Neural Network-Based Information Transfer for Dynamic Optimization

Xiao-Fang Liu, *Student Member, IEEE*, Zhi-Hui Zhan, *Senior Member, IEEE*, Tian-Long Gu, Sam Kwong, *Fellow, IEEE*, Zhenyu Lu, Henry Been-Lirn Duh, and Jun Zhang, *Fellow, IEEE*

***Abstract*— In dynamic optimization problems (DOPs), as the environment changes through time, the optima also dynamically change. How to adapt to the dynamic environment and quickly find the optima in all environments is a challenging issue in solving DOPs. Usually, a new environment is strongly relevant to its previous environment. If we know how it changes from the previous environment to the new one, then we can transfer the information of the previous environment, e.g., past solutions, to get new promising information of the new environment, e.g., new high-quality solutions. Thus, in this paper, we propose a neural network (NN)-based information transfer method, named NNIT, to learn the transfer model of environment changes by NN and then use the learned model to reuse the past solutions. When the environment changes, NNIT first collects the solutions from both the previous environment and the new environment and then uses an NN to learn the transfer model from these solutions. After that, the NN is used to transfer the past solutions to new promising solutions for assisting the optimization in the new environment. The proposed NNIT can be incorporated into population-based evolutionary algorithms (EAs) to solve DOPs. Several typical state-of-the-art EAs for DOPs are selected for comprehensive study and evaluated using the widely used moving peaks benchmark. The experimental results show that the proposed NNIT is promising and can accelerate algorithm convergence.**

Manuscript received August 28, 2018; revised March 12, 2019 and May 28, 2019; accepted May 28, 2019. Date of publication July 19, 2019; date of current version May 1, 2020. This work was supported in part by the Outstanding Youth Science Foundation under Grant 61822602, in part by the National Natural Science Foundations of China (NSFC) under Grant 61772207, Grant 61873097, and Grant 61773220, in part by the Natural Science Foundations of Guangdong Province for Distinguished Young Scholars under Grant 2014A030306038, in part by the Guangdong Natural Science Foundation Research Team under Grant 2018B030312003, in part by the Guangdong-Hong Kong Joint Innovation Platform under Grant 2018B050502006, and in part by the Hong Kong GRF-RGC General Research Fund 9042489 (CityU 11206317). *(Corresponding authors: Zhi-Hui Zhan; Jun Zhang.)*

X.-F. Liu is with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China.

Z.-H. Zhan is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, and also with the State Key Laboratory of Subtropical Building Science, South China University of Technology, Guangzhou 510006, China (e-mail: zhanapollo@163.com).

T.-L. Gu is with the School of Computer Science and Engineering, Guilin University of Electronic Technology, Guilin 541004, China.

S. Kwong is with the Department of Computer Science, City University of Hong Kong, Hong Kong.

Z. Lu is with the School of Electronic and Information Engineering, Nanjing University of Information Science and Technology, Nanjing 210044, China.

H. B.-L. Duh is with the Department of Computer Science and Information Technology, La Trobe University, Melbourne, VIC 3086, Australia.

J. Zhang is with the College of Engineering and Science, Institute for Sustainable Industries and Liveable Cities, Victoria University, Melbourne, VIC 8001, Australia (e-mail: junzhang@ieee.org).

Digital Object Identifier 10.1109/TNNLS.2019.2920887

***Index Terms*— Dynamic optimization problem (DOP), information transfer, neural network (NN).**

I. Introduction

MANY real-world problems have uncertainties in objective functions, environmental parameters, constraints, or problem representation, resulting in the dynamic change of the optima through time [1]–[4]. These problems are called dynamic optimization problems (DOPs). Although evolutionary algorithms (EAs) have successfully solved many stationary problems [5]–[8], they still face challenges in solving DOPs due to their convergence nature. A converged population has lost exploration ability and is difficult to adapt to a new environment [9], [10]. Restarting the algorithm may be simple but not effective since it completely discards the optimization efforts in past environments. Usually, a new environment is strongly relevant to its previous environment, and thereby, the large number of solutions, especially the good solutions, obtained from the previous environment may be useful in the new environment. Therefore, the reuse of past solutions becomes a great potential to accelerate convergence toward the new optima in new environments.

In the literature, some efforts have been made to reuse past information for DOPs. They can be loosely categorized as three types, memory scheme [11], prediction model [12], [13], and transfer learning method [14]. Among them, memory scheme stores good solutions in memory and reintroduces special solutions into the population when necessary [11]. Instead, the prediction model uses the past solutions comprehensively and extracts the change pattern of the optima in past environments for predicting the new optima in new environments [15], [16]. This may be helpful if the change pattern is stable. However, in many real-world applications, the environment usually changes stochastically [14], and hence, the prediction model may be not applicable. To adapt to any change pattern, transfer learning methods directly consider the difference between a new environment and its previous one so as to construct a transfer model to relocate the past solutions for the new environment [14].

Recently, transfer learning methods have gained increasing attention. For example, a recent work of transfer learning in objective space adopts domain adaptation methods to transfer the past optimal objective values to their new values in new environments [14]. However, to the best of our knowledge,

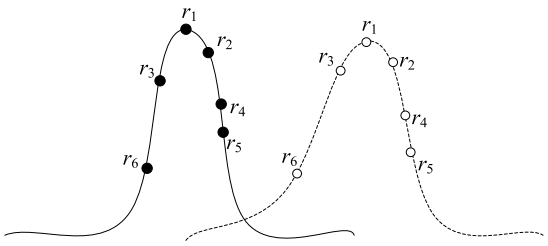


Fig. 1. Example of solution movements based on rankings in two successive environments. The solid dots and hollow dots are the sampled solutions in the previous environment and the new environment, respectively. r_k represents that the solution ranks kth in the corresponding environment.

there are still very few works that directly learn environment changes and transfer information in search space. Thus, how to learn the environment change from the already known data and further utilize the obtained environment change to transfer past solutions to new solutions is still important and challenging.

Indeed, in a new environment, as the fitness function changes, the rankings of solutions also change. Conversely, given a certain ranking, the corresponding solution moves to a different position in the new environment, e.g., the best solution moves to a new position, as shown in Fig. 1. Considering the solutions with the same rankings in two successive environments, the environment change presents as the movements of the solutions from a fixed ranking's angle. Hence, the environment change can be expressed as a transfer function that maps the solutions in an old (previous) environment to the solutions in a new environment. Thus, to learn the environment change, we can treat it in two steps: solution pairing between the two environments and function approximation for these solution mapping pairs. Because neural network (NN) theoretically is able to discover the functions among the given data to any degree of accuracy [17]–[21], in this paper, we propose to adopt the NN to learn the transfer function of environment changes and then use it to reuse past solutions. NN is selected rather than others such as traditional statistical regression methods since NN is a universal approximator and does not require the data to conform to some assumptions, such as linear relationship. Moreover, NN has strong robustness and fault tolerance. Since the relationship of data is unknown and there are noises in the data, NN is a good choice for transfer model learning. Following this idea, an NN-based information transfer method, named NNIT, is developed. In NNIT, we collect the solutions from both the previous and the new environments and then pair these solutions to construct data for NN training. After the NN has been trained, the good solutions in the previous environment are reused to obtain new solutions. Practically, the proposed NNIT can be incorporated into any EA to solve DOPs. To test the effect of the NNIT, several typical state-of-the-art algorithms are selected to work with the NNIT and the resultant algorithms are evaluated on 28 instances of the widely used moving peaks benchmark (MPB). The experimental results show that the proposed NNIT method is promising.

The rest of this paper is organized as follows. In Section II, we present a review of the studies on DOPs and give a brief introduction of NN. In Section III, we outline our proposed NNIT procedure in detail. Section IV further discusses the behavior of NNIT. The experimental studies are shown in Section V. Finally, the conclusion is drawn in Section VI.

II. Background

A. Related Work of DOPs

As mentioned earlier, traditional EAs face challenges on DOPs due to the loss of population diversity in dynamic environments. In addition, the plentiful information of past environments may be useful for the optimization of new environments. Therefore, in the literature, the work of DOPs mainly focuses on diversity increase and information reuse. They can be classified into five types: diversity maintenance, multipopulation method, memory scheme, prediction model, and transfer learning method.

1) Diversity Maintenance: To handle the diversity loss of the population in new environments, there are two common ways: increasing population diversity once the environment changes or maintaining population diversity all the time. First, to increase diversity when the environment changes, some researchers propose restarting techniques that reinitialize some part of or the whole population. For example, Yang [22] and Yang and Li [23] proposed to use one newly generated population to search new areas; Woldesenbet and Yen [24] relocated solutions according to the rough relationship between fitness changes and variable sensitivities obtained in the previous environments. Second, different methods have been developed to maintain population diversity along the whole evolutionary process. For example, diversity is continuously evaluated and will be increased if it decreases below the predefined threshold [25]; charged particle [26], composition particle [27], and grid topology [28] are developed to maintain the local diversity of the swarm; particles are migrated among swarms according to repulsive diversity [29].

2) Multipopulation Method: To track the multiple moving peaks, a multipopulation method is proposed. A parent population is in charge of exploring new peaks and child populations are splitting from the parent population to exploit the found peaks [30], [31]. Instead of the splitting idea, the populations can also be created by the division of a large population and different populations exploit different areas. To divide a large population, different techniques, such as speciation-based methods [32], niching by dot product of the individuals [33], and clustering methods [23], [34], [35], can be adopted. Additionally, these methods require to determine the number of populations to create. To adapt the number of populations to dynamic environments, multiple control strategies are developed based on historical experience and heuristic information [36], [37]. Some researchers also proposed to generate a fixed number of populations at the beginning and create new populations afterward if necessary [38], [39].

3) Memory Scheme: Memory scheme stores the solutions implicitly in redundant individuals [40] or explicitly in an external archive [41] and reuses them later if necessary. Explicit memory is commonly used. It considers two parts: solution selection for storage and solution reuse. Solutions are periodically chosen based on their fitness values, age, and

diversity [42], [43]. The selected solutions can be directly stored as special independent solutions [42] or used to construct a probability model [11]. When the environment changes, the solutions can be reintroduced into the population [44]–[46] or used for local search to find promising solutions [47]. Even during the evolutionary process in an environment, the solutions can also be used to increase population diversity [48].

4) Prediction Model: Since the consecutive environments are often correlative, the change pattern of the environment can be extracted from the past information to predict new changes. For example, Rossi *et al.* [15] adopted the Kalman to model the movement of the optima and predict the possible optima in new environments. Similarly, Simões and Costa [16] used linear regression to estimate the time of the next environment change and adopted Markov chains to predict new optima according to the optima found in past environments. Likewise, Zhou *et al.* [49] employed the center points of Pareto sets in past environments as data and adopted regression model to simulate the change pattern of the center points. These methods have shown that the extracted knowledge from past environments can provide effective guidance for the optimization of new environments.

5) Transfer Learning Method: Considering the correlation and difference between the previous environment and the new one, a transfer learning method uses the data from these two environments to directly construct a transfer model of solutions/fitness. Based on the idea that the solutions of different environments are often in different distributions, Jiang *et al.* [14] proposed to adopt the transfer component analysis technique to construct a transfer model in objective space and then transferred the optimal objective values in past environments to new values in new environments. However, this paper only models the change in objective space and requires an additional optimization method to obtain the corresponding solutions in the search space. It is still at issue to directly transfer information in search space.

