Reproduction Report of "Neural Net-Enhanced Competitive Swarm Optimizer for Large-Scale Multiobjective Optimization"

Abstract

This reproduction report aims to reproduce the study titled "Neural Network-enhanced Competitive Swarm Optimizer (NN-CSO) for Large-scale Multi-objective Optimization Problems." In order to fill the research gap in the evolution of winner particles in traditional competitive swarm optimizers (CSO). The original paper has proposed NN-CSO, which combines the strategies of CSO with a neural network model to guide the search of loser particles and evolve the winner ones. This amalgamation of CSO search patterns and neural network optimization presents a promising approach for addressing large-scale multi-objective optimization problems. This report reproduces NN-CSO as far as possible and validates the effectiveness of the NN-CSO method demonstrated in the original study.

Keywords: Reproduction report, competitive swarm optimizer, neural network, paper reproduction.

1 Introduction

The integration of Competitive Swarm Optimizers (CSO) with Neural Networks (NN) to address Large-scale Multi-objective Optimization Problems (LMOP) represents an innovative approach in the field of optimization algorithms. CSO segregates a population of particles into winners and losers, utilizing the superior particles to guide the inferior ones in their search strategies. Although CSO has demonstrated promising performance in LMOP resolution, current studies often overlook the evolutionary trajectory and refinement of winner particles, despite their pivotal role in shaping the final optimization outcomes.

The proposed method by Li in the original paper [12], named NN-CSO, seeks to bridge this research gap by combining the basic CSO strategy with a well-trained Neural Network model. This methodology begins by competitively categorizing the particle population into winners and losers. Subsequently, the losers and winners are employed as the inputs and outputs, respectively, for training a Neural Network model. This model learns and encapsulates superior evolutionary guidance, directing the loser particles towards the winner ones, thereby enhancing their optimization paths. Upon completion of the model training, the successful particles evolve under the tutelage of the well-trained Neural Network model, while the losers persist in benefitting from the guidance provided by the winner particles. This concurrent evolution and guidance mechanism retain the inherent search pattern of CSO while enhancing its efficiency through the informed evolutionary guidance from the neural network model.

The significance of this research lies in its innovative fusion of CSO and Neural Networks, promising a breakthrough in addressing LMOPs by incorporating a more refined evolution strategy for successful particles. Given my research focus on the convergence of evolutionary algorithms and edge computing, this study aligns seamlessly with the exploration of advanced optimization techniques in complex computational environments. The potential implications extend to enhancing evolutionary algorithms in edge computing applications, where optimized algorithms play a critical role in resource-constrained environments.

2 Related works

2.1 Competitive Swarm Optimizer for LMOPs

The Competitive Swarm Optimizer (CSO) was initially introduced in literature [4] as a variant of Particle Swarm Optimization (PSO) designed specifically to address large-scale variable problems. PSO, a typical Multi-Swarm Optimization (MSO) method, is renowned for its ease of implementation and high search efficiency [15]. In traditional PSO, each particle possesses two fundamental attributes, i.e., velocity (v) and position (x), as well as two historical memory attributes, i.e., personal best position (pbest) and global best position (gbest). Generally, pbest and gbest are regarded as the cognitive and social components of particles. The particle updates its velocity (v) by learning from pbest and gbest, represented as:

$$v \leftarrow \omega v + c_1 r_1 (pbest - x) + c_2 r_2 (gbest - x). \tag{1}$$

The position (x) is then updated as:

$$x \leftarrow x + v, \tag{2}$$

Here, ω denotes the inertia weight, c_1 and c_2 are learning coefficients, and r_1 and r_2 are two random vectors in the range $[0, 1]^n$. However, in scenarios where pbest and gbest become trapped in local optima, the learning strategy of PSO may lead to premature convergence [4], especially in Large Scale Optimization Problems (LSOP) [14]. Addressing this issue of premature convergence without involving pbest and gbest, CSO proposes a competitive learning strategy to alleviate this problem. Each particle in CSO is a potential leader and guides the search when it wins a competition. Consequently, CSO retains two fundamental attributes: v and x. CSO randomly selects two particles from the swarm P and competes based on their fitness values. Only the losing particle with attributes (v_l, x_l) learns from the winning particle with attributes (v_w, x_w), updating its v_l as:

$$v_l \leftarrow r_1 vl + r_2 (x_w - x_l) + \phi r_3 (\bar{x} - x_l). \tag{3}$$

Here, r_1, r_2, and r_3 are three random vectors in $[0, 1]^n$, $\bar{x}$ represents the average position of relevant particles, and ϕ is the corresponding control parameter. Subsequently, x_l is updated according to equation (3). Lacking historical memory, the winning particles x guide the losing particles as cognitive and social components, thus simulating biological behavior intelligently and credibly [4].

