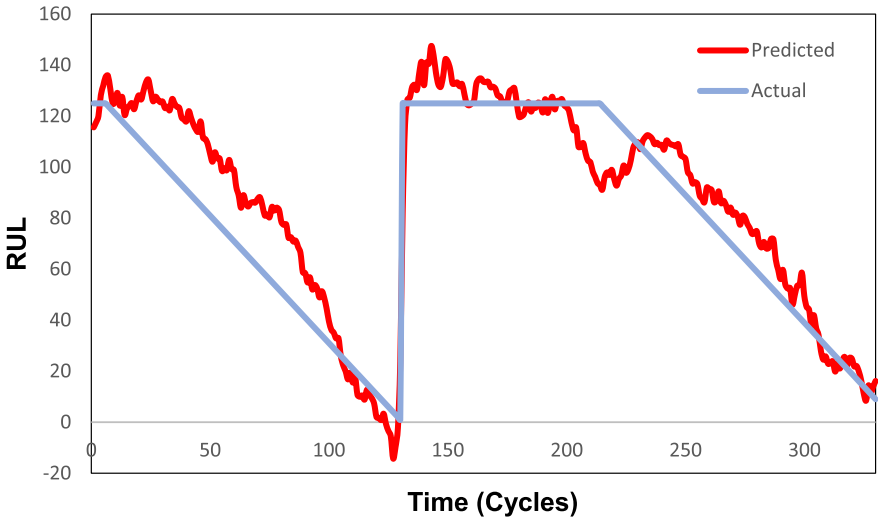


FIGURE 8. A typical plot of actual RUL Vs predicted RUL for two machines from FD001.

TABLE 5. Table showing the means squared error (MSE) prediction for the Mackey-Glass series time-series.

Stats.	shallowESN	deepESNv1	deepESNv2
Avg.	**5.94E-13**	6.46E-11	7.05E-12
SD.	**1.72E-13**	2.46E-10	1.52E-11
Best	**4.28E-13**	1.56E-12	1.11E-12
Worst	**1.16E-12**	1.13E-09	7.15E-11
Avg. time(s)	**0.302**	0.874	0.848

fixed weight values. Regarding time complexity, on average, the deepESNs took more than twice the time needed to train and test the shallowESN.

From results in this section and our preliminary implementation of the deepESN for turbofan engine RUL prediction (See Table 6), we discover that the deepESN offers no advantage at least for our type of test cases. Thus, all subsequent experiments were done on the shallow ESN and performed on the C-MAPSS data.

B. RUL PREDICTION OF TURBOFAN ENGINES

This section provides the experimental findings for applying the optimized ESN for the RUL prediction of turbofan engines. The proposed method is compared with other existing works on the same task. Also, we use all the train data in C-MAPSS in this work. For train_FD002 and train_FD004 the train/validate/test sequence are $24,500/10,500/15,000$ samples. While for train_FD001 and train_FD003 the sequence is 9800/4200/6000 samples. This division makes sense since from Table 4, we can see that FD001 and FD003 have approximately 20,000 train samples, while FD002 and FD004 have an estimate of 50,000 train samples each.

TABLE 6. Average Test MSE for deepESN with different # of layers.

# Layers	FD001	FD002	FD003	FD004
3	587.53 ± 293.73	**543.69 ± 21.65**	5163.80 ± 4601.00	1.7e6 ± 5.1e6
5	522.46 ± 209.64	611.35 ± 139.94	1463.82 ± 1636.80	7.3e4 ± 1.14e5
10	**488.49 ± 32.21**	749.20± 250.27	637.50 ± 177.77	6.0e5 ± 1.79e6
15	500.12 ± 34.05	791.08 ± 449.63	618.14 ± 191.65	1248.2 ± 1198.75
20	518.24 ± 44.86	682.97 ± 24.61	**527.02 ± 66.8**	2.1e5 ± 6.4e5
30	546.89 ± 34.67	734.85 ± 28.83	549.82 ± 30.35	**904.24 ± 97.25**
40	568.90 ± 68.75	778.95 ± 35.38	594.40 ± 74.02	909.01 ± 74.0
50	584.52 ± 57.23	876.37 ± 63.63	613.82 ± 72.12	2167.0 ± 3121.69

1) ALGORITHM PARAMETER SETTINGS

Table 7 presents the best settings for the implemented algorithms. The values of these parameters were set by carefully considering suggestions from existing literature. For each of the population-based algorithms, we set the population size to 20 and the maximum iteration to 100.