B. Neural Network

NN is a simple abstraction of biological NN that stores function in the neurons and in the connections between them [50]. It learns to perform useful functions by training on data. Particularly, a multilayer feedforward NN is a universal approximator and is adopted in this paper. Mathematically, the NN is a function that maps a domain R^D to another domain R^N, where D and N are the numbers of dimensions of the domains [51]. When perceiving environment changes, it inputs the solutions from the previous environment and outputs the solutions from the new environment. The basic structure of NN is shown in Fig. 2. It includes multiple layers each with multiple neurons and connections between them. The leftmost layer uses the input vector as an input and is named the input layer, the rightmost layer whose output is the NN output is named the output layer, and the other layers are called hidden layers. In the hidden layers and output layer, each neuron has a bias b and an activation function φ. Each connection between the neurons has a weight w.

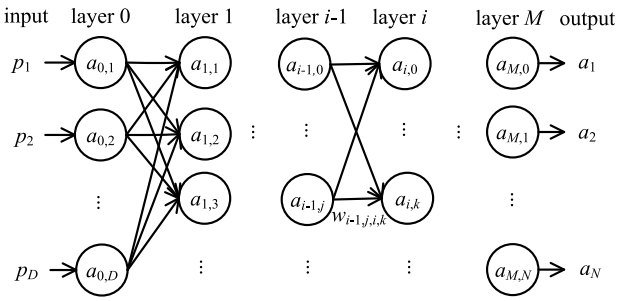


Fig. 2. Basic structure of NN.

Given an input $\mathbf{p} = [p_1, \ldots, p_D]$, the output of the kth neuron in the input layer is as

$$a_{0,k} = p_k \tag{1}$$

The output of the input layer becomes the input of the following layer. For the kth neuron in the following layer i ($i \geq 1$), the weighted inputs are summed with the bias and then are transferred by activation function φ to form the net output $a_{i,k}$ as

$$a_{i,k} = \varphi\left(\sum_j w_{i-1,j,i,k} a_{i-1,j} + b_{i,k}\right) \tag{2}$$

where $w_{i-1,j,i,k}$ is the weight of the connection between the jth neuron of layer $i-1$ and the kth neuron of layer i, $b_{i,k}$ is the bias of the kth neuron of layer i, and the activation function φ can be any linear or nonlinear function chosen according to the problem to solve. The output of the output layer M becomes the network output $\mathbf{a} = [a_1, \ldots, a_N]$ as

$$a_k = a_{M,k} \tag{3}$$

Before being a qualified approximator, the NN needs to go through a full training on the given data to get the optimal parameters. Generally, the data for NN training, called training set, often include multiple training samples each having an input vector and a target output. The most popular back-propagation techniques are commonly used to train the NN. After training, the NN has learned the function and can predict new data.

III. Neural Network-Based Information Transfer For Dynamic Optimization

In this section, the motivations of developing the NNIT are first introduced, followed by the complete procedure of the NNIT. Then, the details of the NNIT are presented, including the transfer function learning by NN and the information reuse by NN.

A. Motivations of NNIT

The first motivation of the NNIT is the strong correlation between a new environment and its previous one. Given abundant solutions obtained in the previous environment, once we know how the new environment is changed from the previous one, we can relocate these solutions to obtain promising solutions in the new environment. Thus, learning

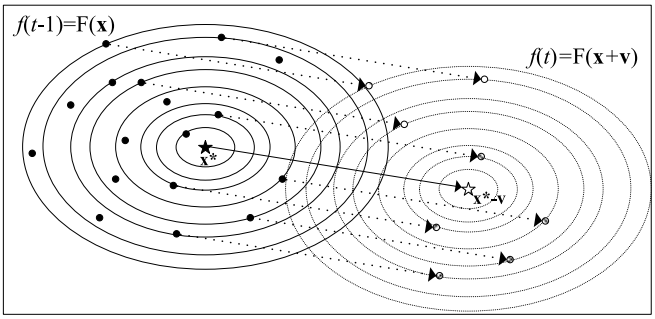


Fig. 3. Example of environment change. The function is $f(t1) = F(\mathbf{x})$ at time t 1 and is changed to $f(t) = F(\mathbf{x} + \mathbf{v})$ at time t. The solid dots and black star are the solutions found at time t 1, while the hollow dots are the known solutions at time t. The dotted lines with arrow represent the movements of the solutions from time t 1 to time t, which can be used to learn the transfer model of the environment change, i.e., the solid line with arrow connected the two stars means that x is moved to $\mathbf{x}-\mathbf{v}$ in the new environment t.

the transfer function of the environment change can help accelerate convergence in new environments. An example is shown in Fig. 3, where the objective function $f(t-1) = F(\mathbf{x})$ is changed to $f(t) = F(\mathbf{x} + \mathbf{v})$ at time t. To relocate the past solutions, we need to discover the transfer function of the environment change $\mathbf{x}' = \mathbf{x}\text{-}\mathbf{v}$, where $\mathbf{x}$ is a solution in the previous environment and $\mathbf{x}'$ is the corresponding position that $\mathbf{x}$ is moved to in the new environment.

The second motivation of the NNIT is that NN has strong function approximation and generalization ability. On the one hand, the environment change may be complex and cannot be exactly formulated. On the other hand, the data observed in the two environments are incomplete and often have noise. Hence, it is hard to construct an accurate mathematical model to represent the environment change. However, the NN can be taught to recognize the function among data with noise and used to learn the transfer function of the environment change.

Therefore, this paper aims to develop the NNIT to perceive environment changes and reuse past information, for faster convergence of the optimization in new environments.

B. NNIT

When the environment changes, the NNIT is performed. In NNIT, NN first perceives the change between the new environment and the previous one and then reuses the past solutions to generate promising solutions to assist the optimization of the new environment. The complete procedure is shown in Fig. 4. First, solutions are collected from the previous and the new environments. Second, the solutions of the two environments are paired to construct data. Third, the NN is designed and trained to learn the transfer function of the environment change. Fourth, the trained NN reuses the solutions in the previous environment to generate new solutions. Finally, the new solutions are added to the population to assist EAs. The details of the NNIT will be described in Sections III-C and III-D. Note that the new environment is also indicated as the current environment in the following.

C. Transfer Function Learning by NN

Assume that the current environment is indexed by i ($i \geq 2$). The solutions are first collected from

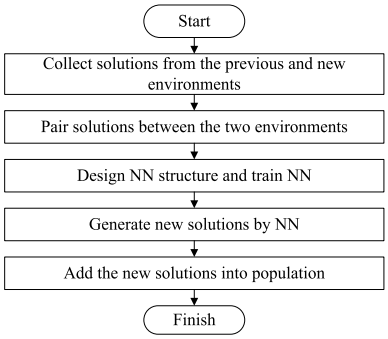


Fig. 4. Flowchart of the proposed NNIT.

environments $i-1$ and i and then are paired to construct data. The data will next be used by NN to learn the environment change.

1) Collect Solutions from the Previous and Current Environments: Assume that all the solutions found in the previous environment $i-1$ (i.e., all the solutions in all the generations) are stored in set S_{i-1}. To preprocess these large numbers of solutions, they are divided into multiple clusters such that each cluster contains the solutions of a special subarea. For this purpose, we first denote some local optimal solutions (will be described later) to act as cluster seeds and form a set $L_{i-1} = \{B^1_{i-1}, \ldots, B^M_{i-1}\}$, where M is the number of clusters. Next, each solution in S_{i-1} finds the nearest cluster seed based on the Euclidean distance and then joins the corresponding cluster. Finally, M clusters $C^1_{i-1}, \ldots, C^M_{i-1}$ are formed. The procedure is presented in Algorithm 1. Note that, when incorporating NNIT into a multipopulation EA, since each subpopulation searches a special subarea, the best solutions in all the subpopulations are directly used to form L_{i-1}. Herein, the number of clusters M is the same as the number of local optimal solutions. Likewise, when incorporating NNIT into a single-population EA, the solutions of the final population can be clustered first, and then, the best solutions in all clusters are collected into L_{i-1} or the final population directly forms the L_{i-1}.

Algorithm 1 Clustering Procedure for the Collected Solutions (S, L)

Input: solution set $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, cluster seeds $L = \{B^1, \ldots, B^M\}$
Output: clusters $C^1, \ldots, C^M$
Begin
1. **For** each cluster C^j **Do**
2. $C^j = \Phi$
3. **End for**
4. **For** each solution $\mathbf{x}_a \in S$ **Do**
5. find the nearest cluster seed B^k in L based on Euclidean distance
6. $C^k = C^k \cup \mathbf{x}_a$
7. **End for**

End

For the current environment i, the solutions of the initial population (i.e., the first generation) are stored in set I_i.

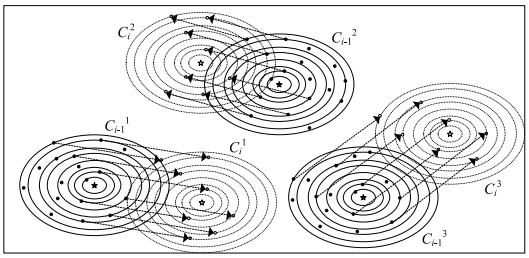


Fig. 5. Example of solution pairing in a DOP with three peaks (subareas). The solid dots are the solutions that form clusters C_{i-1}^1, C_{i-1}^2, and C_{i-1}^3 of environment i-1, and hollow dots are the solutions that form clusters C_i^1, C_i^2, and C_i^3 of environment i.

Since the new local optima of environment i are unknown but may be likely near to the local optima of the previous environment $i-1$, the cluster seeds in L_{i-1} of the previous environment are adopted. Using these cluster seeds, clustering procedure Algorithm 1 is performed on I_i to get M clusters $C_i^1, \ldots, C_i^M$. The obtained clusters represent the solutions found in each subarea.

2) Pair Solutions Between the Two Environments: As discussed earlier, the environment change presents as the movements of the solutions. Hence, the next step is to pair the solutions between the two environments to represent the movements of the solutions. These solution pairs will form multiple training samples each having an input vector of one solution from the previous environment i-1 and a target of its corresponding solution that has been moved to in environment i.

To reduce the difficulty of solution pairing, we pair solutions in each subarea. As the clusters in environments $i-1$ and i (e.g., C_{i-1}^1 and C_i^1) are formed by the same cluster seed, they are likely in the same subarea before and after change. Therefore, we pair solutions between the clusters formed by the same cluster seed from the two environments. An example is illustrated for a DOP with three peaks in Fig. 5. The solutions in C_i^1 are paired with those in C_{i-1}^1. Similarly, the solutions in C_i^2 and C_i^3 are paired with those in C_{i-1}^2 and C_{i-1}^3, respectively.

Then, given two clusters C_i^j and C_{i-1}^j $(1 \le j \le M)$, how to pair the solutions in them? Since the fitness function may be scaled, fitness values are not reliable to determine whether two solutions are the ones before and after change/movement. Indeed, it is the sameness of their rankings in the subareas that says the movement relationship. However, since only limited solutions are known in the subareas (i.e., the cluster only includes some parts of the solutions in the subarea), the rankings of the solutions in the clusters are not consistent with their real rankings among all the solutions in the whole subareas. Thus, in this paper, we propose to use fitness normalization to reflect their relative rankings in subareas and then pair solutions by the similarity of their normalized fitness.