2.2 Neural Network Model

Artificial Neural Networks (ANNs) have been extensively researched and applied in the field of time series forecasting [1,3,7]. Zhang et al. [19] conducted a comprehensive review of ANNs. ANNs excel in their flexible

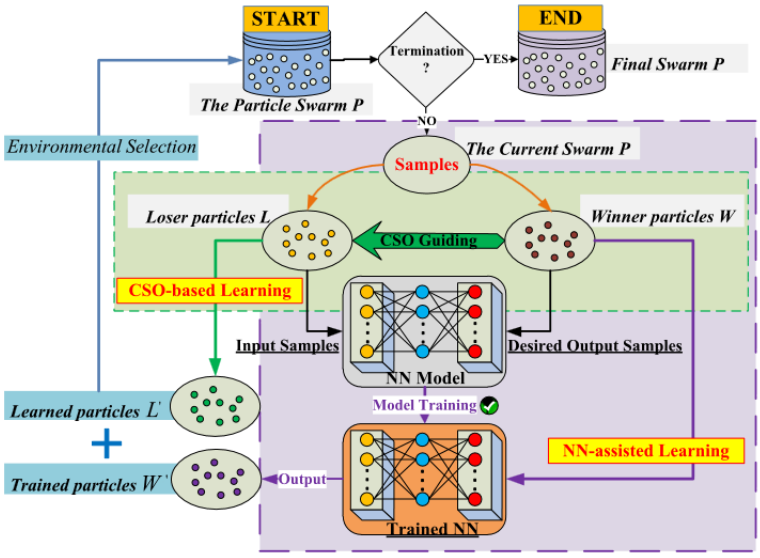

Figure 1. Outline of NN-CSO [12], including the sampling process, model training, NN-assisted learning, CSO-based learning, and environmental selection.

nonlinear modeling capabilities, robust adaptability, and capacity for learning and large-scale parallel computation [16]. Unlike traditional approaches that require specifying specific model forms, artificial neural networks adaptively construct models based on data characteristics. This data-driven approach is suitable for many empirical datasets where there might be no theoretical guidance for an appropriate data generation process. Within artificial neural networks, feedforward neural networks are the most widely used type. Backpropagation Neural Networks (BPNN) are among the most common feedforward neural networks [17]. Also known as error backpropagation networks, BPNNs are multi-layer mapping networks that adjust weights by minimizing errors during forward information propagation. Typically, a single-hidden-layer BPNN can approximate various nonlinear functions with arbitrary precision [2], making BPNNs highly favored in predicting complex nonlinear systems. The aforementioned studies and methodologies provide insights into the application and advantages of artificial neural networks in time series forecasting, laying a solid theoretical foundation and methodological support for subsequent research.

3 Method

3.1 Overall framework of NN-CSO

The overall framework of NN-CSO is shown in Figure 1, including the sampling process, model training, NN-assisted learning, CSO-based learning and environment selection. Specifically, the NN-CSO starts with a particle swarm **P** with N particles. Then, the algorithm enters the main evolutionary process of P, which is the purple part of the figure. The sampling process is first performed for constructing the input samples (i.e., loser particles, abbreviated as L) and the desired output samples (i.e., winner particles, abbreviated as W) of the NN model. More details of Samples are described in Section 3.2. Then, the network training process is executed using inputs W and L. A detailed description of network training is given in Section 3.3. After the model training is completed, the algorithm will execute two offspring reproduction strategies, including NN-

assisted learning strategy and CSO-based learning strategy, respectively. More details of offspring reproduction are introduced in Section 3.4. After each offspring reproduction, environmental selection is performed to produce the next generation of particle population. Noted that the adopted CSO-based learning strategies and environmental selections can be chosen from existing CSOs. When the termination condition is reached, the main evolutionary loop terminates. Finally, all optimal particles in P will be reported as the final optimal solution set.