2) AVERAGE RESULTS

Table 8 shows the average test MSE for each algorithm. The table shows the average of 10 trials. We can deduce the following from the table:

- In terms of average test MSE, the proposed improved GOA outperforms other methods in three out of the four case studies, i.e., FD001, FD002, and FD003. Moreover, even in the fourth case (FD004), where it lags, it came very close to the best (DE).
- Concerning average timings, the proposed algorithm is faster than three of the methods (CS, GOA, BPSO, and PSO). However, it is slower than DE, classical ESN, deepESN, and LSTM.
- On average, we can also observe that the metaheuristic-based optimized ESNs are better than the classical ESN. This shows that optimizing the ESN with such algorithms can significantly enhance prediction accuracy. Nonetheless, the GOA and PSO come short of the classical ESN in many cases. The PSO was inferior because the authors in [20] did not attempt to optimize the reservoir's spectral radius, and thus, the ESN had no echo state property (ESP). In contrast, the GOA algorithm produced many outliers that pulled the average MSE high.
- To further put things into perspective, we have plotted the predicted RUL versus the actual RUL of some selected machines in Figure 8. Without loss of generality, this is for the FD001 data set, and the predicted path is that of our proposed method. The graph shows how the predicted values closely follow the actual path.

3) STATISTICAL SIGNIFICANCE

To further test for the statistical significance of our work, we conducted the Wilcoxon rank-sum non-parametric test. We need this test since statistics such as mean and standard deviation of Table 8 may have resulted due to chance. Ten trials of our proposed method are compared with that of other techniques. The Wilcoxon rank-sum null hypothesis is that the two trial groups are from continuous distributions having equal medians. This may loosely translate that the two groups are not significantly different. In contrast, the alternative hypothesis is that they are not, and our results are statistically significant. Table 9 shows such comparison. Each entry in the table is a comparison of the ten trials of proposed improved GOA with other methods at 5% significance level. From the low P-values (mostly < 0.05), we can categorically say that the null hypothesis is rejected in 30 of the 32 tests.

4) CONVERGENCE PLOTS

Graphs in Figure 9 show the cost (validation MSE) per iteration for each case study. Each data point is the average of the ten conducted trials. Here we compare the convergence of the proposed algorithm with other methods. From the graphs, we can see that our proposed method has the fastest convergence in 3 out of the 4 cases. These graphs also reflect the results obtained in Table 8.

Additionally, in each iteration of the proposed algorithm, we collect the average cost of the population. The graphs of Figure 10 show such an average for each case study. Again, the data is the mean of the ten conducted trials. It can be vividly seen the values rise and fall as the execution of the algorithm proceeds. This is possible because we occasionally accept inferior solutions into our population (see Algorithm 2 line 31). This is a hill-climbing behavior that helps most metaheuristics escape local optima.

5) SENSITIVITY ANALYSIS

In this case, we test the reaction of the proposed algorithm to adjustments of its parameters. The test was conducted on the FD004 test case without loss of generality. The parameters

TABLE 7. Initial algorithm parameter settings.

Alg.	Parameter	Value
ESN [42]	Reservoir Size	1500
	Spectral radius	1.0
	Leaky rate	0.3
	Reservoir connectivity	0.5
deepESN [40]	# of layers	35
	Reservoir Size	1500/(# layers)
	Spectral radius	1.0
	Leaky rate	0.3
	Reservoir connectivity	0.5
	Regularization term (reg))	1e-08
PSO [20]	Acceleration constants	[0.12, 2.2]
	Inertia w	0.9
	Reservoir Size	100-1500
	Fraction of selected weights	0.5
	Leaky rate	0.3
	Reservoir connectivity	0.5
BPSO [18]	Acceleration constants	[2, 2]
	Inertia [w_minm w_max]	[0.1, 0.6]
	Reservoir Size	1500
	Spectral radius	1.0
	Leaky rate	0.3
	Reservoir connectivity	0.5
	Regularization term (reg))	1e-08
GOA [13]	[cMin, CMax]	[0.00001, 1]
	Reservoir Size	100-1500
	Spectral radius	0.1-2.0
	Leaky rate	0.3
	Reservoir connectivity	0.1-0.9
	fraction of disabled out. conn. (D)	0.2-0.7
CS [23]	Percentage of bottom nests (P_a)	75%
	Reservoir Size	100-1500
	Spectral radius	0.1-2.0
	Leaky rate	0.3
	Reservoir connectivity	0.1-0.9
	fraction of disabled out. conn. (D)	0.2-0.7
DE [24], [43]	Crossover control param. (CR)	0.9
	[f_u,f_l]	[0.9, 0.1]
	f_i	0.5
	[τ1,τ2]	[0.1, 0.1]
	Reservoir Size	100-1500
	Spectral radius	0.1-2.0
	Leaky rate	0.3
	Reservoir connectivity	0.1-0.9
LSTM [10]	numHiddenUnits	256
	MiniBatch size	20
	fully connected size	50
	Dropout	0.5
	Validation frequency	50
	Validation patience	5
Proposed	Number of children	50% of pop. Size
	Percentage of bottom agents (P_a)	75%
	Reservoir Size	100-1500
	Spectral radius	0.1-2.0
	Leaky rate	0.1-1
	Reservoir connectivity	0.05-0.9
	input weight Scaling	0.2-2.0
	Regularization term (reg))	1e-09-1

we varied were: the number of children *numChildren* (Alg. 2 line 10), percentage of agents in the bottom group (Pa) (see Alg. 2 line 9), and a parameter we call *takingFrom* (see Alg. 2 line 30). Table 10 shows the average of 10 runs for each setting.