First, we take the fitness values of the local optimum and the available worst solution in the subarea as lower and upper boundaries to normalize the fitness of each solution. For the two clusters C_i^j and C_{i-1}^j, the solutions are sorted based on their fitness values, and the minimum and maximum fitness values are recorded as $f_{i,\min}^j$ and $f_{i,\max}^j$ for cluster C_i^j and as $f_{i-1,\min}^j$ and $f_{i-1,\max}^j$ for cluster C_{i-1}^j. Especially, in a maximization problem, for the cluster C_i^j of environment i, since the new local optimum has not been found and only a few solutions are known, the minimum and maximum fitness values among them are not equal to those for the subarea and hence are scaled as

$$f_{i,\min}^j = \min\left\{f_{i,\min}^j, f_{i-1,\min}^j\right\}, \quad \forall 1 \le j \le M \tag{4}$$

$$f_{i,\max}^j = f_{i,\max}^j + \left(f_{i,\max}^j - f_{i,\min}^j\right) \times F, \quad \forall 1 \le j \le M \tag{5}$$

where F is a scale factor that controls the extension of the upper bound since it should be larger than the maximum value found in the cluster. Similarly, the minimum value in the subarea must not be larger than the minimum fitness value in the cluster and may be closed to that in the previous environment. Note that the situation is similar to a minimization problem. After that, the fitness value of each solution $\mathbf{x}$ in the clusters is normalized by

$$f_n(\mathbf{x}) = \frac{f(\mathbf{x}) - f_{l,\,\min}^j}{f_{l,\,\max}^j - f_{l,\,\min}^j}, \ \forall \mathbf{x} \in C_l^j, \ 1 \le j \le M, \ l = i-1 \vee i \tag{6}$$

where $f(\mathbf{x})$ is the fitness value of a solution $\mathbf{x}$ and $f_n(\mathbf{x})$ is its normalized fitness value.

Later, each solution $\mathbf{q}$ in cluster C_i^j of environment i selects the solution $\mathbf{p}$ with the closest normalized fitness value to itself from the cluster C_{i-1}^j of environment i-1 for pairing by

$$\mathbf{p} = \arg\min_{\mathbf{x} \in C_{i-1}^j} |f_n(\mathbf{x}) - f_n(\mathbf{q})| \tag{7}$$

and then, this solution pair forms a training sample $(\mathbf{p}, \mathbf{q})$, where $\mathbf{p}$ is an input and $\mathbf{q}$ is the target output. Note that, for the training samples constructed from the same cluster, their inputs will keep the same relative rankings as their outputs. Finally, the training samples form a training set $\mathrm{T} = \{(\mathbf{p}_1,\mathbf{q}_1), \ldots, (\mathbf{p}_{|T|}, \mathbf{q}_{|T|})\}$ for learning the transfer function, where $|\mathrm{T}|$ is the number of training samples. The procedure is described in Algorithm 2.

3) Design NN Structure: With the training set, NN is then used to learn the transfer function of the environment change between environments i-1 and i. For this purpose, we need to choose appropriate network architecture, i.e., number of layers, number of neurons, and activation function. Assume that the number of dimensions of decision space is D, and then, there will be D elements in the input and target output. Herein, we use D 3-layer networks each for a different dimension of the target output. Each NN has one input layer with D neurons, one hidden layer, and one output layer with one neuron. In this way, each NN not only has lower complexity compared with one single NN with D output elements but also can be trained in parallel to reduce training time. For a training sample in T, all the D NNs take the input vector as the input, and then, the outputs of the D NNs will form a complete output vector. The number of neurons in the hidden layer S_{hidden} is

Algorithm 2 Solution Pairing Procedure

Input: clusters $C_{i-1}^1, \ldots, C_{i-1}^M$ of environment $i-1$ and clusters $C_i^1, \ldots, C_i^M$ of environment i
Output: Training set T
Begin
1. T = Φ
2. **For** $j = 1$ to M **Do**
3. **If** $C_{i-1}^j \neq \Phi \wedge C_i^j \neq \Phi$
4. Sort C_{i-1}^j from worst to best and record $f_{i-1,\min}^j$ and $f_{i-1,\max}^j$
5. Sort C_i^j from worst to best and record $f_{i,\min}^j$ and $f_{i,\max}^j$
6. Update $f_{i,\min}^j$ and $f_{i,\max}^j$ using Eq. (4) and (5)
7. Normalize the fitness of the solutions in C_{i-1}^j and C_i^j by Eq. (6)
8. $k = 0, \upsilon = 0$
9. **For** each $\mathbf{q} \in C_i^j$ **Do**
10. **If** $|C_{i-1}^j| < |C_i^j|$
11. $l = 1, r = |C_{i-1}^j|$
12. **Else**
13. $l = \upsilon + 1, r = |C_{i-1}^j| - |C_i^j| + k + 1$
14. **End if**
15. $\mathbf{p} = \arg\min_{\mathbf{x}_a \in C_{i-1}^j \wedge l \le a \le r} |f_n(\mathbf{q}) - f_n(\mathbf{x}_a)|$
16. υ = the index of $\mathbf{p}$ in C_{i-1}^j
17. T = T∪{(**p**, **q**)}
18. $k = k + 1$
19. **End for**
20. **End if**
21. **End for**
End

determined following the suggestion in [52] by

$$S_{\text{hidden}} = \max\left\{\frac{|\mathrm{T}|}{(S_{\text{input}} + S_{\text{ourput}}) \times 10}, 1\right\} \tag{8}$$

where |T | is the number of training samples in training set T, and $S_{\text{input}} = D$ and $S_{\text{output}} = 1$ are the number of neurons in input and output layers, respectively. The activation function φ for the neurons in the hidden layer is a tan-sigmoid function [52] as

$$\varphi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{9}$$

and φ for the neurons in the output layer is a linear function [52] as

$$\varphi(x) = x \tag{10}$$

4) Train NN: Before training, the training set of each NN is first constructed. For the dth NN, its training set T_d has the same inputs as the training set T but only takes the dth dimension of the target outputs of T as target outputs. The performance indicator for the network training (training error) is the mean-squared error over the output targets in the training set T_d

$$E = \frac{1}{Q}\sum_{q=1}^{Q}\mathbf{e}_q^T\mathbf{e}_q = \frac{1}{Q}\sum_{q=1}^{Q}(\mathbf{t}_q - \mathbf{a}_q)^T(\mathbf{t}_q - \mathbf{a}_q) \tag{11}$$

where Q is the number of samples in the training set, $\mathbf{e}_q$ is the error for the qth input/target pair, $\mathbf{t}_q$ is the target output for the qth input, and $\mathbf{a}_q$ is the network output for the qth input.

Each NN adopts the typical Levenberg–Marquardt algorithm [53] with 30 iterations for training. The Levenberg–Marquardt algorithm is adopted since it can quickly find good parameters [52]. This can reduce training time, being promising for dynamic optimization.

D. Information Reuse by NN

1) Generate New Solutions: Once the NN has finished training, it has learned and stored the transfer function of the environment change in itself. The NN can relocate the solutions found in the previous environment to obtain new solutions. Herein, we take the best solution found in each subarea (stored in L_{i-1}) as NN inputs and use the NN outputs to form new solutions.

2) Add New Solutions Into Population: The obtained new solutions can be added to the initial population of EA. Especially, if the NNIT is incorporated into a single-population EA, the new solutions are added to the population directly. However, when incorporating NNIT into a multipopulation EA, each solution calculates its distances to the best solution in each subpopulation and then selects the closest subpopulation to add. After a solution has been added, no matter the EA is a single-population or a multipopulation algorithm, the population size control mechanism of the EA is performed. Note that if there is no any population size control strategy in the EA, then the worst individual in the population is deleted to avoid a rapid increase in population size.

IV. Discussion of NNIT

In this section, we discuss the scale factor F and NN structure in the proposed NNIT. In order to analyze NNIT's ability of learning transfer function, we adopt the famous and widely used MPB [43] as the test example and implement the NNIT using the data constructed from the MPB instances.

A. Test Problem

The MPB [43] has multiple peaks, and the height, width, and position of each peak dynamically change every time the environment changes. The function is defined as

$$F(\mathbf{x}, t) = \max_{i=1,.,P} \frac{H_i(t)}{1 + W_i(t)\sum_{j=1}^{D}(x_j(t) - X_{ij}(t))^2} \tag{12}$$

where $\mathbf{x} = [x_1, \ldots, x_D]$ is a vector in the D-dimensional search space, P is the number of peaks, and $H_i(t)$ and $W_i(t)$ are the

TABLE I
SPECIATION OF MPB

Parameters	Value
Number of peaks (P)	10, 20, 30, 40, 50, 80, 100, 120, 150, 200
Change of frequency (u)	5000
Number of environments (C)	100
Height severity (s_H)	[1, 10]
Width severity (s_W)	[0.1, 1.0]
Peak shape	cone
Basic function	no
Correlated parameter (λ)	0
Shift length (s_X)	1, 2, 3, 4, 5, 6
H	[30, 70]
W	[1, 12]
Number of Dimensions (D)	5

height and width of the ith peak at time t, respectively, and are changed by a random Gaussian variable σ as

$$H_i(t) = H_i(t-1) + s_H \times \sigma \tag{13}$$

$$W_i(t) = W_i(t-1) + S_W \times \sigma \tag{14}$$

$$\sigma = \text{Gaussian}(0, 1) \tag{15}$$

where s_H and s_W are the severity parameters of the height and width, respectively, and $\mathbf{X}_i(t) = [X_{i1}, \ldots, X_{iD}]$ is the position of the ith peak and is shifted with a random vector $\mathbf{v}$ of length s_X as

$$\mathbf{X}_i(t) = \mathbf{X}_i(t-1) + \mathbf{v}_i(t) \tag{16}$$

$$\mathbf{v}_i(t) = \frac{s_X}{|(1-\lambda)\times\mathbf{r} + \lambda \times \mathbf{v}_i(t-1)|}((1-\lambda)\times\mathbf{r} + \lambda \times \mathbf{v}_i(t-1)) \tag{17}$$

where $\mathbf{r}$ is a random vector, and λ is the correlated parameter and is set as 0, representing that the peak movements of different environments are uncorrelated. The shift length s_X controls the change severity of the environment. The speciation setting of the MPB problem is listed in Table I. Multiple MPB instances can be constructed by setting the number of peaks (P) and shift length (s_X) as different values.

In this section, we construct ten MPB instances for test, where the number of peaks P is set as 10, 30, 50, 150, and 200, and the shift length s_X is set as 2 and 5 following [37]. The change frequency is set as 5000 in all instances, which means that the maximum function evaluation (FE) time is 5000 in each environment.

B. NNIT Implementation Using Sampled Data of MPB

To give a comprehensive analysis, we do not integrate the NNIT with a special algorithm to get data but construct data according to the problem instances. In each MPB instance, each peak is considered as a subarea and its position is a local optimum. The local optima are collected in L_i for environment i. For each environment i ($i \geq 2$), we construct two solution sets S_{i-1} and I_i as follows.

1) Set S_{i-1} of the Previous Environment: Since S_{i-1} collects the solutions found by EA in the previous environment i-1, it is constructed before the environment change. According to the gradually convergence behavior of EA, S_{i-1} must include four parts: initial random solutions (rs) in the whole search space, the past local optima of its last environment i-2 (plo), multiple solutions with different distances to each peak (ps), and the local optima found (clo) for each peak (solutions in L_{i-1}). Particularly, rs and plo simulate the initial population, while clo and ps simulate the evolutionary process of the EA toward the local optima. Especially, ps is the position randomly distributed around each peak in different radii and they are constructed as follows. First, we define a maximum radius r_k of each peak k. We calculate the Euclidean distances d between any two peaks in environment i-1, and each peak k finds its closest peak and defines the half of their distance as r_k. Second, the number of solutions n for each peak is defined. Assume that the change frequency of the problem is u, and then, $n = (u - n_{\text{rs}} - n_{\text{plo}} - n_{\text{clo}})/P$, where $n_{\text{rs}} = 100$, $n_{\text{plo}} = P$, and $n_{\text{col}} = P$ are the number of the solutions of the other three parts. Finally, we define n values in the range of $(0, r_k]$ with a step size of r_k/n and then randomly generate n positions with these distances to each peak.