3.2 Sampling process

The algorithm initiates the sample collection process to create input and expected output samples for training the Neural Network (NN) model. Employing the current particle swarm P as an illustration, the procedure describes the required sample gathering process for training the NN model. Specifically, the algorithm executes the pairwise competition process within CSO, segmenting the particle swarm into two subsets: the set of winning particles W and the set of losing particles L based on Pareto-dominance relationships. To address potential mutual non-dominance among particles in subsequent iterations, the shift density estimate (SDE) method [13] is further applied to assess non-dominance, effectively reflecting the diversity among particles. Ultimately, the algorithm prepares the training samples, where the loser particle set L and winner particle set W are considered input and expected output samples, respectively, for the model.

The formula for calculating SDE is as follows:

$$f_{\mathrm{SDE}}(x_i) = \min_{x_j \in P \Lambda i \neq j} \sqrt{\sum_{m=1}^{M} (\max\{0, f_m(x_j) - f_m(x_i)\})^2}, \tag{4}$$

where $f_m(x_i)$ denotes the mth objective value of particle x_i, and M is the number of objectives.

3.3 Model training

After completing the aforementioned sampling process, the algorithm proceeds to the model training phase. This chapter introduces the general training process of the NN model. Initially, relevant parameters of the NN model, namely bias and weight vectors, are initialized. Subsequently, the training process is executed, involving the computation of the current error rate E. Moreover, the weight vectors and biases of each neuron are updated using the gradient descent method, denoted as $\frac{\partial E}{\partial(w,v)}$. It's important to note that training terminates when the current error rate E is no longer greater than the predefined expected minimum error rate E_{min} or when the training time T reaches the maximum allowed training time T_{max}. Finally, the well-trained NN model captures the approximate optimal evolutionary search directions. Consequently, the final NN model serves as a reproduction operator to evolve the winning particles, aiming to generate high-quality solutions and guide the CSO search.

3.4 Offspring reproduction

After the model training is completed, offspring reproduction is performed. Figure 2 illustrates the schematic diagrams of two offspring reproduction strategies, including the NN-assisted learning strategy and the CSO-based learning strategy.

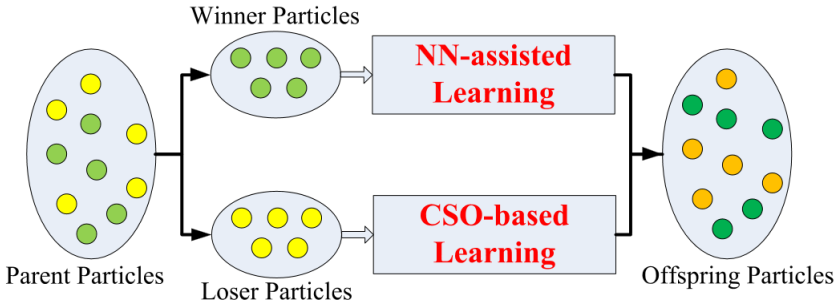

Figure 2. Two offspring reproduction strategies in NN-CSO [12]: NN-assisted learning for winner particles and CSO-based learning for loser particles.

1. NN-assisted learning strategy: The NN-assisted learning strategy is specifically designed to guide the evolution of winning particles by employing the well-trained NN model obtained in Section 3.3. This process entails utilizing the trained NN model's expertise by feeding winning particles as inputs into the model. Subsequently, the model evaluates these particles based on their features, deriving guidance for their evolutionary path. By drawing upon the learned knowledge within the well-trained NN model, this strategy nurtures potentially fruitful descendants, thereby steering CSO towards more promising evolutionary trajectories. This approach maximizes the knowledge encapsulated within the NN model to guide the evolution of winning particles in a more targeted and forward-thinking manner.

2. CSO-based learning strategy: The evolution of loser particles employs a learning strategy based on CSO, where the velocity and position of these loser particles are updated by learning from the winning particles. This ensures the preservation of the original search pattern of CSO. This process involves the loser particles, identified through pairwise competition within the CSO framework. Specifically, these loser particles adapt their velocity and position by assimilating knowledge from the winning particles. The update of the loser particles' velocity (V_l) and position (X_l) is achieved by considering the winning particles' information (X_w).