TABLE 8. Test MSE of the turbofan RUL prediction.

Case	Alg.	Test MSE Avg. $\pm$ σ	Avg. Time(mins)
FD001	ESN [42]	354.18 ± 23.19	0.32
	deepESN [40]	545.76 ± 53.83	5.34
	PSO [20]	949.58 ± 78.82	313.68
	BPSO [18]	277.60 ± 10.27	680.93
	GOA [13]	292.66 ± 56.06	45.93
	CS [23]	226.36 ± 8.04	176.89
	DE [24]	233.86 ± 14.95	15.07
	LSTM [10]	284.89 ± 146.60	3.56
	Proposed	**208.90 ± 10.61**	37.83
FD002	ESN [42]	479.82 ± 9.03	0.78
	deepESN [40]	767.13 ± 27.38	13.07
	PSO [20]	812.05 ± 31.33	797.87
	BPSO [18]	475.18 ± 4.53	1708.44
	GOA [13]	856.84 ± 714.88	248.91
	CS [23]	435.08 ± 3.42	285.91
	DE [24]	435.43 ± 4.17	54.18
	LSTM [10]	1049.66 ± 314.33	6.02
	Proposed	**410.10 ± 14.76**	152.08
FD003	ESN [42]	3097.22 ± 1325.58	0.31
	deepESN [40]	639.38 ± 342.44	5.29
	PSO [20]	953.00 ± 44.02	320.78
	BPSO [18]	968.70 ± 199.83	681.01
	GOA [13]	474.09 ± 117.44	56.45
	CS [23]	265.79 ± 9.53	126.77
	DE [24]	257.87 ± 22.32	8.35
	LSTM [10]	544.14 ± 412.75	3.15
	Proposed	**227.01 ± 17.64**	50.62
FD004	ESN [42]	505.11 ± 3.68	0.78
	deepESN [40]	1275.31 ± 1092.98	13.03
	PSO [20]	1017.88 ± 28.49	800.23
	BPSO [18]	522.03 ± 7.19	1683.29
	GOA [13]	737.61 ± 285.27	226.38
	CS [23]	452.15 ± 3.98	649.53
	DE [24]	**452.13 ± 3.66**	161.71
	LSTM [10]	1297.05 ± 254.20	5.18
	Proposed	460.16 ± 8.05	168.12

Regarding the *numChildren*, it can be observed that as the number of children increases, the performance of the algorithm improves. However, this comes at the expense of increased time.

Considering the percentage of worst agents, we see that as the percentage improves, the performance follows. This shows that there is value in keeping more agents in the bottom agents' category. In the *takenFrom* rows, we can