2) Set I_i for the Current Environment i: I_i often includes the initial population that is worse than the new optima in environment i. We construct it with two parts: the past local optima in L_{i-1} and four solutions around each peak. For each peak, we first select four positions, which are more than s_X away from the old peak position, sequentially from ps in S_{i-1} and then take their moved positions in environment i as the four solutions. Thus, all the solutions are more than s_X away from the new local optima.

After that, the two solution sets S_{i-1} and I_i are clustered according to Algorithm 1 using the positions in L_{i-1} as clustering seeds, and then, we construct a training set $\text{T} = \{(\mathbf{p}_1, \mathbf{q}_1), \ldots, (\mathbf{p}_Q, \mathbf{q}_Q)\}$ according to Algorithm 2, where Q is the size of T. Additionally, to test the prediction ability of the NNs, a test set is constructed as $\text{U} = \{(\mathbf{p}_1, \mathbf{t}_1), \ldots, (\mathbf{p}_q, \mathbf{t}_q), \ldots, (\mathbf{p}_V, \mathbf{t}_V)\}$, where $\mathbf{p}_q$ is a position from L_{i-1}, $\mathbf{t}_q$ is the corresponding moving position from L_i, and V is the size of set U. Next, we determine the number of neurons of hidden layers of the D NNs and train the NNs. Later, we use the NNs to predict the test set.

To evaluate the perception ability of the NNIT, we adopt two indicators: training error E_T and prediction error E_P over all environments as

$$E_T = \frac{1}{C-1}\sum_{i=2}^{C}\left(\frac{1}{Q}\sum_{q=1}^{Q} e_q\right) \tag{18}$$

$$E_P = \frac{1}{C-1}\sum_{i=2}^{C}\left(\frac{1}{V}\sum_{q=1}^{V} e_q\right) \tag{19}$$

$$e_q = ((\mathbf{t}_q - \mathbf{a}_q)^{\text{T}}(\mathbf{t}_q - \mathbf{a}_q))^{1/2} \tag{20}$$

where $\mathbf{a}_q$ is the NN output and $\mathbf{t}_q$ is the target output for an input $\mathbf{p}_q$. In addition, the estimation error E_f of the optimal objective value (fitness of the local optimum) in (5)

TABLE II

PERFORMANCE OF NNIT WITH DIFFERENT SCALE FACTOR F IN TERMS OF ESTIMATION ERROR OF THE OPTIMAL OBJECTIVE VALUE E_f, TRAINING ERROR E_T, PREDICTION ERROR E_P, AND TRAINING TIME

s_x	P	$\lvert S_{i-1}\rvert$	$\lvert I_i\rvert$	Scale factor F																	Time
				0			0.025			0.05			0.075			0.10			0.125			
				E_f	E_T	E_P	E_f	E_T	E_P	E_f	E_T	E_P	E_f	E_T	E_P	E_f	E_T	E_P	E_f	E_T	E_P	(s)
2	10	5000	50	12.02	1.11	**1.22**	7.97	1.05	2.07	**6.06**	0.99	2.99	6.42	0.97	3.97	8.41	**0.93**	4.92	11.43	0.95	4.92	1.99
	30	5000	150	12.90	1.44	1.61	10.22	**1.37**	**1.25**	7.90	1.52	1.29	6.39	1.59	1.44	**5.87**	1.69	1.58	6.30	1.77	1.98	4.12
	50	5000	250	12.72	**1.81**	1.90	10.64	1.82	1.71	8.70	1.86	1.62	7.17	2.02	**1.60**	6.16	2.08	1.65	**5.72**	2.20	1.75	6.20
	150	5000	750	12.74	2.43	2.09	11.5	2.46	2.06	10.28	2.48	1.99	9.14	2.53	1.95	8.13	2.57	1.91	**7.30**	2.63	**1.90**	16.91
	200	5000	1000	12.82	2.70	2.13	11.65	**2.70**	2.09	10.49	2.72	2.04	9.41	2.76	2.01	8.43	2.80	1.99	**7.61**	2.86	**1.95**	22.36
5	10	5000	50	28.91	2.69	**3.50**	25.03	2.44	3.81	21.36	2.30	4.77	18.46	2.23	5.80	16.58	2.18	7.33	**15.59**	**2.17**	7.98	1.98
	30	5000	150	31.05	**2.99**	3.58	28.81	3.18	3.43	26.58	3.31	3.32	24.42	3.45	**3.31**	22.43	3.43	3.32	**20.65**	3.52	3.42	4.14
	50	5000	250	30.15	**4.12**	4.39	28.5	4.17	4.26	26.86	4.17	4.04	25.23	4.42	4.13	23.68	4.36	**3.95**	**22.24**	4.54	4.02	6.28
	150	5000	750	29.03	**5.34**	5.08	28.18	5.39	5.07	27.33	5.37	4.98	26.49	5.44	4.96	25.66	5.44	4.90	**24.85**	5.47	**4.85**	16.96
	200	5000	1000	29.14	**5.60**	5.14	28.34	5.64	5.13	27.55	5.66	5.07	26.75	5.71	5.06	25.97	5.73	4.99	**25.20**	5.77	**4.99**	22.42

where $|S_{i-1}|$ and $|I_i|$ indicate the size of the data sets S_{i-1} and I_i used in NNIT in each environment i.

is calculated as

$$E_f = \frac{1}{C-1}\sum_{i=2}^{C}\left(\frac{1}{P}\sum_{j=1}^{P}\left|f_{i,\max}^{j} - f_{i,\text{peak}}^{j}\right|\right) \tag{21}$$

where $f_{i,\max}^{j}$ and $f_{i,\text{peak}}^{j}$ are the estimated value and real fitness value of the jth peak position in environment i, respectively.

C. Scale Factor F in NNIT

The scale factor F in (5) is used to estimate the optimal objective values in new environments for normalization. To investigate its effect, we test F in the range of [0, 0.125] with a step size of 0.025. The performance is measured by four indicators: estimation error of the optimal objective value E_f, training error E_T, prediction error E_P, and training time (from solution collection to the finishing of NN training).

The results are reported in Table II. Since the training time is mostly related to the size of the two solution sets, we only report the time obtained by $F = 0$ as an example due to space limit. We can see that when the size of the two data sets S_{i-1} and I_i increases to 5000 and 1000, the required time is still less than 23 s. Note that the experiments are implemented in C++ language and ran on a computer with Intel Core i7-7700 CPU (3.60 GHz), 8-GB memory, and Ubuntu 16.04.2. Since the reported training time has included the most expensive parts in NNIT, solution collection, solution pairing, and NN training, it shows that the proposed NNIT can perform in a reasonable time.

From Table II, we can see that different F values have different performances on different instances, and the value of 0.05 generally performs well.

As for the estimation error E_f in the normalization level, we can see that a relatively medium value in the range of [0.05, 0.10] for F is better for few peaks (10 and 30 peaks) and small shift length ($s_X = 2$), while a large value 0.125 is preferred for a large number of peaks (50, 150, and 200 peaks) and large shift length ($s_X = 5$). This is because when the environment drastically changes, the solutions are farther away from the new optima and, hence, the error of the known maximum value to the real optimum objective is larger, and thereby, larger F can scale better. On the contrary, a relatively medium value can scale well on the instances with small fluctuation.

For the training error E_T and prediction error E_P, it is interesting to find that, as F increases, E_T decreases on 10 peaks but increases on 50, 150, and 200 peaks. However, E_P presents different changing trends. As F increases, E_P increases on 10 peaks; decreases first but increases later on 30 and 50 peaks; while continuously decreasing on 150 and 200 peaks. Thus, to obtain good prediction ability, as the number of peaks increases, the required F value increases. However, since the medium value of 0.05 can get relatively medium E_P values on all the tested instances and E_P even is less than s_X on seven out of ten instances, 0.05 is a promising value for F.

Taking E_f, E_T, and E_P together, we can see that the changing trends of E_f and E_T may be not consistent with the prediction error E_P. The reason is on two sides. On the one hand, E_f only shows the error of the upper bound for normalization, while the prediction error E_P also depends on the training set and NN training at a higher level. On the other hand, there is noise in the data, and large E_T but small E_P are preferred since they show that the NN has good generalization to learn the environment change. Overall, E_P comprehensively shows NNIT's perception ability for environment changes and is the indicator that we most concern about.

In general, the value of 0.05 performs stable and can get good performance on most instances. Therefore, we set the scale factor F as 0.05 in the following experiments.

D. Comparisons of Different NN Structures in NNIT

In the NNIT, we use D three-layer NNs each for one different output dimension. In this section, we compare the adopted NNs with one single three-layer NN with D elements in the output layer. Note that only the three-layer network structure is used for comparisons since a simple three-layer network can perform well. In the compared NN structures using only one single NN, there are D elements in the input layer and D elements in the output layer, and the number of neurons in the hidden layer is set in the range of [1, 15]. They are tested on five problem instances, where the shift length is set as 2 and the number of peaks is set as 10, 30, 50, 100, and 200. The training error E_T, prediction error E_P, and training time are reported in Table III. It can be seen that all the NN structures with only one single NN perform poorly,

TABLE III

PERFORMANCE OF NNIT WITH DIFFERENT NN STRUCTURES IN TERMS OF TRAINING ERROR E_T, PREDICTION ERROR E_P, AND TRAINING TIME

P	One single 3-layer NN: S_{hidden}(total number of parameters n)															D 3-layer NNs			
	1(16)			2(27)			3(38)			4(49)			5(60)						
	E_T	E_P	Time(s)	E_T	E_P	Time(s)	E_T	E_P	Time(s)	E_T	E_P	Time(s)	E_T	E_P	Time(s)	S_{hidden} (n)	E_T	E_P	Time (s)
10	72.03	71.78	0.36	71.49	71.27	0.58	75.23	74.96	0.93	75.83	75.56	1.36	76.43	75.93	1.94	1(8)	**0.99**	**2.99**	1.99
30	68.21	68.19	0.86	69.24	69.22	1.52	70.24	70.18	2.39	72.24	72.01	3.65	74.08	73.77	5.26	2(15)	**1.52**	**1.29**	4.12
50	71.37	71.42	1.37	72.07	72.12	2.4	74.56	74.49	3.88	75.94	75.78	5.97	77.2	77.05	8.36	4(29)	**1.86**	**1.62**	6.20
150	71.27	71.32	3.95	72.25	72.33	6.93	74.71	74.7	11.34	76.07	75.93	17.5	77.32	77.08	24.67	12(85)	**2.48**	**1.99**	16.91
200	70.08	70.15	5.29	71.22	71.27	9.19	72.92	72.87	14.93	74.89	74.77	23.45	76.9	76.72	32.42	16(112)	**2.72**	**2.04**	22.36
P	6(71)			7(82)			8(93)			9(104)			10(115)						
	E_T	E_P	Time(s)	E_T	E_P	Time(s)	E_T	E_P	Time(s)	E_T	E_P	Time(s)	E_T	E_P	Time(s)				
10	77.99	77.2	2.84	80.5	79.47	3.66	82.44	81.12	4.8	83.5	81.42	6.05	86.89	83.82	7.65				
30	77.21	76.54	7.27	77.75	76.92	9.57	80.62	79.18	11.87	82.83	80.64	14.62	84.25	81.13	18.14				
50	79.91	79.15	11.72	80.74	79.69	14.84	83.93	82.44	19.2	85.61	83.55	23.22	86.78	83.66	29.09				
150	79.87	79.23	33.46	81.18	80.28	42.66	82.84	81.5	54.88	84.05	81.75	66.83	86.36	83.16	82.22				
200	78.83	78.18	45.01	80.49	79.35	57.24	82.51	81.09	71.93	83.54	81.26	88.82	86.58	83.58	109.68				
P	11(126)			12(137)			13(148)			14(159)			15(170)						
	E_T	E_P	Time(s)	E_T	E_P	Time(s)	E_T	E_P	Time(s)	E_T	E_P	Time(s)	E_T	E_P	Time(s)				
10	85.76	82.51	8.89	88.58	84.84	10.88	89.55	85.34	12.95	92.87	87.18	19.58	94.78	88.66	22.97				
30	83.5	81	21.55	90.26	85.64	26.33	88.56	83.99	30.52	93.48	87.34	45.98	91.67	85.43	52.74				
50	88.28	85.37	33.92	92.3	88.32	41.2	92.63	87.79	48.66	93.15	87.97	71.09	95.03	88.88	64.58				
150	88.2	84.83	98.16	90.68	86.92	117.91	91.67	86.93	136.81	94.04	88.42	205.19	94.23	88.23	174.87				
200	87.07	84.02	129.01	90.67	86.74	154.6	91.66	86.5	231.57	95.2	88.78	269.33	94.07	87.85	251.85				

where n is the numbers of parameters in each neural network. For $S_{hidden}(n)$, S_{hidden} indicates the number of the neurons in the hidden layer of a single NN and n represents the number of total parameters in the single NN. For example, for the NN denoted 1(16), there are 1 neuron in the hidden layer and totally 16 parameters (including weights and biases) in the NN.