   Mathematically, the update mechanism involves the following equations:

$$V_l(t+1) = r_1 V_l(t) + r_2(X_w(t) - X_l(t)) + \phi r_3(\bar{X}(t) - X_l(t)), \tag{5}$$

$$X_l(t+1) = X_l(t) + V_l(t+1). \tag{6}$$

   Here, r_1, r_2, and r_3 represent random vectors in the range [0, 1], $X(t)$ signifies the average position of relevant particles at iteration t, and ϕ stands for the respective control parameter. This process ensures that the loser particles learn and adapt from the winning particles' characteristics, thereby maintaining the fundamental search pattern of the CSO algorithm.

4 Implementation details

4.1 Comparison to open source code

Since this original article does not publicize any relevant source code, the reproduced code cannot be compared with the public source code.

Algorithm 1 NN-CSO

Input: $MaxFes$ (termination criterion), N (swarm size).

Output: P (the final optimal swarm).

$P \leftarrow$ initialize a particle swarm of size N;
$NN \leftarrow$ initialize the network model;
while $MaxFes$ is not reached **do**
    $W, L \leftarrow$ **Samples** (P);
    $NN_{Trained} \leftarrow$ **Network Training** (W, L);
    $W^{'} \leftarrow NN_{Trained}$ (W);
    $L^{'} \leftarrow$ **CSO** (W, L);
    $P \leftarrow$ Environmental Selection $(L^{'} \cup W^{'} \cup P)$;
end while

Algorithm 2 Network Training (W, L)

Input: W (the winner samples), L (the loser samples), E_{min} (the minimum error) and T_{max} (the maximum training times).

Output: $NN_{Trained}$ (the trained network model).

initialize the biases and weight vectors for NN;
while all samples in W and L **do**
    calculate the current error rate E of NN;
    if $E > E_{min}$ or $T < T_{max}$ **then**
        adjust the parameters using $\partial E/\partial(w, v)$;
        T=T+1;
    end if
end while

4.2 Overall framework of NN-CSO

In the NN-CSO reproduction code as shown in **Algorithm** 1, N represents the population size, while V denotes the weight vector for objectives, a crucial parameter for environmental selection. Initially, the population is randomly initialized, meaning each particle within the population is generated randomly. Upon entering the loop, each particle in the population is first evaluated to acquire Fitness values. Subsequently, it enters what's termed as the sampling module, though it's more a categorization process for the particles in the population. Based on their Fitness values, the particles are categorized into two groups: the winners (W) and losers (L). Winners are the top $2/N$ particles in terms of fitness, while losers consist of particles with lower fitness values. Once W and L are identified, the process proceeds to the network training and particle learning module, as further elaborated.

4.3 Network Training

As shown in **Algorithm** 2, the reproduced code shows the backpropagation neural network (BPNN) training process within a loop, conditioned by the maximum number of iterations (maxIteration). Within this loop,

Algorithm 3 NN-assisted learning strategy

Input: W (the winner samples).

Output: New_W (the new winner samples).

initialize the new winner samples New_W;
for all samples a in W **do**
    $b = logsig(a * V + HNT)$; // calculate the hidden layer activation value
    $c = logsig(b * W + ONT)$; // calculate the final prediction
    $New\_W \leftarrow c$;
end for

each iteration evaluates and updates the weights and thresholds of the neural network layers for a given number of training samples (trainNum).

The process starts by extracting input (a) from the loser particles and setting the desired output (ck) from the winner particles. It then proceeds with the forward propagation by calculating the hidden layer activations (b) and subsequently the output layer activations (c) using the hyperbolic tangent (tansig) activation function.

Subsequently, the code computes the errors in the output layer (d) and hidden layer (e) to facilitate the backpropagation process. These errors are then utilized to update the weights (W) and thresholds (ONT) in the connection between the hidden and output layers. Similarly, the weights (V) and thresholds (HNT) associated with the input and hidden layers are updated accordingly.

This updating process iterates within the loop until the set maximum number of iterations is reached or until the error rate (d) falls below a specified threshold of 0.01. Overall, the code exemplifies the fundamental training process of a neural network through forward and backward propagation, adjusting weights and thresholds to minimize errors iteratively over multiple epochs.