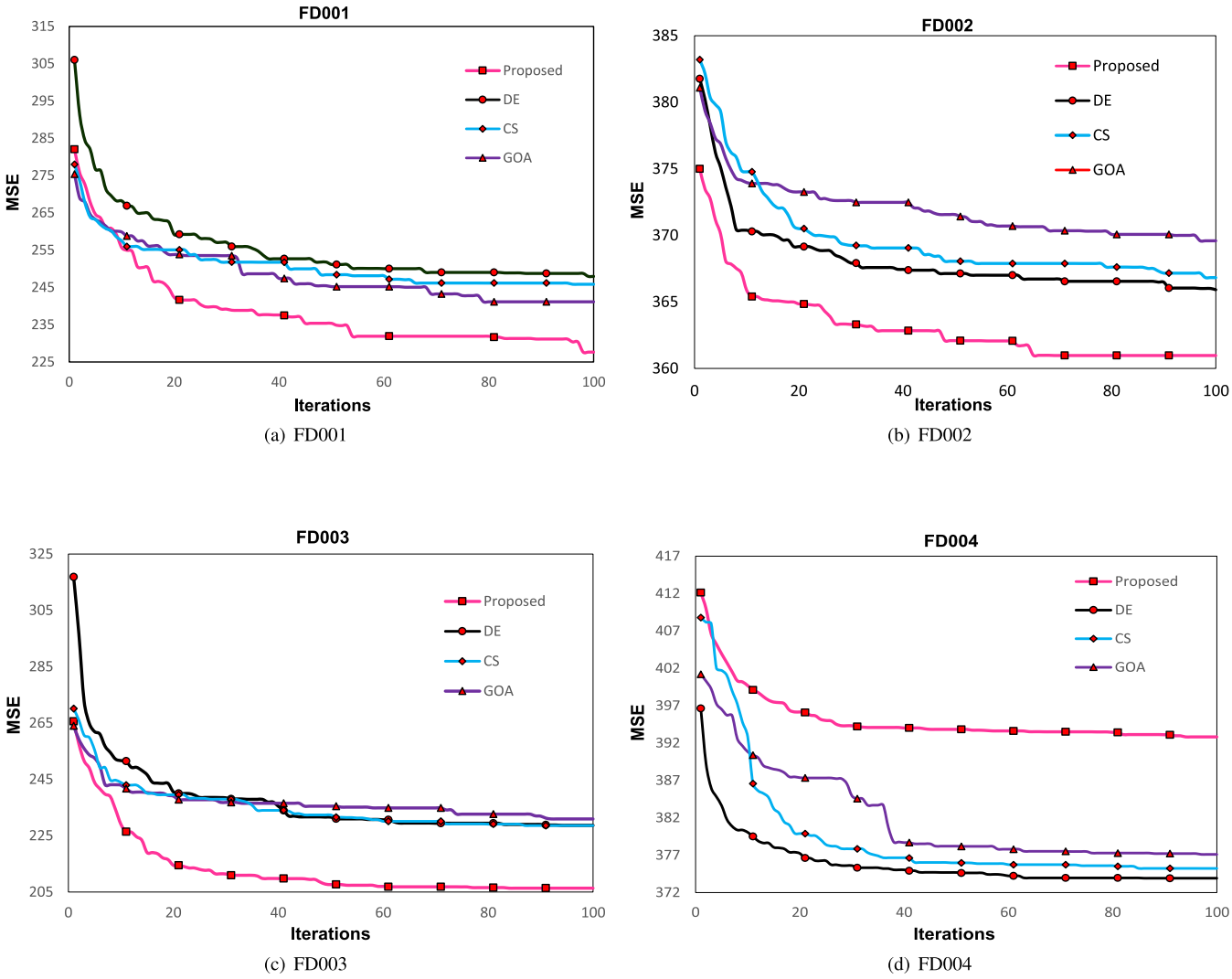


FIGURE 9. A plot of average best validation MSE per iteration for 10 trials of each case study. Comparing the convergence of the algorithms.

see that taking 25% was the best compared to taking from 75% or 50%.

VII. CHALLENGES OF ESN

It is worth noting that for real big data predictions, the ESN may suffer from instability and high computation cost.

A. INSTABILITY

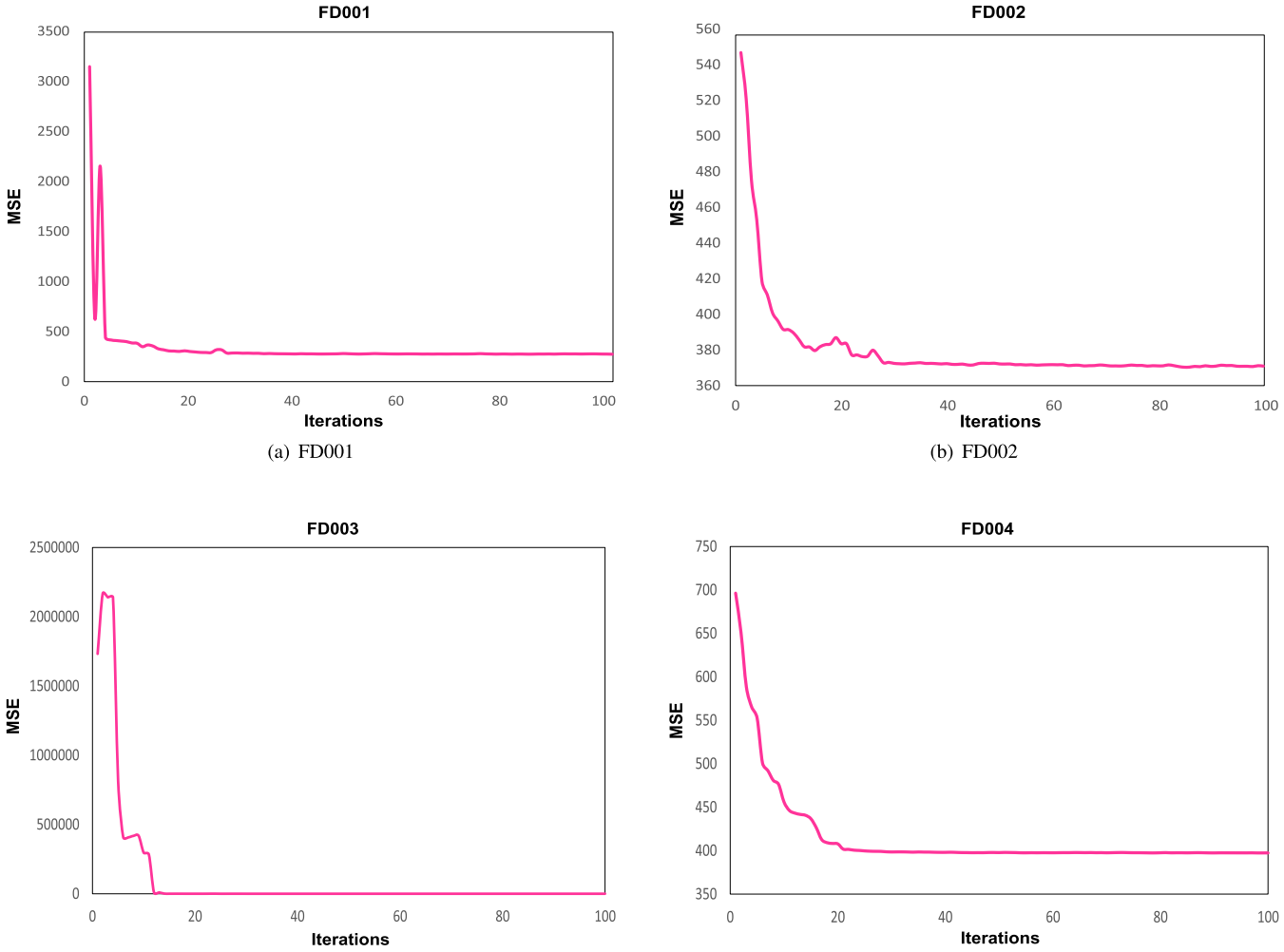
Since the data is very large, there exists a huge reservoir state (reservoir with many units). The instability often occurs due to multicollinearity between the reservoir state values [44]. Multicollinearity is a situation where there is an almost linear relationship between the independent variables (reservoir states). This means that some of the reservoir units are redundant. This problem often causes the matrix S to be ill-posed, making the output weights too sensitive to changes in S and Y. This leads to inaccurate estimates of the output weights and thus limits the generalization ability of the network.