getting significantly worse training errors and prediction errors than the D three-layer NNs. This may be due to that training is more difficult when there are D elements in the output layer. In contrast, the D three-layer NNs have only one element in the output layer, leading to fewer parameters in each NN and easier training when using the same number of neurons in the hidden layer. The adopted D three-layer NNs can well fit training sets with different sizes. The training time of the adopted structures is acceptable and even can be further reduced by training all the D NNs in parallel. In general, the adopted NN structure with D three-layer NNs obtains the smallest training errors and prediction errors in acceptable time on all tested instances. Hence, the adopted NN structure can be believed to be a good choice.

V. EXPERIMENTAL RESULTS AND COMPARISON STUDIES

In this section, we present the experimental results of the NNIT assisted EAs. They are compared with the original EAs to observe the performance improvement brought by NNIT. Five typical state-of-the-art algorithms based on particle swarm optimization (PSO) [54] and differential evolution [55] are selected for comprehensive study, including clustering PSO (CPSO) [23], CPSO without change detection (CPSOR) [25], adaptive multipopulation framework with PSO (AMP/PSO) [37], cluster-based dynamic DE with external archive (CDDE_Ar) [34], and dynamic DE with Brownian and quantum individuals (DDEBQ) [56]. The experiments are done on the well-known MPB. Multiple MPB instances are constructed for test.

A. Algorithms for Study

The five algorithms adopt different techniques to deal with the diversity loss in new environments. Among them, both of CPSO [23] and CDDE_Ar [34] restart when the environment changes and divide the population into multiple subpopulations by clustering methods to track the moving optima. On the contrary, CPSOR [25], AMP/PSO [37], and DDEBQ [56] maintain population diversity along the whole evolutionary process and do nothing when the environment changes. Particularly, CPSOR keeps detecting population diversity and adds new individuals when diversity decreases below a predefined threshold, and AMP/PSO adaptively updates the number of subpopulations, while DDEBQ adopts aging mechanism and quantum and Brownian individuals.

These EAs integrate with the NNIT to form CPSO-NNIT, CPSOR-NNIT, AMP/PSO-NNIT, CDDE_Ar-NNIT, and DDEBQ-NNIT. In the NNIT variants, the EA searches the optima in each environment, while NNIT transfers past information to assist the EA when the environment changes. Once the environment changes, the initial population in CPSO and CDDE_Ar is generated and the existing populations in CPSOR, AMP/PSO, and DDEBQ are reevaluated. Then, the NNIT procedure is performed to generate new solutions and adds them to the initial population. In the NNIT procedure, the solutions found in the previous environment are collected as the set S, while the initial population of the EA in the new environment is used as the set I. After that, the population is evolved by EA to find the new optima.

B. Experimental Configuration

1) Parameter Settings: The parameters of each algorithm are set following their original papers. The population size for CPSO, AMP/PSO, CDDE_Ar, and DDEBQ is set as 100, 100, 80, and 60, respectively. Especially, in CPSOR, the population size is set as $300 \times (1\text{-}e^{-0.33\times P^{0.5}})$ according to the number of peaks. For the PSO parameters, the inertia weight is set as 0.6 and 0.7298 in CPSOR [25] and AMP/PSO [37], respectively, while it linearly decreases from 0.6 to 0.3 in CPSO [23]; the acceleration coefficients c_1 and c_2 are set as 1.7 in CPSO and CPSOR, while they are set as 1.496 for AMP/PSO. The DE operators (i.e., mutation and crossover) are used in both of CDDE_Ar and DDEBQ. The crossover rate is set as 0.9 for both. The scale factor is set as 0.5 in CDDE_Ar [34], while it is adaptively updated in DDEBQ [56].

TABLE IV
RESULTS OF DIFFERENT ALGORITHMS ON MPB INSTANCES WITH A DIFFERENT NUMBER OF PEAKS

s_X	P	Error	CPSO	CPSO-NNIT		CPSOR	CPSOR-NNIT		AMP/PSO	AMP/PSO-NNIT		CDDE_Ar	CDDE_Ar-NNIT		DDEBQ	DDEBQ-NNIT	
5	10	E_o	10.75(1.22)	**10.06(1.30)**	=	4.42(0.18)	**4.06(0.12)**	+	2.83(0.09)	**2.76(0.10)**	+	5.14(0.18)	**4.97(0.20)**	+	8.33(0.51)	**8.12(0.55)**	=
		E_B	3.04(0.81)	**2.56(0.73)**	+	0.57(0.06)	**0.53(0.10)**	=	**0.24(0.10)**	0.27(0.10)	=	2.93(0.20)	**2.87(0.24)**	=	4.71(0.47)	**4.66(0.51)**	=
	20	E_o	9.45(1.04)	**8.12(0.86)**	+	4.08(0.25)	**3.75(0.24)**	+	4.28(0.16)	**4.19(0.13)**	=	6.20(0.31)	**6.17(0.32)**	=	9.92(0.38)	**9.86(0.57)**	=
		E_B	2.94(0.52)	**2.61(0.67)**	=	**1.25(0.21)**	1.31(0.24)	=	**1.03(0.15)**	1.12(0.14)	=	**3.50(0.33)**	3.58(0.32)	=	**5.57(0.47)**	5.78(0.53)	=
	30	E_o	8.17(0.88)	**7.53(0.88)**	+	4.23(0.43)	**3.82(0.32)**	+	3.68(0.09)	**3.51(0.11)**	+	4.89(0.21)	**4.82(0.18)**	=	7.32(0.36)	**7.04(0.45)**	=
		E_B	2.77(0.56)	**2.47(0.44)**	=	1.58(0.27)	**1.45(0.19)**	=	**0.85(0.09)**	0.91(0.11)	=	**2.99(0.23)**	3.02(0.18)	=	4.05(0.31)	**3.97(0.38)**	=
	40	E_o	7.60(0.40)	**6.92(0.60)**	+	4.48(0.25)	**3.99(0.17)**	+	4.35(0.14)	**4.20(0.13)**	+	5.71(0.25)	**5.62(0.29)**	=	8.44(0.47)	**8.34(0.44)**	=
		E_B	2.53(0.34)	**2.36(0.33)**	=	1.59(0.19)	**1.45(0.12)**	+	**1.58(0.20)**	1.63(0.19)	=	3.87(0.28)	**3.86(0.34)**	=	**5.21(0.57)**	5.28(0.42)	=
	50	E_o	6.97(0.61)	**6.50(0.51)**	+	4.70(0.18)	**4.25(0.23)**	+	4.75(0.13)	**4.52(0.12)**	+	5.71(0.31)	**5.47(0.30)**	+	9.19(0.66)	**8.86(0.46)**	+
		E_B	2.28(0.40)	**2.16(0.32)**	=	2.00(0.14)	**1.83(0.20)**	+	**1.61(0.10)**	1.65(0.13)	=	3.69(0.32)	**3.51(0.31)**	=	5.71(0.66)	**5.58(0.43)**	=
	80	E_o	6.33(0.43)	**5.80(0.30)**	+	4.92(0.24)	**4.54(0.23)**	+	4.46(0.13)	**4.21(0.16)**	+	5.26(0.19)	**5.18(0.19)**	=	7.91(0.29)	**7.69(0.35)**	+
		E_B	2.05(0.26)	**2.01(0.22)**	=	2.08(0.19)	**1.98(0.17)**	=	**1.51(0.12)**	1.53(0.09)	=	3.45(0.20)	**3.42(0.22)**	=	4.92(0.26)	**4.85(0.37)**	=
	100	E_o	5.93(0.35)	**5.57(0.30)**	+	5.00(0.32)	**4.43(0.29)**	+	4.10(0.13)	**3.72(0.11)**	+	4.61(0.16)	**4.48(0.13)**	+	7.46(0.42)	**7.21(0.46)**	=
		E_B	1.96(0.25)	**1.92(0.22)**	=	2.29(0.27)	**2.07(0.26)**	+	1.34(0.10)	**1.30(0.08)**	+	2.80(0.20)	**2.76(0.13)**	=	4.45(0.42)	**4.31(0.41)**	=
	120	E_o	5.74(0.33)	**5.20(0.40)**	+	4.76(0.33)	**4.24(0.25)**	+	4.07(0.14)	**3.91(0.15)**	+	4.48(0.19)	**4.41(0.20)**	=	7.56(0.31)	**7.29(0.36)**	+
		E_B	1.83(0.24)	**1.71(0.22)**	=	2.03(0.21)	**1.88(0.18)**	+	**1.35(0.08)**	1.38(0.18)	=	**2.71(0.20)**	2.73(0.22)	=	4.35(0.27)	**4.27(0.34)**	=
	150	E_o	5.35(0.28)	**5.08(0.33)**	+	5.42(0.42)	**4.61(0.29)**	+	4.02(0.13)	**3.85(0.11)**	+	4.24(0.17)	**4.21(0.16)**	=	6.99(0.37)	**6.60(0.26)**	+
		E_B	1.71(0.22)	**1.64(0.18)**	=	2.62(0.34)	**2.31(0.27)**	+	**1.41(0.09)**	1.44(0.08)	=	**2.52(0.15)**	2.55(0.16)	=	3.93(0.35)	**3.77(0.23)**	=
	200	E_o	5.03(0.27)	**4.88(0.23)**	=	4.41(0.27)	**3.79(0.18)**	+	3.81(0.09)	**3.56(0.12)**	+	3.78(0.20)	**3.62(0.14)**	+	6.46(0.35)	**6.19(0.28)**	+
		E_B	**1.52(0.17)**	1.61(0.17)	=	2.01(0.16)	**1.78(0.14)**	+	1.27(0.09)	**1.22(0.08)**	=	2.15(0.18)	**2.03(0.15)**	+	3.52(0.26)	**3.39(0.23)**	=
NNIT is better (+)			9			16			10			5			5		
=			11			4			10			15			15		
NNIT is worse (−)			0			0			0			0			0		

where "+, =, −" indicate that the NNIT variant is significantly better, equal to, or worse than the original algorithm according to Wilcoxon's signed rank test at a 0.05 significance level.

Noted that all these EAs are based on multiple populations, therefore, the cluster number M is the same as the number of local optima collected by the EAs and the population size is controlled by their corresponding mechanisms.