4.4 Offspring reproduction

In this section, the NN-assisted learning strategy and CSO-based learning strategy are performed for the winner particle and the loser particle, respectively, as reproduced as follows.

1. NN-assisted learning strategy:

   The reproduced code by **Algorithm** 3 represents the prediction phase using a trained neural network (NN) model. It involves utilizing the trained weights and thresholds from the BPNN for prediction purposes. It starts by looping through the input samples (inputn) derived from Winner particles. Within each iteration, the code calculates the activation of the hidden layer (b) using the logistic sigmoid activation function, given the input and the trained weights and thresholds. Following the calculation of the hidden layer activations, the code computes the final predictions (c) by using the calculated hidden layer activations, the trained weights, and thresholds associated with the connection between the hidden and output layers using the logistic sigmoid function. Finally, the generated predictions are stored in the initialized population. Additionally, the predictions are mapped back to the original range to transform the predictions from the normalized range back to the original output space (outputs). This process allows the trained NN model to predict outputs based on the provided input data, leveraging the learned weights

Algorithm 4 CSO-based learning strategy

Input: W (the winner samples), L (the loser samples).

Output: **OffDec** (the new Loser samples).

1: $r1, r2 \leftarrow$ random values from [0, 1];

2: **OffVel** $= r1 * L + r2 * (W - L)$;

3: **OffDec** $= L +$ **OffVel** $+ r1 * ($**OffVel** $- L)$;

Algorithm 5 Code of non-dominated selection strategy

Input: Population.

Output: The new Population.

1: $[FrontNo, MaxFNo] =$ **NDSort**$(Population, N)$;

2: $Next = FrontNo < MaxFNo$;

3: $CrowdDis =$ **Crowding Distance**$(Population, FrontNo)$;;

4: $Next \leftarrow$ Select solutions in the last front based on their $CrowdDis$

5: $Population = Population(Next)$;

and thresholds from the training phase.

2. CSO-based learning strategy: The reproduced code by **Algorithm** 4 represents a competitive swarm optimizer (CSO) utilized for a learning strategy. In this context, it involves the evolution and adaptation of the particles based on the competitive learning mechanism within the swarm. First, random values (r1 and r2) are generated to facilitate the competitive evolution of particles within the swarm. These random values are used to compute the new velocity and the updated position of the particles. The OffVel calculation involves a combination of the loser particle's velocity and the difference between the winner and loser particle positions. The random values (r1 and r2) scale and determine the contributions of these components in the calculation of the updated velocity. Subsequently, the OffDec (new position) is computed by adjusting the loser particle's position based on the computed OffVel and the previously defined random values. This operation essentially updates the position of the loser particle along with its velocity. Overall, this CSO-based learning strategy is designed to update the velocity and position of the loser particle in the swarm, leveraging both winner and loser particle information along with random factors to guide the evolutionary process of the swarm particles.

4.5 Environmental selection

4.6 Non-dominated Selection Strategy

Since there is no support for the associated source code, a non-dominated selection strategy is used as the selection operator, and its reproduction code is shown in **Algorithm** 6.

The reproducted code implements a selection process for multi-objective optimization, focusing on maintaining a diverse set of non-dominated solutions. The code begins with a method known as non-dominated sorting, categorizing solutions into different fronts based on their objective values and constraints. This sorting identifies which solutions dominate others, assigning them to various tiers or fronts. The maximum front

Algorithm 6 Angle Penalty Selection Strategy

Input: Population.

Output: The new Population.

1: $PopObj \leftarrow$ normalize objective values of Population;
2: $CV \leftarrow$ calculate the violation degree for each solution;
3: $cosine \leftarrow$ calculate the cosine similarity between vectors;
4: $Angle \leftarrow$ calculate the angle between solutions and vectors;
5: choose best solution with the smallest Adjusted Projection Distance (APD) and the smallest constraint violation;
6: Updata the new Population using the best solution;

number, denoted as MaxFNo, signifies the furthest non-dominated front among the solutions. Next, it marks solutions from fronts not belonging to the maximum front as candidates for the next generation. The algorithm then calculates the crowding distance for each solution in the population. This metric assesses the density of solutions within their respective fronts. To select solutions for the next generation while ensuring diversity, the algorithm prioritizes solutions from the last front based on their crowding distances. Finally, the population is updated for the subsequent generation. This step ensures that the non-dominated solutions, chosen based on their crowding distances, move forward for further iterations in the optimization process.