To remedy this problem, we use regularization techniques such as the Tikhonov regularization (ridge regression) [44], [45] instead of the ordinary least squares method. The ridge regression adds a small bias that breaks the correlations between the state values (please see equation (6)) and thus decreases the variance of the predicted output weights. Another technique used to solve the collinearity problem is the Principal Component Analysis (PCA) [16].

B. HIGH COMPUTATIONAL COST

Again, this is caused by the considerable size of the reservoir. Thus, causing the linear regression to take a longer time. This is because it often involves either the multiplication of larger matrices and finding its inverse, its Singular Value Decomposing (SVD), or QR decomposition. The following methods may be employed to reduce the computational costs.

One of the proposed solutions is the use of Principal Component Regression (PCR) [46]. PCR runs the regression



(a) FD001 (b) FD002

(c) FD003 (d) FD004

FIGURE 10. A plot of average population validation MSE per iteration for 10 trials (The mean value for the 10 trials are taken).

TABLE 9. Wilcoxon rank-sum test P-values.

Test	FD001	FD002	FD003	FD004
ESN [42]	1.83e-4	1.83e-4	1.83e-4	1.83e-4
deepESN [40]	1.83e-4	1.83e-4	1.83e-4	1.83e-4
PSO [20]	1.83e-4	1.83e-4	1.83e-4	1.83e-4
BPSO [6]	1.83e-04	1.83e-04	1.83e-04	1.83e-04
GOA [13]	3.30e-04	1.83e-4	1.83e-4	1.31e-03
CS [23]	1.40e-02	7.69e-04	7.28e-03	0.345
DE [24]	2.83e-03	1.71e-03	5.8e-03	0.021
LSTM [10]	0.473	1.83e-4	1.31e-03	1.83e-4

TABLE 10. Sensitivity analysis.

Parameter	Value	Test MSE Avg. ± σ	Avg. Time(mins)
numChildren	25%	469.34 ± 5.75	66.17
	50%	460.16 ± 8.05	168.12
	75%	456.90 ± 11.63	341.93
Pa	25%	466.15 ± 11.37	98.3
	50%	462.03 ± 12.37	122.77
	75%	460.16 ± 8.05	168.12
takingFrom	25%	460.16 ± 8.05	168.12
	50%	474.58 ± 17.18	169.21
	75%	469.26 ± 16.82	136.94

only on some selected principal components of the design matrix S. Another interesting method of ridge regression is the divide and conquer kernel ridge regression of [47], [48]. Here, the design matrix is partitioned into batches, and the kernel ridge regression is run on each subset. The final global predictor is obtained from the mean of local regressions. Thirdly, if the above solutions prove abortive, the ridge regression may be formulated as an optimization

problem, and the stochastic gradient descent (SGD) may be applied [28].

VIII. CONCLUSION

We provided an improved Grasshopper Optimization Algorithm (GOA) based optimized echo state network (ESN) for predicting the remaining useful life (RUL) of turbofan engines. To improve the original GOA, we developed a new solution representation that gave birth to a simplified attraction and repulsion of grasshopper agents. Additionally, we compare the proposed technique to methods such as the Cuckoo Search (CS), Particle Swarm Optimization (PSO), Binary PSO (BPSO), Differential Evolution (DE), original GOA, the classical ESN, deep ESN, and LSTM with interesting results. Since this improved algorithm is entirely new, it would be interesting to investigate how it will perform on other optimization problems as future work.

CODE AVAILABILITY

All programming codes and data generated in the experiments are available publicly at the GitHub repository. https://github.com/bala-221/Airplane-fault-prediction

ACKNOWLEDGMENT

The authors would like to thank Universiti Teknologi PETRONAS (UTP), Malaysia, and the King Fahd University of Petroleum and Minerals, Saudi Arabia, for their support.

REFERENCES

[1] R. Zhao, R. Yan, J. Wang, and K. Mao, "Learning to monitor machine health with convolutional bi-directional LSTM networks," *Sensors*, vol. 17, no. 2, p. 273, Jan. 2017.

[2] S. Selcuk, "Predictive maintenance, its implementation and latest trends," *Proc. Inst. Mech. Eng. B, J. Eng. Manuf.*, vol. 231, no. 9, pp. 1670–1679, Jul. 2017.

[3] J. Liu, A. Saxena, K. Goebel, B. Saha, and W. Wang, "An adaptive recurrent neural network for remaining useful life prediction of lithium-ion batteries," Nat. Aeronaut. Space Admin. Moffett Field CA AMES Res., NASA, Washington, DC, USA, Tech. Rep. ADA562707, 2010.

[4] F. Ferraty, A. Goia, E. Salinelli, and P. Vieu, "Functional projection pursuit regression," *TEST*, vol. 22, no. 2, pp. 293–320, Jun. 2013.

[5] M.-Y. Cheng and M.-T. Cao, "Accurately predicting building energy performance using evolutionary multivariate adaptive regression splines," *Appl. Soft Comput.*, vol. 22, pp. 178–188, Sep. 2014.

[6] W.-C. Wang, K.-W. Chau, D.-M. Xu, and X.-Y. Chen, "Improving forecasting accuracy of annual runoff time series using ARIMA based on EEMD decomposition," *Water Resour. Manage.*, vol. 29, no. 8, pp. 2655–2675, Jun. 2015.

[7] R. Liu, B. Yang, E. Zio, and X. Chen, "Artificial intelligence for fault diagnosis of rotating machinery: A review," *Mech. Syst. Signal Process.*, vol. 108, pp. 33–47, Aug. 2018.

[8] N. Chouikhi, B. Ammar, A. Hussain, and A. M. Alimi, "Bi-level multi-objective evolution of a multi-layered echo-state network autoencoder for data representations," *Neurocomputing*, vol. 341, pp. 195–211, May 2019.

[9] C. Gallicchio, A. Micheli, and L. Pedrelli, "Comparison between DeepESNs and gated RNNs on multivariate time-series prediction," in *Proc. 27th Eur. Symp. Artif. Neural Netw., Comput. Intell. Mach. Learn. (ESANN)*, 2019, pp. 619–624.

[10] Y. Wu, M. Yuan, S. Dong, L. Lin, and Y. Liu, "Remaining useful life estimation of engineered systems using vanilla LSTM neural networks," *Neurocomputing*, vol. 275, pp. 167–179, Jan. 2018.

[11] H. Jaeger, *Tutorial on Training Recurrent Neural Networks, Covering BPPT, RTRL, EKF and the 'Echo State Network' Approach*, vol. 5. Bonn, Germany: GMD-Forschungszentrum Informationstechnik Bonn, 2002.

[12] A. Bala, I. Ismail, R. Ibrahim, and S. M. Sait, "Applications of metaheuristics in reservoir computing techniques: A review," *IEEE Access*, vol. 6, pp. 58012–58029, 2018.

[13] S. Saremi, S. Mirjalili, and A. Lewis, "Grasshopper optimisation algorithm: Theory and application," *Adv. Eng. Softw.*, vol. 105, pp. 30–47, Mar. 2017.

[14] A. A. Ferreira and T. B. Ludermir, "Genetic algorithm for reservoir computing optimization," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jun. 2009, pp. 811–815.

[15] A. A. Ferreira, T. B. Ludermir, and R. R. B. de Aquino, "An approach to reservoir computing design and training," *Expert Syst. Appl.*, vol. 40, no. 10, pp. 4172–4182, Aug. 2013.

[16] Q. Ma, L. Shen, and G. W. Cottrell, "Deep-ESN: A multiple projection-encoding hierarchical reservoir computing framework," 2017, *arXiv:1711.05255*. [Online]. Available: http://arxiv.org/abs/1711.05255

[17] S. Zhong, X. Xie, L. Lin, and F. Wang, "Genetic algorithm optimized double-reservoir echo state network for multi-regime time series prediction," *Neurocomputing*, vol. 238, pp. 191–204, May 2017.

[18] H. Wang and X. Yan, "Optimizing the echo state network with a binary particle swarm optimization algorithm," *Knowl.-Based Syst.*, vol. 86, pp. 182–193, Sep. 2015.

[19] S. Basterrech, E. Alba, and V. Snášel, "An experimental analysis of the echo state network initialization using the particle swarm optimization," in *Proc. 6th World Congr. Nature Biol. Inspired Comput. (NaBIC)*, Jul. 2014, pp. 214–219.

[20] N. Chouikhi, B. Ammar, N. Rokbani, and A. M. Alimi, "PSO-based analysis of echo state network parameters for time series forecasting," *Appl. Soft Comput.*, vol. 55, pp. 211–225, Jun. 2017.

[21] E. J. Amaya and A. J. Alvares, "Prognostic of RUL based on echo state network optimized by artificial bee colony," *Int. J. Prognostics Health Manage.*, vol. 7, no. 1, pp. 46–57, May 2016.

[22] A. Bala, I. Ismail, and R. Ibrahim, "Cuckoo search based optimization of echo state network for time series prediction," in *Proc. Int. Conf. Intell. Adv. Syst. (ICIAS)*, Aug. 2018, pp. 1–6.

[23] A. Bala, I. Ismail, R. Ibrahim, S. M. Sait, and H. O. Salami, "Prediction using cuckoo search optimized echo state network," *Arabian J. Sci. Eng.*, vol. 44, no. 11, pp. 9769–9778, Nov. 2019.

[24] L. Wang, H. Hu, X.-Y. Ai, and H. Liu, "Effective electricity energy consumption forecasting using echo state network improved by differential evolution algorithm," *Energy*, vol. 153, pp. 801–815, Jun. 2018.

[25] M. Lukoševičius, "A practical guide to applying echo state networks," in *Neural Networks: Tricks Trade*. Berlin, Germany: Springer, 2012, pp. 659–686.

[26] M. Lukoševičiusand H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Comput. Sci. Rev.*, vol. 3, no. 3, pp. 127–149, Aug. 2009.

[27] C. Gallicchio, "Chasing the echo state property," 2018, *arXiv:1811.10892*. [Online]. Available: http://arxiv.org/abs/1811.10892

[28] H. Jaeger, M. Lukoševičius, D. Popovici, and U. Siewert, "Optimization and applications of echo state networks with leaky-integrator neurons," *Neural Netw.*, vol. 20, no. 3, pp. 335–352, Apr. 2007.

[29] S. M. Sait and H. Youssef, *Iterative Computer Algorithms With Applications in Engineering: Solving Combinatorial Optimization Problems*. Washington, DC, USA: IEEE Computer Society Press, 1999.

[30] S. Z. Mirjalili, S. Mirjalili, S. Saremi, H. Faris, and I. Aljarah, "Grasshopper optimization algorithm for multi-objective optimization problems," *Int. J. Speech Technol.*, vol. 48, no. 4, pp. 805–820, Apr. 2018.

[31] S. J. Simpson, A. R. McCaffery, and B. F. Hägele, "A behavioural analysis of phase change in the desert locust," *Biol. Rev.*, vol. 74, no. 4, pp. 461–480, Jan. 2007.

[32] I. Aljarah, A.-Z. Ala'M, H. Faris, M. A. Hassonah, S. Mirjalili, and H. Saadeh, "Simultaneous feature selection and support vector machine optimization using the grasshopper optimization algorithm," *Cognit. Comput.*, vol. 10, no. 3, pp. 478–495, 2018.

[33] A. Saxena, K. Goebel, D. Simon, and N. Eklund, "Damage propagation modeling for aircraft engine run-to-failure simulation," in *Proc. Int. Conf. Prognostics Health Manage. (PHM)*, Oct. 2008, pp. 1–9.

[34] M. M. Rigamonti, P. Baraldi, and E. Zio, "Echo state network for the remaining useful life prediction of a turbofan engine," in *Proc. Annu. Conf. Prognostics Health Manage. Soc.*, 2016, pp. 255–270.

[35] E. Ramasso, "Investigating computational geometry for failure prognostics," *Int. J. Prognostics Health Manage.*, vol. 5, no. 1, p. 005, 2014.

[36] F. O. Heimes, "Recurrent neural networks for remaining useful life estimation," in *Proc. Int. Conf. Prognostics Health Manage. (PHM)*, Oct. 2008, pp. 1–6.
[37] X. Li, Q. Ding, and J.-Q. Sun, "Remaining useful life estimation in prognostics using deep convolution neural networks," *Rel. Eng. Syst. Saf.*, vol. 172, pp. 1–11, Apr. 2018.
[38] Z. Zhao, B. Liang, X. Wang, and W. Lu, "Remaining useful life prediction of aircraft engine based on degradation pattern learning," *Rel. Eng. Syst. Saf.*, vol. 164, pp. 74–83, Aug. 2017.
[39] C. Zhang, P. Lim, A. K. Qin, and K. C. Tan, "Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2306–2318, Oct. 2017.
[40] C. Gallicchio, A. Micheli, and L. Pedrelli, "Design of deep echo state networks," *Neural Netw.*, vol. 108, pp. 33–47, Dec. 2018.
[41] C. Gallicchio, A. Micheli, and L. Pedrelli, "Deep reservoir computing: A critical experimental analysis," *Neurocomputing*, vol. 268, pp. 87–99, Dec. 2017.
[42] H. Jaeger, "The 'echo state' approach to analysing and training recurrent neural networks-with an erratum note," German Nat. Res. Center Inf. Technol. GMD, Bonn, Germany, Tech. Rep. 148, 2001, vol. 148, no. 34, p. 13.
[43] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," *IEEE Trans. Evol. Comput.*, vol. 10, no. 6, pp. 646–657, Dec. 2006.
[44] M. Xu and M. Han, "Adaptive elastic echo state network for multivariate time series prediction," *IEEE Trans. Cybern.*, vol. 46, no. 10, pp. 2173–2183, Oct. 2016.
[45] J. Qiao, L. Wang, and C. Yang, "Adaptive lasso echo state network based on modified Bayesian information criterion for nonlinear system modeling," *Neural Comput. Appl.*, vol. 31, no. 10, pp. 6163–6177, Oct. 2019.
[46] L. Boccato, A. Lopes, R. Attux, and F. J. Von Zuben, "An extended echo state network using volterra filtering and principal component analysis," *Neural Netw.*, vol. 32, pp. 292–302, Aug. 2012.
[47] Y. Zhang, J. Duchi, and M. Wainwright, "Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates," *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 3299–3340, Jan. 2015.
[48] Y. Zhang, J. Duchi, and M. Wainwright, "Divide and conquer kernel ridge regression," in *Proc. Conf. Learn. Theory*, 2013, pp. 592–617.

ABUBAKAR BALA (Member, IEEE) received the bachelor's degree (Hons.) in computer engineering from Bayero University Kano, Nigeria, in 2011, and the master's degree in computer engineering from the King Fahd University of Petroleum and Minerals (KFUPM), Saudi Arabia, in 2015. He is currently pursuing the Ph.D. degree in electrical engineering with Universiti Teknologi PETRONAS (UTP), Malaysia. He is also a Lecturer with Bayero University Kano (BUK), Nigeria. He is working on developing new algorithms to optimize the echo state network (ESN), a type of recurrent neural network. His research interests include optimization, artificial neural networks, fault predictions, cloud computing, and renewable energy.

IDRIS ISMAIL received the bachelor's degree in electrical engineering from Wichita State University, KS, USA, in 1986, the master's degree in control system engineering from The University of Sheffield, U.K., in 2000, and the Ph.D. degree in electrical and electronics engineering from The University of Manchester, U.K., in 2009. He is currently an Associate Professor with Universiti Teknologi PETRONAS, Malaysia, and a Registered Professional Engineer with the Board of Engineers, Malaysia. His current research interests include data analytics and model predictive control.

ROSDIAZLI IBRAHIM (Senior Member, IEEE) received the B.Eng. degree in electrical engineering from Universiti Putra Malaysia, Seri Kembangan, Malaysia, in 1996, the M.Sc. degree in automation and control from Newcastle University, Newcastle upon Tyne, U.K., in 2000, and the Ph.D. degree in electrical and electronic engineering from the University of Glasgow, Glasgow, U.K., in 2008. He is currently an Associate Professor and a Chartered Engineer registered with the U.K. Engineering Council. He is also the Dean of the Center of Postgraduate Studies, Universiti Teknologi PETRONAS. His current research interests include intelligent control and nonlinear multivariable process modeling for process control applications.

SADIQ M. SAIT (Senior Member, IEEE) received the bachelor's degree in electronics engineering from Bangalore University, in 1981, and the master's and Ph.D. degrees in electrical engineering from KFUPM, in 1983 and 1987, respectively. He is currently a Professor of computer engineering and the Director of the Center for Communications and IT Research, KFUPM. He has authored over 200 research articles, contributed chapters to technical books, and lectured in over 25 countries. He is also the principal author of two books. He received the Best Electronic Engineer Award from the Indian Institute of Electrical Engineers, Bengaluru (where he was born), in 1981.

DIEGO OLIVA (Member, IEEE) received the B.S. degree in electronics and computer engineering from the Industrial Technical Education Center (CETI), Guadalajara, Mexico, in 2007, the M.Sc. degree in electronic engineering and computer sciences from the University of Guadalajara, Mexico, in 2010, and the Ph.D. degree in informatics from the Universidad Complutense de Madrid, in 2015. He is currently an Associate Professor with the University of Guadalajara. His research interests include evolutionary and swarm algorithms, hybridization of evolutionary and swarm algorithms, and image processing.