2) Quality Indicators: To evaluate the performance of the algorithms, we adopt two widely used quality indicators, i.e., offline error E_o and best-before-change error E_B. Both of the two indicators are calculated based on the assumption that the real optima of the problem in different environments are known, that is, the error is calculated as the difference between the found solutions and the real optima. Especially, the offline error E_o is calculated based on the data during the whole evolutionary process, while the best-before-change error E_B is calculated based on the final best solution of each environment. Therefore, E_o is in fact an online indicator to reflect the convergence speed of the algorithm to some extent, while E_B represents the global search ability of the algorithm.

For E_o, we use the best solution in every two FEs along the whole evolutionary process to calculate the average value as

$$E_o = \frac{1}{C \times num\_sample} \sum_{i=1}^{C} \sum_{j=1}^{num\_sample} E_{ij} \tag{22}$$

where C is the number of environments; *num_sample* is the number of samples in each environment and it is sampled every two FEs following [37]; and E_{ij} is the error of the best solution found at the jth sampling in the ith environment.

For E_B, we calculate the average error of the best solutions found in all environments as

$$E_B = \frac{1}{C} \sum_{i=1}^{C} E_i^{\text{Best}} \tag{23}$$

where E_i^{Best} is the error of the best solution found at the end of the ith environment. These two indicators give a comprehensive evaluation on both of convergence speed and solution quality.

In the experiments, all algorithms and their corresponding NNIT variants independently run 20 times and the average results are reported. The better algorithm is marked in bold. In addition, Wilcoxon's signed rank test is performed between the algorithms and their NNIT variants at a 0.05 significance level.

C. Experimental Results

To test the effect of the NNIT on a different number of peaks and a different severity of environment changes, we perform the comparison experiments on ten instances with a different number of peaks and 18 instances with different shift lengths.

1) Results on Instances With a Different Number of Peaks: The number of peaks is set as 10, 20, 30, 40, 50, 80, 100, 120, 150, and 200, and the shift length is fixed at 5. The results on these instances are reported in Table IV.

From Table IV, we can see that all the NNIT variants can get smaller E_o and E_B values on most instances, showing that NNIT can help EAs accelerate convergence and improve solution quality on instances with a different number of peaks. In detail, in terms of the E_o indicator, according to the significance test, the NNIT variants perform significantly better than CPSO, CPSOR, AMP/PSO, CDDE_Ar, and DDEBQ on 8, 10, 9, 4, and 5 out of 10 instances, respectively. This shows that the NNIT can accelerate algorithm convergence on most instances for the selected algorithms. As for the E_B indicator of solution quality, there is no significant difference on most instances for all the algorithms. This is because the NNIT only reuses the past solutions for convergence acceleration but not exploring new areas. On the other hand, the NNIT only provides the solutions for guidance, and the final solution quality depends on the search ability of the EAs. Hence,

TABLE V
RESULTS OF DIFFERENT ALGORITHMS ON MPB WITH DIFFERENT VALUES OF SHIFT LENGTH

P	s_X	Error	CPSO	CPSO-NNIT		CPSOR	CPSOR-NNIT		AMP/PSO	AMP/PSO-NNIT		CDDE_Ar	CDDE_Ar-NNIT		DDEBQ	DDEBQ-NNIT	
P=10	1	E_o	6.16(0.87)	**5.54(0.93)**	+	1.47(0.07)	**1.47(0.07)**	=	1.18(0.09)	**1.11(0.05)**	+	2.62(0.19)	**2.50(0.26)**	=	**5.71(0.89)**	5.74(0.71)	=
		E_B	2.28(0.51)	**1.82(0.73)**	+	**0.14(0.07)**	0.17(0.07)	=	0.05(0.03)	**0.05(0.05)**	=	1.75(0.21)	**1.64(0.26)**	=	**4.34(0.83)**	4.41(0.72)	=
	2	E_o	7.45(0.89)	**6.74(1.01)**	+	2.43(0.09)	**2.34(0.08)**	+	1.71(0.10)	**1.69(0.07)**	=	3.34(0.20)	**3.28(0.19)**	=	6.67(0.90)	**6.22(0.89)**	=
		E_B	2.36(0.58)	**1.89(0.58)**	+	0.30(0.07)	**0.30(0.06)**	=	**0.08(0.02)**	0.09(0.04)	=	**1.93(0.22)**	1.95(0.21)	=	4.70(0.84)	**4.30(0.89)**	=
	3	E_o	8.89(1.45)	**7.99(0.79)**	+	3.22(0.13)	**3.03(0.14)**	+	2.24(0.16)	**2.03(0.09)**	+	4.14(0.21)	**4.03(0.24)**	=	7.15(0.49)	**6.84(0.65)**	+
		E_B	2.48(0.78)	**2.13(0.51)**	=	0.42(0.12)	**0.39(0.08)**	=	0.17(0.06)	**0.16(0.07)**	=	**2.40(0.27)**	2.42(0.29)	=	4.56(0.43)	**4.34(0.65)**	=
	4	E_o	10.00(1.59)	**9.10(1.31)**	=	3.81(0.15)	**3.61(0.14)**	+	2.50(0.11)	**2.43(0.10)**	+	4.68(0.24)	**4.57(0.20)**	=	7.60(0.45)	**7.56(0.59)**	=
		E_B	2.74(1.04)	**2.41(0.82)**	=	0.49(0.09)	**0.46(0.08)**	=	0.20(0.06)	**0.17(0.05)**	+	2.71(0.31)	**2.65(0.20)**	=	**4.51(0.47)**	4.63(0.59)	=
	5	E_o	10.75(1.22)	**10.06(1.30)**	=	4.42(0.18)	**4.06(0.12)**	+	2.83(0.09)	**2.76(0.10)**	+	5.14(0.18)	**4.97(0.20)**	+	8.33(0.51)	**8.12(0.55)**	=
		E_B	3.04(0.81)	**2.56(0.73)**	+	0.57(0.06)	**0.53(0.10)**	=	**0.24(0.10)**	0.27(0.10)	=	2.93(0.20)	**2.87(0.24)**	=	4.71(0.47)	**4.66(0.51)**	=
	6	E_o	11.23(1.49)	**10.13(1.12)**	+	4.94(0.20)	**4.58(0.14)**	+	**3.19(0.16)**	3.21(0.18)	=	5.55(0.20)	**5.34(0.22)**	+	9.17(0.41)	**8.64(0.52)**	+
		E_B	2.93(0.51)	**2.36(0.58)**	+	0.75(0.09)	**0.72(0.13)**	=	**0.47(0.10)**	0.56(0.19)	=	3.07(0.25)	**3.00(0.28)**	=	5.10(0.40)	**4.74(0.50)**	+
P=80	1	E_o	3.83(0.25)	**3.76(0.37)**	=	3.62(0.27)	**3.28(0.32)**	+	2.72(0.07)	**2.52(0.09)**	+	3.67(0.20)	**3.49(0.22)**	+	5.08(0.26)	**5.06(0.37)**	=
		E_B	**1.74(0.19)**	1.85(0.33)	=	2.40(0.28)	**2.20(0.30)**	+	1.03(0.07)	**1.01(0.08)**	=	2.86(0.21)	**2.70(0.23)**	=	**3.90(0.27)**	3.93(0.38)	=
	2	E_o	4.89(0.44)	**4.51(0.34)**	+	4.31(0.29)	**3.95(0.35)**	+	3.48(0.12)	**3.20(0.13)**	+	4.08(0.19)	**4.05(0.13)**	=	6.09(0.33)	**5.88(0.35)**	=
		E_B	2.02(0.31)	**1.85(0.24)**	=	2.28(0.22)	**2.26(0.29)**	=	1.39(0.09)	**1.30(0.10)**	+	**2.95(0.20)**	2.96(0.16)	=	4.37(0.32)	**4.20(0.39)**	=
	3	E_o	5.47(0.46)	**5.06(0.49)**	+	4.62(0.31)	**4.15(0.37)**	+	3.97(0.10)	**3.65(0.14)**	+	4.53(0.19)	**4.47(0.16)**	=	6.72(0.35)	**6.41(0.28)**	+
		E_B	2.03(0.32)	**1.86(0.25)**	+	2.22(0.25)	**2.08(0.31)**	=	1.49(0.06)	**1.43(0.13)**	=	3.15(0.21)	**3.14(0.18)**	=	4.49(0.37)	**4.32(0.23)**	=
	4	E_o	5.81(0.53)	**5.45(0.38)**	+	4.93(0.29)	**4.46(0.31)**	+	4.27(0.11)	**3.98(0.11)**	+	4.96(0.17)	**4.80(0.18)**	+	7.43(0.21)	**7.14(0.23)**	+
		E_B	1.99(0.34)	**1.92(0.25)**	=	2.22(0.28)	**2.10(0.26)**	=	1.58(0.07)	**1.52(0.10)**	+	3.31(0.20)	**3.23(0.17)**	=	4.84(0.22)	**4.66(0.23)**	+
	5	E_o	6.33(0.43)	**5.80(0.30)**	+	4.92(0.24)	**4.54(0.23)**	+	4.46(0.13)	**4.21(0.16)**	+	5.26(0.19)	**5.18(0.19)**	=	7.91(0.29)	**7.69(0.35)**	+
		E_B	2.05(0.26)	**2.01(0.22)**	=	2.08(0.19)	**1.98(0.17)**	=	**1.51(0.12)**	1.53(0.09)	=	3.45(0.20)	**3.42(0.22)**	=	4.92(0.26)	**4.85(0.37)**	=
	6	E_o	6.43(0.40)	**6.07(0.36)**	+	5.28(0.28)	**4.73(0.18)**	+	4.64(0.13)	**4.43(0.13)**	+	5.61(0.24)	**5.45(0.20)**	+	8.38(0.40)	**8.27(0.36)**	=
		E_B	**1.98(0.19)**	2.03(0.19)	=	2.14(0.21)	**1.95(0.18)**	+	1.52(0.09)	**1.52(0.08)**	=	3.65(0.26)	**3.57(0.22)**	=	**4.94(0.35)**	5.07(0.36)	=
P=200	1	E_o	3.24(0.13)	**3.07(0.15)**	+	3.27(0.22)	**2.93(0.34)**	+	2.49(0.15)	**2.35(0.08)**	+	2.37(0.18)	**2.31(0.14)**	=	**4.28(0.45)**	4.32(0.40)	=
		E_B	1.47(0.10)	**1.47(0.12)**	=	2.16(0.23)	**2.04(0.32)**	+	1.17(0.09)	**1.10(0.08)**	+	1.67(0.17)	**1.63(0.15)**	=	**3.12(0.39)**	3.25(0.33)	=
	2	E_o	3.91(0.20)	**3.73(0.24)**	+	3.96(0.37)	**3.21(0.31)**	+	3.06(0.08)	**2.69(0.09)**	+	2.81(0.16)	**2.79(0.15)**	=	4.98(0.42)	**4.92(0.37)**	=
		E_B	1.54(0.14)	**1.49(0.16)**	=	2.27(0.34)	**1.91(0.25)**	+	1.29(0.06)	**1.16(0.10)**	+	**1.77(0.18)**	1.78(0.17)	=	**3.29(0.39)**	3.33(0.31)	=
	3	E_o	4.40(0.17)	**4.17(0.23)**	+	4.09(0.34)	**3.35(0.22)**	+	3.28(0.14)	**3.01(0.15)**	+	3.16(0.15)	**3.12(0.15)**	=	5.47(0.36)	**5.16(0.33)**	+
		E_B	**1.52(0.12)**	1.56(0.14)	=	2.10(0.27)	**1.77(0.19)**	+	1.25(0.07)	**1.23(0.14)**	=	**1.89(0.15)**	1.92(0.16)	=	3.38(0.31)	**3.20(0.27)**	+
	4	E_o	4.73(0.23)	**4.39(0.26)**	+	4.32(0.31)	**3.60(0.26)**	+	3.60(0.12)	**3.36(0.10)**	+	3.44(0.15)	**3.41(0.14)**	=	5.89(0.33)	**5.81(0.38)**	=
		E_B	1.53(0.15)	**1.50(0.17)**	=	2.08(0.20)	**1.81(0.19)**	+	1.28(0.06)	**1.26(0.09)**	=	**1.95(0.15)**	2.01(0.14)	=	3.45(0.29)	**3.40(0.36)**	=
	5	E_o	5.03(0.27)	**4.88(0.23)**	=	4.41(0.27)	**3.79(0.18)**	+	3.81(0.09)	**3.56(0.12)**	+	3.78(0.20)	**3.62(0.14)**	+	6.46(0.35)	**6.19(0.28)**	+
		E_B	**1.52(0.17)**	1.61(0.17)	=	2.01(0.16)	**1.78(0.14)**	+	1.27(0.09)	**1.22(0.08)**	=	2.15(0.18)	**2.03(0.15)**	+	3.52(0.26)	**3.39(0.23)**	=
	6	E_o	5.40(0.29)	**4.96(0.25)**	+	4.56(0.32)	**3.91(0.25)**	+	4.00(0.10)	**3.81(0.19)**	+	3.92(0.18)	**3.91(0.17)**	=	6.87(0.30)	**6.53(0.30)**	+
		E_B	1.65(0.21)	**1.54(0.15)**	=	1.89(0.20)	**1.66(0.19)**	+	**1.31(0.10)**	1.34(0.14)	=	**2.11(0.18)**	2.15(0.18)	=	3.53(0.25)	**3.45(0.24)**	=
NNIT is better (+)			19			25			21			7			11		
=			17			11			15			29			25		
NNIT is worse (−)			0			0			0			0			0		