4.7 Angle Penalty Selection Strategy

According to recent work [8], an angular penalty selection strategy is also used as a second environment selection operator, and its reproduction code is shown in **Algorithm** 6. The reproduced code aims to select solutions for the next generation based on angular information with penalty coefficients in the context of multi-objective optimization.

Initially, the objective values of the population are normalized by subtracting the minimum values along each objective dimension, effectively translating the population to an appropriate range. Following this normalization step, the degree of constraint violation for each solution in the population is computed. A cosine similarity matrix is generated based on the angle between vectors in the reference set (V). The solutions in the population are associated with the reference vectors by calculating the cosine of the angles between each solution's objective vector and the reference vectors.

Next, for each reference vector, a selection criterion is applied: 1) Solutions associated with a reference vector are divided into two groups based on constraint violation status. 2) If there are constraint-violating solutions associated with a reference vector, the solution with the minimum constraint violation (CV) is chosen. 3) For non-violating solutions, an angle-penalty distance (APD) metric is computed to select the solution that balances the angle with the proximity to the reference vector. The solution with the minimum APD is chosen.

Finally, based on the selected solutions from the above criteria, the population for the next generation is updated, discarding solutions that do not contribute to the next iteration. This selection process ensures the generation of a well-balanced and diverse set of solutions for the subsequent evolutionary steps.

5 Results and analysis

5.1 Test Problems and Performance Measures

Benchmark problem: In the experiment, like the original article [12], five test suites (LSMOP1-LSMOP9 [5], UF1-UF10 [20], DTLZ1-DTLZ7 [6], and WFG1-WFG9 [11]) are used to validate the performance of the algorithms (i.e., NN-CSO-ND and NN-CSO-AP) that we replicated.

Regarding the number of targets (M), $M = \{2, 3, 5, 8, 10\}$ is used in the experiments for LSMOP1-LSMOP9, DTLZ1-DTLZ7, and WFG1-WFG9, $M = 2$ for UF1-UF7, and $M = 3$ for UF8-UF10. Regarding the number of decision variables (D), $D = \{100, 200, 500, 1000\}$.

Performance Measure: In order to accurately reflect the quality of the final solution set obtained by each comparison algorithm, we adopt the inverted generational distance (IGD) as the performance measure. the smaller the IGD value, the better the algorithm performance. Note that all problems were tested with 10 independent runs and the mean and standard deviation of the IGD values were recorded.

5.2 The effectiveness of the NN model

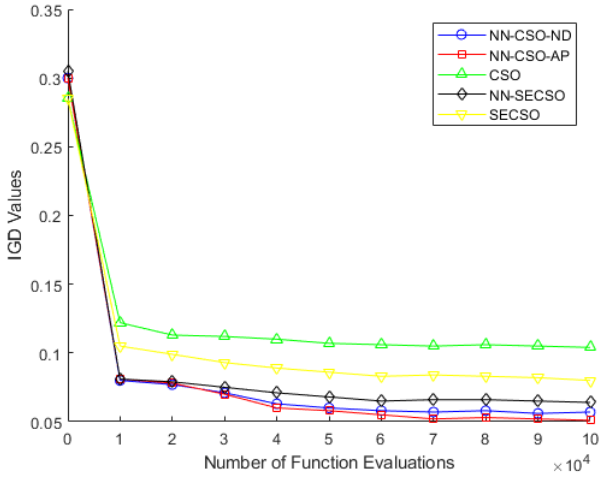

Figure 3. Convergence profiles of the five compared algorithms on LSMOP2 with two objectives and 100 decision variables.

From Fig. 4, some meaningful conclusions can be drawn. First, NN-SECSO, NN-CSO show faster convergence than their corresponding primitive CSOs (i.e., S-ECSO [18] and CSO), which confirms that the NN model can accelerate the search process of the primitive CSOs. Secondly, NN-CSO-AP and NN-CSO-ND could achieve the desired IGD levels at early evolutionary stages, while NN-CSO-AP showed better performance at later stages. In summary, we can conclude that NN-enhanced CSOs exhibit significant advantages in solving various LMOPs compared to the corresponding primitive CSOs, which is the same as the original paper. Meanwhile, this study verifies that among the two selection operators reproduced, the angular penalty selection strategy exhibits better scalability and diversity.

Table 1. Summary of the Comparison of Results Between NN-CSO-AP and its Competitors based on IGD Values

NN-CSO-AP vs LMOEAs	D	Test Benchmark Problems			
		on LSMOP +/-/=	on UF +/-/=	on DTLZ +/-/=	on WFG +/-/=
LSMOF	100	8/15/4	4/3/3	28/17/0	30/14/1
	200	19/22/4	5/4/1	20/11/4	23/20/2
	500	15/24/6	4/6/0	16/15/4	15/23/7
	1000	10/28/7	3/5/2	12/18/5	15/24/3
	ALL	**52/89/21**	**16/18/6**	**76/61/13**	**83/81/13**
DGEA	100	8/14/5	2/7/1	7/21/7	12/27/6
	200	7/23/15	3/7/0	9/21/5	14/24/7
	500	11/24/10	3/6/1	10/22/3	11/26/8
	1000	9/25/11	2/7/1	9/17/9	17/19/9
	ALL	**35/86/41**	**10/27/3**	**35/81/24**	**54/96/30**
LMEA	100	7/16/4	5/4/1	10/17/8	17/18/10
	200	10/20/15	2/7/1	8/22/5	14/23/8
	500	13/21/11	1/8/1	7/20/8	11/23/11
	1000	12/24/9	2/6/2	4/22/9	9/29/7
	ALL	**45/81/39**	**10/25/5**	**29/81/30**	**51/93/36**

5.3 The effectiveness of the reproduced NN-CSO-AP

Table 1 summarizes the statistical results of these three LMOEAs i.e., LSMOF [10], DGEA [9], and LMEA [21], versus NN-CSO-AP in solving four test problems. From these statistical results, it is clear that NN-CSO-AP outperforms its three competitors in different dimensions of the LSMOP and UF test suites. Moreover, NN-CSO-AP outperforms LSMOF, DGEA, and LMEA in most cases when solving the DLTZ and WFG test problems, while NN-CSO-AP is only slightly inferior to LSMOF when solving the DTLZ and WFG test problems (D = 100). When solving the LSMOP test problem, NN-CSO-AP outperforms its competitors in most cases. Therefore, it can be concluded that the reproduced NN-CSO-AP has a significant advantage over the these state-of-the-art LMOEAs in solving most of the LSMOP, DTLZ and WFG test problems. However the reproduced results are slightly worse than the original ones, which may be a difference due to the different operating environments.

6 Conclusion

In this replication study, a replication of the NN-CSO methodology, which is a hybrid of Competitive Swarm Optimization (CSO) and Neural Networks (NNs) for solving Large Scale Multi-Objective Optimization Problems (LMOPs), has been performed. With this re-implementation effort, the main objective is to validate the potential of NN-CSO in terms of optimization performance.

Experimental results confirm the effectiveness of NN-CSO-AP in LMOP benchmarking. Although no new methodology was introduced, this replication study emphasizes the enhanced potential of NN-CSO relative to traditional CSO algorithms. By replicating this combined model, the study demonstrates its ability to enhance the performance of traditional CSO algorithms. This replication work provides practical validation of the combination of evolutionary algorithms with edge computing, and provides future efforts to explore more

optimal avenues when dealing with complex problems.

References

[1] Ayodele Ariyo Adebiyi, Aderemi Oluyinka Adewumi, Charles Korede Ayo, et al. Comparison of arima and artificial neural networks models for stock price prediction. *Journal of Applied Mathematics*, 2014, 2014.

[2] Atilla Aslanargun, Mammadagha Mammadov, Berna Yazici, and Senay Yolacan. Comparison of arima, neural networks and hybrid models in time series: tourist arrival forecasting. *Journal of Statistical Computation and Simulation*, 77(1):29–53, 2007.

[3] Christopher Bennett, Rodney A Stewart, and Cara D Beal. Ann-based residential water end-use demand forecasting model. *Expert systems with applications*, 40(4):1014–1023, 2013.

[4] Ran Cheng and Yaochu Jin. A competitive swarm optimizer for large scale optimization. *IEEE transactions on cybernetics*, 45(2):191–204, 2014.

[5] Ran Cheng, Yaochu Jin, Markus Olhofer, et al. Test problems for large-scale multiobjective and many-objective optimization. *IEEE transactions on cybernetics*, 47(12):4108–4121, 2016.

[6] Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. Scalable multi-objective optimization test problems. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 1, pages 825–830. IEEE, 2002.

[7] Zong Woo Geem and William E Roper. Energy demand estimation of south korea using artificial neural network. *Energy policy*, 37(10):4049–4054, 2009.

[8] Qinghua Gu, Dejun Pang, and Qian Wang. Evolutionary many-objective algorithm with improved growing neural gas and angle-penalized distance for irregular problems. *Applied Intelligence*, pages 1–30, 2023.

[9] Cheng He, Ran Cheng, and Danial Yazdani. Adaptive offspring generation for evolutionary large-scale multiobjective optimization. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(2):786–798, 2022.

[10] Cheng He, Lianghao Li, Ye Tian, Xingyi Zhang, Ran Cheng, Yaochu Jin, and Xin Yao. Accelerating large-scale multiobjective optimization via problem reformulation. *IEEE Transactions on Evolutionary Computation*, 23(6):949–961, 2019.

[11] Simon Huband, Philip Hingston, Luigi Barone, and Lyndon While. A review of multiobjective test problems and a scalable test problem toolkit. *IEEE Transactions on Evolutionary Computation*, 10(5):477–506, 2006.

[12] Lingjie Li, Yongfeng Li, Qiuzhen Lin, Songbai Liu, Junwei Zhou, Zhong Ming, and Carlos A Coello Coello. Neural net-enhanced competitive swarm optimizer for large-scale multiobjective optimization. *IEEE Transactions on Cybernetics*, 2023.

[13] Miqing Li, Shengxiang Yang, and Xiaohui Liu. Shift-based density estimation for pareto-based algorithms in many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 18(3):348–365, 2013.

[14] Xiaodong Li and Xin Yao. Cooperatively coevolving particle swarms for large scale optimization. *IEEE Transactions on Evolutionary Computation*, 16(2):210–224, 2011.

[15] Qiuzhen Lin, Songbai Liu, Qingling Zhu, Chaoyu Tang, Ruizhen Song, Jianyong Chen, Carlos A Coello Coello, Ka-Chun Wong, and Jun Zhang. Particle swarm optimization with a balanceable fitness estimation for many-objective optimization problems. *IEEE Transactions on Evolutionary Computation*, 22(1):32–46, 2016.

[16] Jonathan L Ticknor. A bayesian regularized artificial neural network for stock market forecasting. *Expert systems with applications*, 40(14):5501–5506, 2013.

[17] Lin Wang, Yurong Zeng, Jinlong Zhang, Wei Huang, and Yukun Bao. The criticality of spare parts evaluating model using artificial neural network approach. In *Computational Science–ICCS 2006: 6th International Conference, Reading, UK, May 28-31, 2006, Proceedings, Part I 6*, pages 728–735. Springer, 2006.

[18] Xiangyu Wang, Kai Zhang, Jian Wang, and Yaochu Jin. An enhanced competitive swarm optimizer with strongly convex sparse operator for large-scale multiobjective optimization. *IEEE transactions on evolutionary computation*, 26(5):859–871, 2021.

[19] Guoqiang Zhang, B Eddy Patuwo, and Michael Y Hu. Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting*, 14(1):35–62, 1998.

[20] Qingfu Zhang, Aimin Zhou, Shizheng Zhao, Ponnuthurai Nagaratnam Suganthan, Wudong Liu, Santosh Tiwari, et al. Multiobjective optimization test instances for the cec 2009 special session and competition. 2008.

[21] Xingyi Zhang, Ye Tian, Ran Cheng, and Yaochu Jin. A decision variable clustering-based evolutionary algorithm for large-scale many-objective optimization. *IEEE Transactions on evolutionary Computation*, 22(1):97–112, 2018.