where "+, =, −" indicate that the NNIT variant is significantly better, equal to, or worse than the original algorithm according to Wilcoxon's signed rank test at a 0.05 significance level.

the NNIT variants may obtain a similar solution quality as the original algorithms. Nevertheless, the NNIT variants can still obtain significantly better E_B values than CPSO, CPSOR, AMP/PSO, and CDDE_Ar on 1, 6, 1, and 1 instances, while none instances are worse. Thus, the NNIT helps EAs find good solutions faster on most instances and allows it to use more FEs for exploring new areas to get better solutions.

In general, the NNIT can reuse past solutions to speed up the convergence of EAs in new environments.

2) Results on Instances With Different Shift Lengths: To test the response ability of the proposed NNIT to different change severity, the shift length is tested in the range of [1, 6] with a step size of 1. The number of peaks is set as three values 10, 80, and 200, and thereby, totally, 18 instances are constructed for testing. The results are reported in Table V.

From Table V, we can see that all the NNIT variants can obtain smaller E_o and E_B values than the corresponding original algorithms on most of instances, showing that the NNIT can strongly response to different change severity and can accelerate algorithm convergence even improve solution quality. For example, on the instances with ten peaks, in terms of E_o metric, the NNIT variants for CPSO, CPSOR, AMP/PSO, CDDE_Ar, and DDEBQ can obtain significantly better values than the original algorithms on 4, 5, 4, 2, and 2 out of 6 instances, respectively, while getting equal performance on all the other instances; as for the E_B metric, the NNIT variants of CPSO perform significantly better on four out of six instances, and the NNIT also significantly improves the solution quality on one instance for both of AMP/PSO and DDEBQ. Similar situations can be seen on the instances with 80 and 200 peaks. Therefore, the NNIT can perceive environment changes under different change severity to accelerate convergence and enhance the global search ability of EAs.

D. Discussion on Computational Effort

The above-mentioned experiments show that the NNIT variants can obtain generally better results than the original optimization algorithms. In this section, we discuss the computational effort involving in solving problems based on the number of FEs since FE in DOPs usually consumes most of the time in the optimization process. For an optimization algorithm and its NNIT variant, they only differ on that the NNIT variant has an additional NNIT procedure for information transfer. The NNIT procedure only uses the solutions collected from optimization algorithms for training, including the solutions found in previous environments and the initial population in new environments, and generates new solutions that will be used by optimization algorithms. The solutions for training have been evaluated by optimization algorithms, and the newly generated solutions would be evaluated when they are used by optimization algorithms, that is, the NNIT procedure does not consume any more FEs and only the optimization part consumes FEs. Hence, all algorithms and their NNIT variants

consume the same number of FEs in the compared experiments. Using the same FEs, the NNIT variants can obtain generally better results compared with original algorithms. Additionally, the running time of the external NNIT procedure only depends on the size of the training set (the population size of the optimization algorithm) and is acceptable according to the discussion in Section IV-C. Thus, the NNIT variants perform better than original optimization algorithms when using similar computational resources with the same number of FEs.

VI. Conclusion

To track the information reuse issue in DOPs, this paper proposes an NN-based information transfer method, named NNIT, to reuse past solutions for the optimization of new environments. Solutions are collected from a new environment and its last environment to construct data. Then, the NN perceives the environment change from the data to learn a transfer model and further use the transfer model to reuse past solutions for generating new solutions to assist EAs in new environments. Several state-of-the-art EAs for DOPs are selected to incorporate the proposed NNIT, and the resultant algorithms are tested on the well-known MPB. The experimental results show that the EAs based on the NNIT can converge faster than the original algorithms. The NNIT method is promising. The proposed NNIT method provides a new way to directly learn the transfer function of environment change and transfer information in DOPs. In the future work, we will investigate the performance of reinforcement learning on the involved DOPs and apply the NNIT-assisted EAs to solve DOPs in practical applications.

References

[1] O. K. Oyedotun and A. Khashman, "Prototype-incorporated emotional neural network," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 8, pp. 3560–3572, Aug. 2018.

[2] I. Salgado and I. Chairez, "Adaptive unknown input estimation by sliding modes and differential neural network observer," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 8, pp. 3499–3509, Aug. 2018.

[3] H. Rezaee and F. Abdollahi, "Adaptive consensus control of nonlinear multiagent systems with unknown control directions under stochastic topologies," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 8, pp. 3538–3547, Aug. 2018.

[4] F. Song, Y. Liu, J.-X. Xu, X. Yang, and Q. Zhu, "Data-driven iterative feedforward tuning for a wafer stage: A high-order approach based on instrumental variables," *IEEE Trans. Ind. Electron.*, vol. 66, no. 4, pp. 3106–3116, Apr. 2019.

[5] X.-F. Liu, Z.-H. Zhan, J. D. Deng, Y. Li, T. L. Gu, and J. Zhang, "An energy efficient ant colony system for virtual machine placement in cloud computing," *IEEE Trans. Evol. Comput.*, vol. 22, no. 1, pp. 113–128, Feb. 2018.

[6] Z.-J. Wang *et al.*, "Dual-strategy differential evolution with affinity propagation clustering for multimodal optimization problems," *IEEE Trans. Evol. Comput.*, vol. 22, no. 6, pp. 894–908, Dec. 2018.

[7] Z.-H. Zhan *et al.*, "Cloudde: A heterogeneous differential evolution algorithm and its distributed cloud version," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 704–716, 2018.

[8] Z.-G. Chen *et al.*, "Multiobjective cloud workflow scheduling: A multiple populations ant colony system approach," *IEEE Trans. Cybern.*, vol. 49, no. 8, pp. 2912–2926, Aug. 2019.

[9] C. Cruz, J. R. González, and D. A. Pelta, "Optimization in dynamic environments: A survey on problems, methods and measures," *Soft Comput.*, vol. 15, no. 7, pp. 1427–1448, 2011.

[10] S. Q. Qian, Y. Ye, B. Jiang, and G. Xu, "A Micro-cloning dynamic multiobjective algorithm with an adaptive change reaction strategy," *Soft Comput.*, vol. 21, no. 13, pp. 3781–3801, 2017.

[11] S. Yang and X. Yao, "Population-based incremental learning with associative memory for dynamic environments," *IEEE Trans. Evol. Comput.*, vol. 12, no. 5, pp. 542–561, Oct. 2008.

[12] P. D. Stroud, "Kalman-extended genetic algorithm for search in nonstationary environments with noisy fitness evaluations," *IEEE Trans. Evol. Comput.*, vol. 5, no. 1, pp. 66–77, Feb. 2001.

[13] X.-F. Liu, Z.-H. Zhan, and J. Zhang, "Neural network for change direction prediction in dynamic optimization," *IEEE Access*, vol. 6, pp. 72649–72662, 2018.

[14] M. Jiang, Z. Huang, L. Qiu, W. Huang, and G. G. Yen, "Transfer learning-based dynamic multiobjective optimization algorithms," *IEEE Trans. Evol. Comput.*, vol. 22, no. 4, pp. 501–514, Aug. 2018.

[15] C. Rossi, M. Abderrahim, and J. C. Díaz, "Tracking moving optima using Kalman-based predictions," *Evol. Comput.*, vol. 16, no. 1, pp. 1–30, 2008.

[16] A. Simões and E. Costa, "Evolutionary algorithms for dynamic environments: Prediction using linear regression and Markov chains," in *Proc. Parallel Problem Solving Nature (PPSN X)*. Berlin, Germany: Springer, 2008, pp. 306–315.

[17] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.

[18] M. G. Carneiro and L. Zhao, "Organizational data classification based on theimportanceconcept of complex networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 8, pp. 3361–3373, Aug. 2018.

[19] S. H. Khan, M. Hayat, M. Bennamoun, F. A. Sohel, and R. Togneri, "Cost-sensitive learning of deep feature representations from imbalanced data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 8, pp. 3573–3587, Aug. 2018.

[20] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–15.

[21] J. Yu, C. Zhu, J. Zhang, Q. Huang, and D. Tao, "Spatial pyramid-enhanced NetVLAD with weighted triplet loss for place recognition," *IEEE Trans. Neural Netw. Learn. Syst.*, to be published. doi: 10.1109/TNNLS.2019.2908982.

[22] S. Yang, "Genetic algorithms with memory- and elitism-based immigrants in dynamic environments," *Evol. Comput.*, vol. 16, no. 3, pp. 385–416, 2008.

[23] S. Yang and C. Li, "A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments," *IEEE Trans. Evol. Comput.*, vol. 14, no. 6, pp. 959–974, Dec. 2010.

[24] Y. G. Woldesenbet and G. G. Yen, "Dynamic evolutionary algorithm with variable relocation," *IEEE Trans. Evol. Comput.*, vol. 13, no. 3, pp. 500–513, Jun. 2009.

[25] C. Li and S. Yang, "A general framework of multipopulation methods with clustering in undetectable dynamic environments," *IEEE Trans. Evol. Comput.*, vol. 16, no. 4, pp. 556–577, Aug. 2012.

[26] T. M. Blackwell and P. J. Bentley, "Dynamic search with charged swarms," in *Proc. 4th Annu. Conf. Genetic Evol. Comput.*, 2002, pp. 19–26.

[27] L. Liu, S. Yang, and D. Wang, "Particle swarm optimization with composite particles in dynamic environments," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 40, no. 6, pp. 1634–1648, Dec. 2010.

[28] X. Li and K. H. Dam, "Comparing particle swarms for tracking extrema in dynamic environments," in *Proc. Congr. Evol. Comput.*, 2003, pp. 1772–1779.

[29] M. Daneshyari and G. G. Yen, "Dynamic optimization using cultural based PSO," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2011, pp. 509–516.

[30] H. Wang, N. Wang, and D. Wang, "Multi-swarm optimization algorithm for dynamic optimization problems using forking," in *Proc. Chin. Control Decis. Conf.*, 2008, pp. 2415–2419.

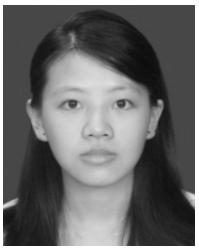
[31] R. I. Lung and D. Dumitrescu, "Evolutionary swarm cooperative optimization in dynamic environments," *Natural Comput.*, vol. 9, no. 1, pp. 83–94, 2010.

[32] A. M. Turky and S. Abdullah, "A multi-population harmony search algorithm with external archive for dynamic optimization problems," *Inf. Sci.*, vol. 272, pp. 84–95, Jul. 2014.

[33] I. Schoeman and A. Engelbrecht, "Niching for dynamic environments using particle swarm optimization," in *Simulated Evolution and Learning*. Berlin, Germany: Springer, 2006, pp. 134–141.

[34] U. Halder, S. Das, and D. Maity, "A cluster-based differential evolution algorithm with external archive for optimization in dynamic environments," *IEEE Trans. Cybern.*, vol. 43, no. 3, pp. 881–897, Jun. 2013.

[35] Z. J. Wang *et al.*, "Automatic niching differential evolution with contour prediction approach for multimodal optimization problems," *IEEE Trans. Evol. Comput.*, to be published. doi: 10.1109/TEVC.2019.2910721.
[36] M. C. du Plessis and A. P. Engelbrecht, "Differential evolution for dynamic environments with unknown numbers of optima," *J. Global Optim.*, vol. 55, no. 1, pp. 73–99, 2013.
[37] C. Li, T. T. Nguyen, M. Yang, M. Mavrovouniotis, and S. Yang, "An adaptive multipopulation framework for locating and tracking multiple optima," *IEEE Trans. Evol. Comput.*, vol. 20, no. 4, pp. 590–605, Aug. 2016.
[38] D. Yazdani, B. Nasiri, A. Sepas-Moghaddam, and M. R. Meybodi, "A novel multi-swarm algorithm for optimization in dynamic environments based on particle swarm optimization," *Appl. Soft Comput.*, vol. 13, no. 4, pp. 2144–2158, 2013.
[39] D. Yazdani, B. Nasiri, A. Sepas-Moghaddam, M. Meybodi, and M. Akbarzadeh-Totonchi, "mNAFSA: A novel approach for optimization in dynamic environments with global changes," *Swarm Evol. Comput.*, vol. 18, pp. 38–53, Oct. 2014.
[40] J. Lewis, E. Hart, and G. Ritchie, "A comparison of dominance mechanisms and simple mutation on non-stationary problems," in *Proc. Parallel Problem Solving Nature (PPSN V)*. Berlin, Germany: Springer, 1998, pp. 139–148.
[41] W. Zhong, J. Xing, Y. Liang, and F. Qian, "Dynamic optimization with an improved θ-PSO based on memory recall," in *Proc. 8th World Congr. Intell. Control Automat.*, 2010, pp. 3225–3229.
[42] H. Nakano, M. Kojima, and A. Miyauchi, "An artificial bee colony algorithm with a memory scheme for dynamic optimization problems," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, May 2015, pp. 2657–2663.
[43] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in *Proc. Congr. Evol. Comput.*, 1999, pp. 1875–1882.
[44] T. Zhu, W. Luo, and L. Yue, "Combining multipopulation evolutionary algorithms with memory for dynamic optimization problems," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2014, pp. 2047–2054.
[45] X. Yu and X. Wu, "A multi-point local search algorithm for continuous dynamic optimization," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2016, pp. 2736–2743.
[46] Y. Cao and W. Luo, "Novel associative memory retrieving strategies for evolutionary algorithms in dynamic environments," in *Proc. Adv. Comput. Intell.* Berlin, Germany: Springer, 2009, pp. 258–268.
[47] M. Mavrovouniotis, F. Neri, and S. Yang, "An adaptive local search algorithm for real-valued dynamic optimization," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, May 2015, pp. 1388–1395.
[48] S. Yang, "Memory-based immigrants for genetic algorithms in dynamic environments," in *Proc. 7th Annu. Conf. Genetic Evol. Comput.*, Washington DC, USA, 2005, pp. 1115–1122.
[49] A. Zhou, Y. Jin, and Q. Zhang, "A population prediction strategy for evolutionary dynamic multiobjective optimization," *IEEE Trans. Cybern.*, vol. 44, no. 1, pp. 40–53, Jan. 2014.
[50] C. Li, H. Wu, Z. Yang, Y. Wang, and Z. Sun, "A novel SHLNN based robust control and tracking method for hypersonic vehicle under parameter uncertainty," *Complexity*, vol. 2017, Oct. 2017, Art. no. 6034786.
[51] E. da la Rosa and W. Yu, "Data-driven fuzzy modeling using restricted boltzmann machines and probability theory," *IEEE Trans. Syst., Man, Cybern., Syst.*, to be published. doi: 10.1109/TSMC.2018.2812156.
[52] H. B. Demuth, M. H. Beale, O. D. Jess, and M. T. Hagan, *Neural Network Design*, 2nd ed. Stillwater, OK, USA: Martin Hagan, 2014.
[53] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Trans. Neural Netw.*, vol. 5, no. 6, pp. 989–993, Nov. 1994.
[54] X. F. Liu, Z. H. Zhan, Y. Gao, J. Zhang, S. Kwong, and J. Zhang, "Coevolutionary particle swarm optimization with bottleneck objective learning strategy for many-objective optimization," *IEEE Trans. Evol. Comput.*, to be published. doi: 10.1109/TEVC.2018.2875430.
[55] X.-F. Liu *et al.*, "Historical and heuristic-based adaptive differential evolution," *IEEE Trans. Syst., Man, Cybern., Syst.*, to be published. doi: 10.1109/TSMC.2018.2855155.
[56] S. Das, A. Mandal, and R. Mukherjee, "An adaptive differential evolution algorithm for global optimization in dynamic environments," *IEEE Trans. Cybern.*, vol. 44, no. 6, pp. 966–978, Jun. 2014.



Xiao-Fang Liu (S'14) received the B.S. degree in computer science from Sun Yat-sen University, Guangzhou, China, in 2015, where she is currently pursuing the Ph.D. degree.

Her current research interests include artificial intelligence, evolutionary computation, swarm intelligence, and their applications in design and optimization, such as cloud computing resources' scheduling.

Zhi-Hui Zhan (M'13–SM'18) received the bachelor's and Ph.D. degrees from the Department of Computer Science, Sun Yat-sen University, Guangzhou, China, in 2007 and 2013, respectively.

From 2013 to 2015, he was a Lecturer and an Associate Professor with the Department of Computer Science, Sun Yat-sen University. Since 2016, he has been a Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, where he is also the Changjiang Scholar Young Professor and the Pearl River Scholar Young Professor. His current research interests include evolutionary computation algorithms, swarm intelligence algorithms, and their applications in real-world problems and in environments of cloud computing and big data.

Dr. Zhan's doctoral dissertation was awarded the China Computer Federation (CCF) Outstanding Dissertation and the IEEE Computational Intelligence Society (CIS) Outstanding Dissertation. He was a recipient of the Outstanding Youth Science Foundation from the National Natural Science Foundations of China (NSFC) in 2018 and the Wu Wen Jun Artificial Intelligence Excellent Youth from the Chinese Association for Artificial Intelligence in 2017. He is listed as one of the most cited Chinese researchers in computer science. He is currently an Associate Editor of *Neurocomputing*.

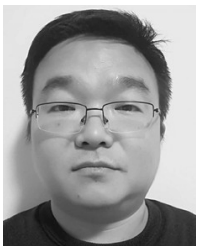
Tian-Long Gu received the M.Eng. degree from Xidian University, Xi'an, China, in 1987, and the Ph.D. degree from Zhejiang University, Hangzhou, China, in 1996.

From 1998 to 2002, he was a Research Fellow with the School of Electrical and Computer Engineering, Curtin University of Technology, Perth, WA, Australia, and a Post-Doctoral Fellow with the School of Engineering, Murdoch University, Perth. He is currently a Professor with the School of Computer Science and Engineering, Guilin University of Electronic Technology, Guilin, China. His current research interests include formal methods, data and knowledge engineering, software engineering, and information security protocol.

Sam Kwong (F'13) received the B.Sc. degree from The State University of New York at Buffalo, Buffalo, NY, USA, in 1983, the M.A.Sc. degree in electrical engineering from the University of Waterloo, Waterloo, ON, Canada, in 1985, and the Ph.D. degree from the University of Hagen, Hagen, Germany, in 1996.

From 1985 to 1987, he was a Diagnostic Engineer with Control Data Canada, Ottawa, ON, where he designed the diagnostic software to detect the manufacture faults of the VLSI chips in the Cyber 430 machine. He later joined Bell Northern Research Canada, Ottawa, as a Member of Scientific staff. In 1990, he joined the Department of Electronic Engineering, City University of Hong Kong, Hong Kong, as a Lecturer, where he is currently a Professor with the Department of Computer Science. His current research interests include pattern recognition, evolutionary computations, and video analytics.

Dr. Kwong was elevated to the IEEE Fellow for his contributions on optimization techniques for cybernetics and video coding in 2014. He was appointed as an IEEE Distinguished Lecturer for the IEEE Systems, Man and Cybernetics (SMC) Society in 2017. He is also the Vice President of the IEEE SMC on Cybernetics.



Zhenyu Lu received the B.Sc. degree in electricity and the M.Sc. degree in information and communication from the Nanjing Institute of Meteorology, Nanjing, China, in 1999 and 2002, respectively, and the Ph.D. degree in optics engineering from the Nanjing University of Science and Technology, Nanjing, in 2008.

He was a Research Associate with the Department of Mathematics and Statistics, University of Strathclyde, Glasgow, U.K., from 2012 to 2013. He is currently a Professor with the School of Electronic and Information Engineering, Nanjing University of Information Science and Technology, Nanjing. He has published 15 international journal papers. His current research interests include neural networks, stochastic control, and artificial intelligence.

Henry Been-Lirn Duh received the Ph.D. degree in engineering from the University of Washington, Seattle, WA, USA, in 2001. He did his postdoctoral training in the NASA-JSC involved in a virtual reality project.

He is currently the Head of the Department of Computer Science and Information Technology, La Trobe University, Melbourne, VIC, Australia. He has published more than 150 conference and journal papers in HCI area. His current research interests include behavioral issues and design in augmented/virtual reality environments.

Mr. Duh is a fellow of the British Computer Society and the Institution of Engineering and Technology, a Companion of Engineer Australia, an ACM Distinguished Speaker, and the Co-Chair of the IEEE Systems, Man and Cybernetics TC on Visual Analytics and Communication.

Jun Zhang (F'17) received the Ph.D. degree from the City University of Hong Kong, Hong Kong, in 2002.

He is currently a Visiting Scholar with Victoria University, Melbourne, VIC, Australia. His current research interests include computational intelligence, cloud computing, high-performance computing, operations research, and power electronic circuits.

Dr. Zhang was a recipient of the Changjiang Chair Professor from the Ministry of Education, China, in 2013, the China National Funds for Distinguished Young Scientists from the National Natural Science Foundation of China in 2011, and the First-Grade Award in Natural Science Research from the Ministry of Education, China, in 2009. He is currently an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the IEEE TRANSACTIONS ON CYBERNETICS, and the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS.