Knowledge Learning for Evolutionary Computation

Yi Jiang, *Student Member*, *IEEE*, Zhi-Hui Zhan, *Senior Member*, *IEEE*, Kay Chen Tan, *Fellow*, *IEEE*, and Jun Zhang, *Fellow*, *IEEE*

***Abstract*—Evolutionary computation (EC) is a kind of meta-heuristic algorithm that takes inspiration from natural evolution and swarm intelligence behaviors. In the EC algorithm, there is a huge amount of data generated during the evolutionary process. These data reflect the evolutionary behavior and therefore mining and utilizing these data can obtain promising knowledge for improving the effectiveness and efficiency of EC algorithms to better solve optimization problems. Considering this and inspired by the ability of human beings that acquire knowledge from the historical successful experiences of their predecessors, this paper proposes a novel EC paradigm, named knowledge learning EC (KLEC). The KLEC aims to learn from historical successful experiences to obtain a knowledge library and to guide the evolutionary behaviors of individuals based on the knowledge library. The KLEC includes two main processes named "*learning from experiences to obtain knowledge*" and "*utilizing knowledge to guide evolution*". First, KLEC maintains a knowledge library model and updates this model by learning the successful experiences collected in every generation. Second, KLEC not only adopts the evolutionary operation but also utilizes the knowledge library model to guide individuals for better evolution. The KLEC is a generic and effective framework, and we propose two algorithm instances of KLEC, which are knowledge learning-based differential evolution and knowledge learning-based particle swarm optimization. Also, we combine the knowledge learning framework with several state-of-the-art EC algorithms, showing that the performance of the state-of-the-art algorithms can be significantly enhanced by incorporating the knowledge learning framework.**

***Index Terms*—Evolutionary computation, knowledge learning, differential evolution, particle swarm optimization, neural network, knowledge library**

Manuscript received XXXX; revised XXXX; accepted XXXX. This work was supported in part by the National Key Research and Development Program of China under Grant 2022ZD0120001, in part by the National Natural Science Foundations of China (NSFC) under Grant 62176094, in part by the Guangdong Natural Science Foundation Research Team under Grant 2018B030312003, and in part by the National Research Foundation of Korea (NRF-2022H1D3A2A01093478). (*Corresponding authors: Zhi-Hui Zhan; Jun Zhang.*)

Yi Jiang and Zhi-Hui Zhan are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China (e-mail: zhanapollo@163.com).

Kay Chen Tan is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR (e-mail: kctan@polyu.edu.hk).

Jun Zhang is with the Key Laboratory of Intelligent Education Technology and Application of Zhejiang Province, Zhejiang Normal University, and also with the Hanyang University, ERICA, South Korea.

I. Introduction

Evolutionary computation (EC) is a kind of meta-heuristic algorithm that takes inspiration from natural evolution and swarm intelligence behaviors. Currently, EC has been developed rapidly because of its effectiveness and efficiency in solving optimization problems [1]-[3]. EC algorithms mainly include two branches [4], which are evolutionary algorithms such as differential evolution (DE) [5]-[7] and genetic algorithm [8]-[10] and swarm intelligence such as particle swarm optimization (PSO) [11]-[13] and ant colony optimization [14]-[16]. During the evolutionary process of the EC algorithms, many data are generated, which can either explicitly or implicitly reveal the evolutionary behavior of the individuals. For example, in the evolutionary process of DE, the successful differential vector reveals the successful behavior of each individual. In PSO, the successful velocities can guide the particles to approach the global optimum.

Through mining these data, the knowledge that can assist the EC algorithms to achieve effective and efficient evolution can be obtained, leading to a new EC paradigm named learning-aid evolution for optimization (LEO) [17]. Currently, many studies in the research area of EC have noted that appropriately utilizing these data can greatly enhance the performance of EC algorithms. First, some existing algorithms for evolutionary transfer optimization [18]-[20] were proposed to transfer and utilize the data generated during the evolutionary process of solving other problems to assist the optimization process of the current problem. We simply denote these algorithms as *cross-problem data utilizing algorithms*. Second, another class of algorithms, which we simply denote as *historical data utilizing algorithms*, collects the data generated in the past generations and utilizes them to enhance the effectiveness of the evolution in the current generation. For example, Zhang *et al.* [21] designed a directional mutation operator, which randomly reused the historical evolutionary directions to help better generate offspring. Ghosh *et al.* [22] proposed a differential vector reuse mechanism, which collected successful differential vectors of DE in an archive and randomly selected vectors in the archive as new differential vectors to guide evolution. The study of such data utilizing EC algorithms has become a new research frontier in the field of EC. The encouraging performance of these algorithms reveals that utilizing data/knowledge to guide the evolution can greatly enhance effectiveness and efficiency.

Although both the *cross-problem data utilizing algorithms* and the *historical data utilizing algorithms* generally achieve

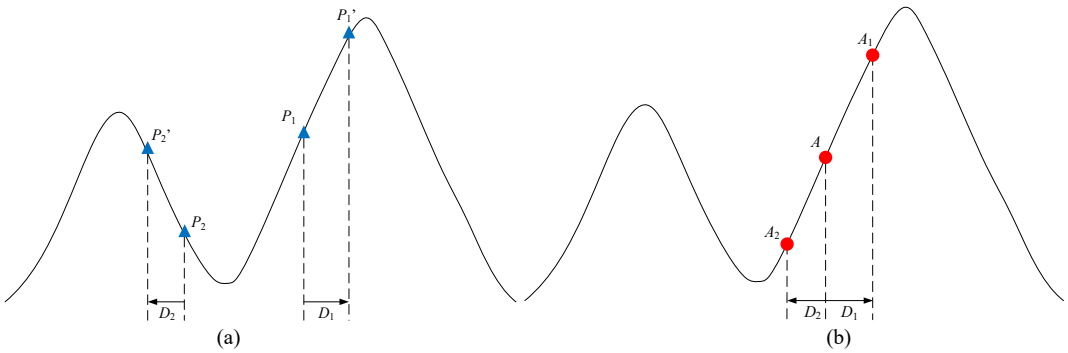


Fig. 1. An example of historical direction utilizing. (a) Two successful directions D_1 and D_2 in the past generations. (b) After applying the two directions D_1 and D_2, two offspring A_1' and A_2' of individual A are generated. However, since the direction D_2 is not suitable for A, the offspring A_2 is worse than A.

promising performance, these kinds of data utilizing strategies only collect and reuse data without deeply mining the knowledge among the data, which are tentative and still have limitations. The two most intuitive and vital limitations are: first, since there is a huge amount of data generated during the past generations, most existing algorithms just simply collect and reuse a very limited amount of them to guide evolution. However, it is hard to obtain general knowledge about successful evolutionary behavior without deeply mining a huge amount of data. Second, most existing algorithms just simply randomly use the data to guide the evolution of the individuals, but without considering the influence of the deep relationship between the individuals and the data.

For clarity, we illustrate an example of a maximum optimization problem in Fig. 1 to show the limitations of the current data utilizing algorithms, where the successful evolutionary directions are collected and utilized. Herein, the individuals in the past generation are illustrated as blue triangles (i.e., P_1, P_1', P_2, and P_2'), and the current individual A and its two offspring A_1 and A_2 are illustrated as red circulars. Two successful evolutionary directions (i.e., D_1 and D_2) are collected, where D_1 is a successful evolutionary direction from P_1 to P_1', and D_2 is a successful evolutionary direction from P_2 to P_2'. Since the individual randomly selects a successful direction and its offspring are generated along this direction. If individual A selects direction D_1, a promising offspring A_1 which achieves better fitness than its parent A will be generated. However, if D_2 is selected, an offspring A_2 with poor fitness will be generated. That is because the start point of D_2 (i.e., the point P_2) is located in another peak far away from A, and therefore the direction D_2 (i.e., successful experience of P_2) is not suitable for A to generate promising offspring. Herein, direction D_1 whose start point P_1 is nearby individual A is more suitable. Therefore, in such a case, the current data utilizing algorithms may select a relatively worse direction to guide the evolution.

To overcome the two above limitations, it is required to design a novel mechanism that not only can deeply mine the knowledge but also can properly utilize the knowledge. We note that human beings can preserve all the successful experiences of their predecessors by summarizing the experiences as many kinds of knowledge. These kinds of knowledge are preserved in a knowledge library, and people can search for and learn suitable knowledge from the library according to their properties to improve themselves.

Inspired by this, this paper proposes a novel and effective knowledge learning (KL) framework for EC. Different from existing data utilizing algorithms that just simply collect and reuse the data, the KL framework is able to deeply mine the knowledge from successful experiences and properly utilize the knowledge to guide the individuals according to their position. Herein, the relationships of data, experience, and knowledge of this paper are briefly clarified. Data usually denotes the information that reveals the evolutionary behavior of the population during evolution. For example, the position information, the direction information, and the fitness information are all data. A successful experience is defined as the successful evolutionary direction at a position. For example, assume that an individual locates at position P_1 with fitness value F_1 jumps to position P_2 with fitness value F_2. If F_2 is better than F_1, which means this individual achieves a successful evolution, then the position P_1 and the successful direction $D = P_2 - P_1$ are paired as a successful experience, denoted as (P_1, D). Moreover, knowledge is defined as a kind of rule about how to obtain successful evolutionary directions by mining from successful experiences. Therefore, the successful experience is a special kind of data generated during the EC evolutionary process, which then can be used for mining the knowledge, so that the knowledge can be used for guiding the evolution of the EC algorithms.

To achieve the goal of KL-based EC algorithms, the KL framework has two processes, "*learning from experiences to obtain knowledge*" and "*utilizing knowledge to guide evolution*". First, in the process of learning from experiences to obtain knowledge, the KL framework maintains a knowledge library model (KLM) based on a feedforward neural network (FNN) to preserve the knowledge. During the evolutionary process, the successful experiences obtained by all the individuals are collected, and these experiences are mined and learned by the KLM to obtain general knowledge about the relation between the individuals and the successful experiences. Second, in the process of utilizing knowledge to guide evolution, individuals can inquire from KLM about the guidance, and the KLM gives each individual a suitable evolutionary direction according to the learned knowledge and the position of the individual. Characteristics and contributions of the proposed KL framework are summarized as follows:

1) A novel and effective KL framework is proposed in this paper. The KL framework can deeply mine successful experiences generated during the evolution to obtain knowledge and can properly utilize the knowledge to guide

individuals according to their position. First, the knowledge in the KL model is more general and effective to guide the evolution, since the knowledge is obtained by mining a huge amount of successful experiences. Second, the KL framework can provide relatively effective guidance of evolutionary direction to each individual, since the provided direction is calculated based on the knowledge and the current status of the individual.

2) The KL framework is a generic framework for EC algorithms and can be easily embedded with many EC algorithms. To clearly show how to combine the KL framework and EC algorithm, we combine the KL framework with two representative EC algorithms, DE and PSO, to propose KL-based DE (denoted as KLDE) and KL-based PSO (denoted as KLPSO). According to the experimental results, these two KL-based EC algorithms are more effective and efficient than their canonical versions.

3) To further evaluate the effect of the KL framework, we combine the KL framework with several state-of-the-art and even champion EC algorithms and show the performance improvement of the KL-based algorithms compared to the original algorithms. The experimental results on both the benchmark functions and the real-world optimization problems indicate our proposed KL framework can significantly improve the performance of these EC algorithms.

The remainder of this paper is organized as follows. We first introduce two representative EC algorithms (i.e., DE and PSO) and the related works on EC algorithms in Section II. Then the KL framework is detailed in Section III, as well as KLDE and KLPSO. Experimental results and analyses are given in Section IV. Section V summarizes the conclusion of this paper and outlines our future works.

II. Background

A. EC

EC algorithms can be divided into two categories, including evolutionary algorithms and swarm intelligence. Herein, two typical EC algorithms such as DE [5] in evolutionary algorithm and PSO [11] in swarm intelligence are introduced.

1) DE

In the initialization operation, each individual is randomly generated according to

$$\boldsymbol{x}_{i,j} = \text{rand}(0,1) \times (U_j - L_j) + L_j \tag{1}$$

where $\boldsymbol{x}_{i,j}$ is the j^{th} dimension of the i^{th} individual, U_j and L_j are the upper bound and lower bound of the j^{th} dimension respectively.

After initialization, the evolutionary operations of the DE that include mutation, crossover, and selection are executed. In the mutation operation, differential vector $\boldsymbol{v}_i$ is generated via the mutation of several parental individuals, as

$$\boldsymbol{v}_i = \boldsymbol{x}_{r1} + F \times (\boldsymbol{x}_{r2} - \boldsymbol{x}_{r3}) \tag{2}$$

where $r1$, $r2$, and $r3$ are mutually exclusive indexes and are randomly selected, F indicates the amplifier factor.

After mutation, the newly generated differential vector $\boldsymbol{v}_i$ is forwarded to the crossover operation. The differential vector $\boldsymbol{v}_i$ crossovers with the current individual $\boldsymbol{x}_i$ to construct trial vector $\boldsymbol{u}_i$ via

$$\boldsymbol{u}_{i,j} = \begin{cases} \boldsymbol{v}_{i,j}, & \text{if rand}(0,1) \le CR \text{ or } j = j_{rand} \\ \boldsymbol{x}_{i,j}, & \text{otherwise} \end{cases} \tag{3}$$

where CR indicates the value of the crossover rate, and j_{rand} is a randomly selected dimension.

To preserve better individuals and discard worse individuals, the selection operation is executed to select the better one between trail vector $\boldsymbol{u}_i$ and current individual $\boldsymbol{x}_i$. The selection operation for solving a minimization optimization problem is shown as

$$\boldsymbol{x}_i = \begin{cases} \boldsymbol{u}_i, & \text{if } f(\boldsymbol{u}_i) < f(\boldsymbol{x}_i) \\ \boldsymbol{x}_i, & \text{otherwise} \end{cases} \tag{4}$$

2) PSO

Different from the DE algorithm, particles in PSO evolve according to the position information of other particles and the best particle. Each particle i in PSO maintains a current position $\boldsymbol{x}_i$, a velocity $\boldsymbol{v}_i$, and a personal best position $\boldsymbol{pbest}_i$. The global best position of the population is denoted by $\boldsymbol{gbest}$.

At the beginning of PSO, the initialization operation is carried out to randomly generate the position $\boldsymbol{x}_i$ and the velocity $\boldsymbol{v}_i$. Then, the particle uses a velocity update operation and a position update operation to update its velocity and position by learning from $\boldsymbol{pbest}_i$ and $\boldsymbol{gbest}$. The velocity update operation is shown as

$$\boldsymbol{v}_i = w \cdot \boldsymbol{v}_i + c_1 \cdot r_1 \cdot (\boldsymbol{pbest}_i - \boldsymbol{x}_i) + c_2 \cdot r_2 \cdot (\boldsymbol{gbest} - \boldsymbol{x}_i) \tag{5}$$

where w is the inertia weight, which controls the proportion of velocity attenuation, c_1 and c_2 are two coefficients. r_1 and r_2 are randomly sampled from [0, 1] independently for each dimension.

The position update operation is carried out to adjust the position $\boldsymbol{x}_i$ of each particle according to its current velocity $\boldsymbol{v}_i$ as

$$\boldsymbol{x}_i = \boldsymbol{x}_i + \boldsymbol{v}_i \tag{6}$$

After the position update, the fitness of each particle is evaluated, and $\boldsymbol{pbest}_i$ and $\boldsymbol{gbest}$ are also updated if necessary.

B. Related Works

To enhance the EC algorithms' performance, researchers have proposed many data utilizing algorithms. We simply classify the existing methods into two categories: cross-problem data utilizing algorithms and historical data utilizing methods.

First, we give a brief introduction to several existing cross-problem data utilizing algorithms. The cross-problem data utilizing algorithms utilize the data and information generated in the solving process of the other optimization problems to guide the evolutionary process of the current problem. For example, Gupta *et al.* [23] proposed a multifactorial optimization framework to utilize individuals of different optimization problems as parents to generate offspring. This way, the data on different problems can be shared to guide the evolution of the current problems. This idea of cross-problem individuals utilizing strategy is also adopted in many existing algorithms for evolutionary multi-task optimization, such as multifactorial evolutionary algorithm with adaptive knowledge transfer [24], evolutionary multitasking via explicit autoencoding [25], and adaptive evolutionary multi-task optimization framework [26]. Moreover, some cross-problem data utilizing algorithms were proposed to utilize the data of evolutionary directions from the

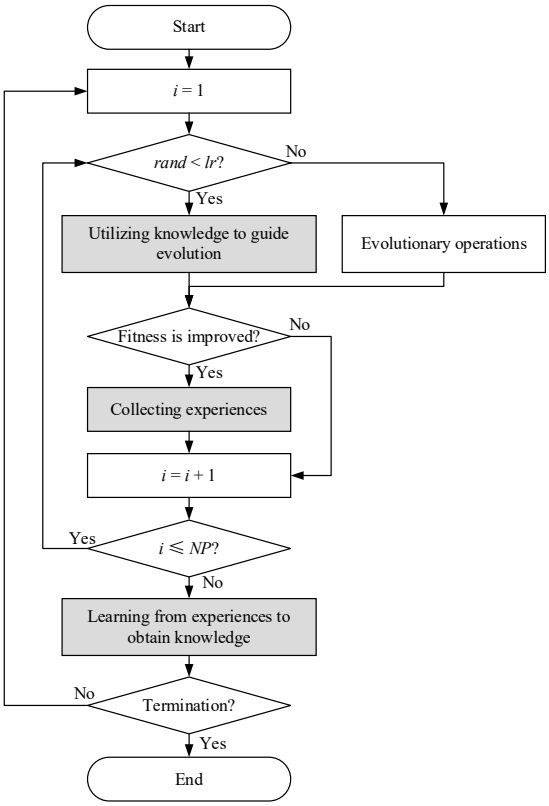


Fig. 2. The general framework of KLEC.

other problem to generate offspring of the current problems. For example, Yin *et al.* [27] proposed to transfer the data of the difference vector from other problems to the current problem to effectively solve the current problem.

Second, several existing historical data utilizing algorithms are introduced here. The historical data utilizing algorithms utilize the historical data generated during the past generations to guide the evolution of the current generation. Several historical data utilizing algorithms were proposed to collect and re-utilize the historical parameter settings or the historical individuals. For example, Zhang *et al.* [28] proposed a JADE, which collected historical successful parameters and generated new parameters based on these successful parameters. Also, in JADE, the past individuals are preserved in a fixed-size archive and are utilized as parental individuals. Tanabe *et al.* [29] further developed the idea of JADE and proposed a success-history based adaptive DE, which maintained several distributions estimated by the successful parameters, and new parameters were sampled from different distributions. Also, based on the idea of JADE and success-history based adaptive DE, some variants were proposed, such as success-history based adaptive DE with linear population size reduction [30] and jSO [31]. Zhan *et al.* [32] proposed an adaptive distributed DE (ADDE), where the parameters for each generation were also adjusted according to the historical parameters and the historical successful individuals were utilized to generate new offspring. Liang *et al.* [33] proposed a self-adaptive dynamic PSO (SaDPSO), where multiple swarms with different parameter settings were evolved concurrently and data of the promising parameter settings was shared among swarms. Zhan and Zhang [34] proposed a PSO-based learning strategy to utilize the historical successful parameters to adaptively control the current parameters. Xue *et al.* [35] proposed an archive-based PSO, where the historical well-performed particles were stored in an archive and the particles in the archive were served as exemplars to update the new particles. Xia *et al.* [36] proposed a triple archive PSO (TAPSO) to preserve historical particles with different properties in different archives and reuse the historical particles to help better evolution. Zhu *et al.* [37] proposed to effectively use the observable parameters and the individuals in historical environments, which achieved encouraging performance in evolutionary dynamic optimization. In their method, the optimal solution in the current environment is predicted via both rote learning and FNN-based strategy, which are trained based on historical parameters and positions.

In addition, some historical data utilizing algorithms were proposed to reuse past successful directions to generate better offspring in the current population. For example, Zhang *et al.* [21] designed a directional mutation operator and incorporated it with classical DE, which was termed directional mutation DE (DMDE). In DMDE, if an individual x achieves a successful evolution to generate a fitter offspring u, the evolutionary direction $(u - x)$ is collected and reused later to generate offspring. In addition, Zhang *et al.* [38] extended DMDE by adopting differential vector archive and the parameter control strategy of success-history based adaptive DE [29]. Ghosh *et al.* [22] proposed a difference vector reuse (DVR) mechanism to utilize the successful differential vector in the past generation to provide data for the current population.

According to the above introduction of existing data utilizing algorithms, we can find that many studies have made great attempts to design different data utilizing strategies to enhance the effectiveness. These existing data utilizing algorithms have achieved promising and encouraging performance on many optimization problems, which indicates the data generated in the evolutionary process contains the knowledge that can effectively guide the evolution. However, these existing data utilizing algorithms still stop at the tier of collecting and utilizing the data, rather than deeply mining the knowledge among the data. This encourages us to propose a novel and effective KL framework to learn and utilize the knowledge rather than just simply collect and utilize the data.

III. KLEC

A. KL Framework

The KL framework includes two main processes, learning from experiences to update knowledge and utilizing knowledge to guide evolution. To clearly describe the KL framework, we illustrate the general framework of the KLEC in Fig. 2.

KL framework maintains the KLM to learn and store the knowledge about the relationship between the individual position and its successful evolutionary direction. Different from the existing algorithms, the proposed KL framework is able to learn all the historical experiences and provide high-quality evolutionary direction to each individual based on the obtained knowledge.

In the beginning, a population with NP individuals is randomly initialized, where NP is the population size. In the main loop of the evolutionary process of KLEC, first, each

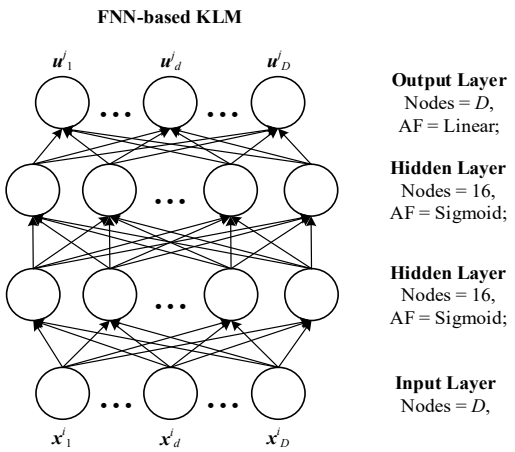


Fig. 3. The architecture of the KLM. The two hidden layers are composed of 16 nodes with sigmoid activation function, the output layer is composed of D (D is the dimension) nodes with linear activation function, and the input dimension of the input layer is D.

individual generates offspring via either utilizing the knowledge of KLM or adopting the evolutionary operations of the EC algorithm. Specifically, if a random number in [0, 1] is smaller than the learning rate *lr*, the individual is selected to utilize the knowledge of KLM. That is, the current position of the individual will be fed to trained KLM as input and the KLM will output a relatively good direction for this individual based on the knowledge and its position. Then, if the fitness of the offspring's new position is better than that of its current position, its successful experience is collected. Finally, the process of learning from experiences to update knowledge is carried out. Specifically, the collected successful experiences are mined and learned by the KLM to update the knowledge about the relationship between the individual position and the successful evolutionary direction.

B. Learning from Experiences to Obtain Knowledge

1) KLM

The main principle of the KLM is to obtain an omniscient knowledge learner that learns all the successful experiences in the past generations to provide appropriate knowledge to individuals according to their current positions. To achieve this, the KLM should be a data structure that can store a huge amount of knowledge and achieve fast knowledge retrieval.

We adopt FNN as the data structure of KLM, as the FNN is a simple but effective model for learning the mapping between position and optimal direction. The information inside the FNN can proceed from the input layer, via the hidden layer, and to the output layer. Recently, a lot of studies and applications have shown that the FNN has good function-fitting ability and pattern recognition ability [39], [40]. The FNN-based KLM can learn the mapping relationship between positions and directions of the experiences. After training, each individual can query for knowledge by feeding its current position to KLM and receiving output data as the evolutionary direction.

Three advantages of the FNN-based KLM are described as follows. First, the FNN-based KLM can get knowledge via learning all the collected successful experiences. For example, if the positions of two experiences are nearby, the KLM can adjust its weights via both these two experiences, thus it can provide suitable direction according to the synthetic of the two directions to the nearby individuals. Second, the FNN-based KLM can provide relatively optimal evolutionary directions to individuals according to their current position. That is because the KLM can learn the knowledge about the relationship between the position and the successful direction, and different input positions for FNN lead to different output evolutionary directions, thus the different individuals will be assigned with different proper evolutionary directions. Third, the FNN-based KLM can achieve relatively fast knowledge retrieval. Once the training process is finished, it is fast to get results from the FNN. To be specific, to get the suitable evolutionary direction from the FNN-based KLM, only the computation of the activation function and computation of the matrix multiplication in the FNN need to be performed, thus the time complexity is relatively low. Therefore, the FNN is a suitable choice for the data structure of KLM.

The architecture of FNN-based KLM is illustrated in Fig. 3. The FNN consists of an input layer, two hidden layers, and an output layer, where the input dimension and the output dimension are both D (D is the problem dimension, i.e., the number of dimensions of decision variables). AF denotes the activation function. Therefore, the output layer contains D nodes, in which the linear activation function is adopted. Besides, each hidden layer contains 16 nodes, in which the sigmoid activation function is adopted to achieve non-linear mapping. The sigmoid activation function is given as follows:

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}} \tag{7}$$

The error backpropagation algorithm is adopted as the supervised learning algorithm to train the FNN [41]. In the backpropagation algorithm, the error between the predicted output of the FNN and the expected output is calculated and backpropagated inside the FNN to adjust its weight. The error is calculated via a loss function, i.e., mean squared error (MSE) loss function [42] as follows:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - f(x_i))^2 \tag{8}$$

where y_i is the expected output of FNN, $f(x_i)$ stands for the predicted output of x_i, and m denotes the number of successful experiences. Besides, to better update the KLM at every generation, the gradient descent optimizer is carried out to optimize the weights of the FNN-based KLM for *ep* epochs in each training process. In the FNN-based KLM, the training data is the collected experiences in every generation. Specifically, the input of the training data is the positions of individuals and the output of the training data is the successful directions corresponding to the positions. For the testing data, the input is the positions of the individuals that are selected to use KLM.

2) Learning from Experiences to Update KLM

In the KL framework, the successful experiences should be first collected before the process of learning from experiences to update KLM. To collect the successful experience, a list Q is created to store the successful experience. Specifically, in each generation, once the fitness value of an individual is improved, this successful experience is collected in Q. The experience will not be learned by KLM immediately until all the successful experiences in the current generation are collected. That is, the

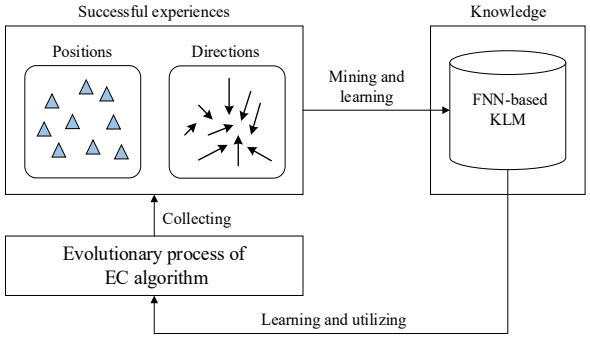


Fig. 4. Illustration of the knowledge learning and utilizing process in KLEC.

experience is temporarily stored in an experience list Q and waits for being learned by KLM.

Then, the successful experiences collected in Q are learned by KLM to obtain knowledge about the relationship between the position and the successful direction. Specifically, the position of each experience in the list Q is fed to KLM as input and the corresponding direction is served as the expected output to train the FNN-based KLM. In the training, the backpropagation algorithm is adopted for *ep* epochs to adjust the weights of KLM. After the learning process of KLM, all the learned experiences in list Q are discarded (i.e., clear all the experiences in list Q at the end of each generation) to make space for the new experiences of the next generation. Noted that, in every generation, KLM is not re-initialized, but is continuously trained by the experiences newly collected in the current generation based on the weights of KLM in the previous generation. Therefore, KLM can in fact obtain knowledge by learning the successful experiences generated over the whole history. If no successful experience is collected in Q, i.e., no individual achieves successful evolution, the KLM will not be updated in the current generation.

C. Utilizing Knowledge to Guide Evolution

The process of utilizing knowledge to guide evolution aims to provide proper evolutionary direction to the individual according to its current position and the learned knowledge. Since the KLM learns about the mapping from the position to the direction according to historical successful experiences, we only need to feed the current position of the individual to the KLM as input, and the output of the KLM is the inquired evolutionary direction for this individual. This way, the utilizing operation of KLM can provide a suitable evolutionary direction to each individual according to its current position. Noted that, if the KLM has not learned any successful experience (e.g., in the first generation or in the first few generations), the KLM is not used to provide directions to individuals. Besides, for clarity, we give Fig. 4 to better illustrate the relationship between the successful experience and the knowledge of KLEC and also to better illustrate the knowledge learning and utilizing process in KLEC.

D. KLDE

To clearly describe how to incorporate the KL framework with the EC algorithms, this section designs a KLDE algorithm by combining the KL framework and DE. The complete KLDE can be found in **Algorithm 1**.

Algorithm 1 KLDE

```
Begin
 1:Initialize population P, experience list Q, and KLM.
 2:While FEs ≤ MaxFEs
 3:    For each individual x_i:
 4:        If rand(0, 1) < lr:
 5:            d_r1 = utilizeKnowledge(KLM, x_r1);  //Section III-C
 6:            v_i = applyDirection(x_r1, d_r1);  //Equation (9)
 7:        Else:
 8:            v_i = mutation(x_r1, x_r2, x_r3, DE/rand/1);  //Equation (2)
 9:        End If
10:        u_i = crossover(v_i, x_i);  //Equation (3)
11:        If f(u_i) < f(x_i):
12:            collecting(Q, x_i, u_i−x_i);
13:            x_i = u_i;
14:        End If
15:    End For
16:    learningKnowledge(KLM, Q);  //Section III-B-2)
17:    clear(Q);
18:End While
End
```

Algorithm 2 KLPSO

```
Begin
 1:Initialize population P, experience list Q, and KLM.
 2:While FEs ≤ MaxFEs
 3:    For each particle x_i:
 4:        temp_i = x_i;
 5:        If rand(0, 1) < lr:
 6:            d_i = utilizeKnowledge (KLM, x_i);  //Section III-C
 7:            x_i = applyDirection(x_i, d_i);  //Equation (9)
 8:        Else
 9:            updateVelocity(x_i, v_i, pbest_i, gbest);  //Equation (5)
10:            x_i = x_i + v_i;  //Equation (6)
11:        End If
12:        If f(x_i) < f(temp_i):
13:            collecting (Q, temp_i, x_i−temp_i);
14:        End If
15:        If f(x_i) < f(pbest_i):
16:            pbest_i = x_i;
17:            If f(pbest_i) < f(gbest):
18:                gbest = pbest_i;
19:            End If
20:        End If
21:    End For
22:    learningKnowledge(KLM, Q);  //Section III-B-2)
23:    clear(Q);
24:End While
End
```

First, a population P, an empty experience list Q to temporarily store the successful experiences, and the KLM are initialized at the beginning of the KLDE (line 1). After initialization, the main loop of the evolutionary process is carried out. Different from the basic DE, in KLDE, each individual in KLDE can evolve via either utilizing knowledge of KLM or evolutionary operations according to the probability lr (lines 5-6). lr is the learning rate, which indicates the probability of evolving individuals via utilizing knowledge of KLM. Specifically, if the current individual $\boldsymbol{x}_i$ is selected to be evolved by utilizing KLM, firstly, a parental individual $\boldsymbol{x}_{r1}$ is randomly selected. To avoid confusion, herein, the individual $\boldsymbol{x}_i$ is named as the current individual, while the individual $\boldsymbol{x}_{r1}$ is named as the parental individual. Secondly, the position of $\boldsymbol{x}_{r1}$ is input to the KLM to get the evolutionary direction $\boldsymbol{d}_{r1}$. Then the parental individual $\boldsymbol{x}_{r1}$ applies the direction provided by KLM to generate a new offspring (line 6). The equation of function applyDirection($\boldsymbol{x}_{r1}$, $\boldsymbol{d}_{r1}$) that is used to apply the evolutionary direction $\boldsymbol{d}_{r1}$ on $\boldsymbol{x}_{r1}$ is given as:

$$\text{applyDirection}(\boldsymbol{x}_{r1}, \boldsymbol{d}_{r1}) = \boldsymbol{x}_{r1} + 2 \times \text{rand}(0,1) \times \boldsymbol{d}_{r1} \quad (9)$$

where rand(0, 1) is a random value generated within [0, 1] to control the influence of the direction. After utilizing the direction, the individual that is out of the search space is set on the boundary.

Otherwise, if the current individual $\boldsymbol{x}_i$ is selected to be evolved via the evolutionary operations, the DE/rand/1 mutation strategy in Eq. (2) is applied to generate differential vector $\boldsymbol{v}_i$ (line 8). Then, the crossover operation of Eq. (3) is carried out between the current individual $\boldsymbol{x}_i$ and the differential vector $\boldsymbol{v}_i$ to generate the trial vector $\boldsymbol{u}_i$ (line 10).

If the generated trial vector $\boldsymbol{u}_i$ is better than the parental individual $\boldsymbol{x}_i$, a successful experience ($\boldsymbol{x}_i$, $\boldsymbol{u}_i$–$\boldsymbol{x}_i$) can be collected. The function collecting(Q, $\boldsymbol{x}_i$, $\boldsymbol{u}_i$–$\boldsymbol{x}_i$) is carried out to collect the success experience ($\boldsymbol{x}_i$, $\boldsymbol{u}_i$–$\boldsymbol{x}_i$) and store it temporarily in the list Q (line 12). After all the individuals finish their evolutionary process in the current generation, the KLM is updated via mining the experiences in list Q to learn the knowledge about the relationship between the position and the successful direction. (line 16). Finally, list Q is emptied to clear all the successful experiences that are already learned by KLM to make room for experiences generated in the next generation.

E. KLPSO

This section incorporates the KL framework with PSO to propose KLPSO, which can be an example algorithm to help clearly describe how to incorporate the KL framework with the swarm intelligence-based algorithms. The pseudo-code of KLPSO can be found in **Algorithm 2**.

In the initialization process of KLPSO, the current position $\boldsymbol{x}_i$ and the velocity $\boldsymbol{v}_i$ are randomly initialized. Additionally, an empty experience list Q is created, and the FNN-based KLM is also randomly initialized (line 1).

After initialization, the main loop of KLPSO is iterated until the termination condition. In the main loop of KLPSO's evolutionary process, first, with the probability lr, the knowledge in KLM is utilized to provide suitable evolutionary direction to the particle, and then the direction is applied to update the position of this particle (lines 6-7). If the particle is not selected to utilize the evolutionary direction provided by KLM, the velocity update (line 9) and the position update (line 10) same as those in the basic PSO algorithm are executed (i.e., Eqs. (5) and (6)). After that, if the new position $\boldsymbol{x}_i$ is better than the former position $temp_i$, successful experience ($temp_i$, $\boldsymbol{x}_i$–$temp_i$) is collected and recorded into the experience list Q (lines 12-14). Finally, the fitness of the new position is compared with the fitness of $pbest_i$ and $gbest$ to update them (lines 15-20). After all the particles in the population have been updated, the process of learning the experience to update KLM is carried out. Specifically, the experiences temporally stored in Q are learned by the KLM to adjust its weights (line 22). Then the list Q is emptied (line 23).

From both KLDE and KLPSO, we can see that KLEC is similar to its basic EC version except for three modifications. Firstly, two kinds of offspring are generated in KLEC by both utilizing knowledge and using traditional evolutionary operations, and therefore the simple if-statement is added. Secondly, a very simple collecting function is added to store the successful experience (i.e., the position and its direction) in the list Q. Thirdly, at the end of every generation, the KLM is updated by the list Q (i.e., the learningKnowledge function in Section III-B-2)) and then the list Q is cleared. Therefore, the KLEC is as easy-used as its basic EC version.

IV. Experimental Verification

A. Experimental Settings

Three different experiments are designed to evaluate the performance of the KL framework. In the first experiment, we compare the experimental results on two canonical EC algorithms (i.e., DE and PSO) and their KL-based variants (i.e., KLDE and KLPSO) to evaluate the effects of the KL framework in improving the performance of basic EC algorithms.

In the second experiment, we compare the results of several state-of-the-art EC algorithms and their KL-based variants. Through this experiment, we can observe that the KL framework can significantly improve the performance of these state-of-the-art algorithms. This way, the effect of the KL framework on performance improvement can be better shown and the effectiveness and efficiency of the KL framework can be further evaluated. The compared state-of-the-art DE-based algorithms include JADE [28], ADDE [32], jSO [31], and hybrid-adaptive DE with decay function (HyDE-DF) [43], while the compared state-of-the-art PSO-based algorithms include SaDPSO [33], hierarchical PSO with time-varying acceleration coefficients (HPSO-TVAC) [44], TAPSO [36], and adaptive weighted PSO (AWPSO) [45]. Many algorithms among these compared algorithms have achieved promising results in many competitions. Specifically, jSO wins second place in the CEC2017 single objective optimization competition, HyDE-DF wins third place in the CEC2019 competition, and SaDPSO wins eighth in the CEC2015 competition, which is the best ranking among the PSO-based algorithms.

In the third experiment, we compare the proposed KL framework with several existing historical data utilizing algorithms. Two recently-proposed historical data utilizing algorithms, the directional mutation operator [21] and the DVR method [22], are adopted in the comparison. Similar to the KL framework, the directional mutation operator and DVR are also easy to be incorporated with DE, termed DMDE and DVR-DE, respectively. The comparison is performed on KLDE, DMDE, and DVR-DE to verify the superiority of the KL framework among historical data utilizing EC algorithms.

The three above experiments are conducted on the well-known CEC2017 single objective test suite [46]. In CEC2017, there are 29 benchmark functions, which include two unimodal functions F_1 and F_2, seven simple multimodal functions F_3-F_9, ten hybrid functions F_{10}-F_{19}, and ten complex composition functions F_{20}-F_{29}. The detailed properties of the CEC2017 test suite can be found in [46].

In the experiments, to reduce accidental error, results are obtained over 51 independent runs for each algorithm. The error value is adopted as the performance metric, which is calculated as the gap between the best fitness obtained by the algorithm and the real optimum fitness. To verify the algorithms' performance in different dimensions, three settings of dimension D = 10, 30, and 50 are adopted. The maximum function evaluations ($MaxFEs$) are adopted as the termination

TABLE I
RESULTS FOR MEAN ERROR OF KLDE, DE, KLPSO, AND PSO AT $D = 30$

Func	KLDE	DE		KLPSO	PSO	
F_1	**0.00E+00**	**0.00E+00**	≈	**4.76E+03**	6.04E+07	≈
F_2	**1.68E-02**	6.25E+01	+	2.63E+02	**7.54E+01**	−
F_3	**5.58E+01**	5.88E+01	≈	**9.27E+01**	1.02E+02	≈
F_4	**2.66E+01**	1.78E+02	+	**6.59E+01**	7.18E+01	≈
F_5	6.46E-03	**0.00E+00**	−	**1.48E-01**	5.60E-01	≈
F_6	**5.80E+01**	2.09E+02	+	1.09E+02	**9.73E+01**	≈
F_7	**2.92E+01**	1.80E+02	+	**6.46E+01**	6.49E+01	≈
F_8	1.44E-01	**0.00E+00**	−	**8.31E-01**	9.30E+00	+
F_9	**2.27E+03**	6.33E+03	+	**2.63E+03**	2.82E+03	≈
F_{10}	**2.56E+01**	6.32E+01	+	**1.07E+02**	1.14E+02	≈
F_{11}	1.46E+04	**9.61E+03**	≈	**3.99E+05**	6.30E+05	≈
F_{12}	**2.39E+01**	8.05E+01	+	**1.44E+04**	5.38E+05	≈
F_{13}	**1.99E+01**	6.16E+01	+	**1.05E+04**	1.71E+04	≈
F_{14}	**5.10E+00**	3.65E+01	+	**5.25E+03**	9.57E+03	≈
F_{15}	**3.69E+02**	6.46E+02	+	**5.90E+02**	6.25E+02	≈
F_{16}	**5.74E+01**	1.02E+02	+	**1.82E+02**	2.32E+02	+
F_{17}	**2.38E+01**	3.78E+01	+	2.38E+05	**1.67E+05**	≈
F_{18}	**6.33E+00**	1.66E+01	+	**8.17E+03**	9.60E+03	≈
F_{19}	7.92E+01	**5.82E+01**	≈	2.89E+02	**2.70E+02**	≈
F_{20}	**2.30E+02**	3.68E+02	+	**2.62E+02**	2.73E+02	+
F_{21}	**8.72E+02**	2.19E+03	≈	**5.42E+02**	1.13E+03	+
F_{22}	**3.86E+02**	5.25E+02	+	**4.71E+02**	**4.71E+02**	≈
F_{23}	**4.60E+02**	5.93E+02	+	5.60E+02	**5.51E+02**	≈
F_{24}	**3.87E+02**	**3.87E+02**	≈	**3.88E+02**	3.90E+02	+
F_{25}	**1.32E+03**	2.55E+03	+	**1.19E+03**	1.49E+03	+
F_{26}	5.05E+02	**4.97E+02**	−	5.41E+02	**5.36E+02**	≈
F_{27}	3.29E+02	**3.19E+02**	−	**4.38E+02**	4.44E+02	≈
F_{28}	**4.58E+02**	5.92E+02	+	6.47E+02	**6.20E+02**	≈
F_{29}	2.07E+03	**2.00E+03**	−	**8.64E+03**	9.62E+03	≈
Number of + / ≈ / −		18 / 6 / 5		+ / ≈ / −	6 / 22 / 1	

condition for the algorithms. The value of *MaxFEs* is $D\times10^4$ for the experiments. Besides, Wilcoxon's rank-sum test at 5% significant level is used to evaluate the experimental results in the statistic view. The notations "+/≈/−" indicate the results obtained by the KL-based variant are significantly "superior/equal/inferior" to those of the original algorithm. In KL framework, the learning rate *lr* is 0.2 and the training epoch *ep* is 10. The population size of the compared algorithms and their KL-based variants are both set as 100. The parameter settings of the compared algorithms are adopted as the recommended values in their papers.

B. Comparisons on DE and PSO

The results of KLDE with DE and KLPSO with PSO are analyzed to show the effectiveness and efficiency of the KL framework. We show the experimental results for the mean error values at $D = 30$ in Table I, while the results at $D = 10$ and 30 are shown in Table S.I and Table S.II of the supplementary material.

From the results, we find that KLDE generally outperforms the original DE at $D = 10$, 30, and 50 on most problems. Also, the performance of KLPSO is generally superior to that of PSO. Moreover, by comparing the results at different dimensions, we can find that with the increase of dimensions, the effects of the KL framework can be better reflected. For example, the experimental results of KLDE are significantly superior to those of DE on 10 functions when $D = 10$, while KLDE is significantly superior to DE on 18 problems on $D = 30$ and 50. That is because the population is often too small to cover the full landscape if the dimension is huge. In this case, only a little data is extracted to generate the offspring, which often leads to the stagnation of pre-mature. However, the KL framework provides extra knowledge with both high-quality and large-quantity. Therefore, KL-based algorithms are generally better than their original versions when encountering high-dimensional problems.

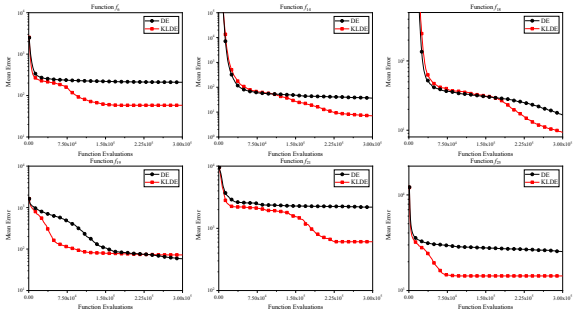


Fig. 5. The convergence graphs of KLDE and DE on f_6, f_{14}, f_{18}, f_{19}, f_{21}, f_{25} when $D = 30$.

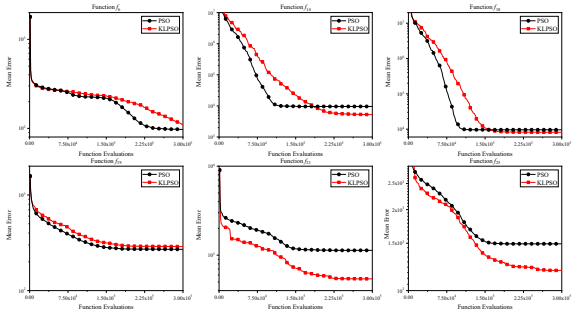


Fig. 6. The convergence graphs of KLPSO and PSO on f_6, f_{14}, f_{18}, f_{19}, f_{21}, f_{25} when $D = 30$.

Besides, the properties of the KL framework over the evolutionary process can be observed through convergence graphs of KLDE, DE, KLPSO, and PSO on some selected problems. The convergence graphs of KLDE and DE at $D = 30$ are shown in Fig. 5, while Fig. 6 illustrates the convergence graphs of KLPSO and PSO at $D = 30$. The convergence graphs of KLDE and DE and the convergence graphs of KLPSO and PSO at $D = 10$ are shown in Fig. S1 and Fig. S2, respectively. The convergence graphs of KLDE and DE and the convergence graphs of KLPSO and PSO at $D = 50$ are shown in Fig. S3 and Fig. S4, respectively. As we can observe from the figures, after several generations, DE and PSO are not able to extract sufficient data to generate promising offspring, which makes the individuals easily pre-mature. However, the KL framework can help the algorithms to jump out of this situation to further optimize the population.

C. Comparisons on State-of-the-Art Algorithms

1) Comparisons of DE-Based Algorithms against Their KL-Based Variants

The experimental results for mean error values at $D = 30$ of DE-based state-of-the-art algorithms (i.e., JADE, ADDE, jSO, and HyDE-DF) and their KL-based variants are shown in Table II, while the results at $D = 10$ and 50 are shown in Table S.III and Table S.IV of the supplementary material.

Experimental results of KL-JADE, KL-ADDE, KL-jSO, and KL-HyDE-DF are generally better than those of their original algorithms: When $D = 30$, the KL-based algorithms achieve the best results on most functions. The KL-JADE significantly dominates JADE on 17 functions; KL-ADDE dominates ADDE on 10 functions; KL-jSO defeats jSO on 13 functions; KL-HyDE-DF defeats HyDE-DF on 20 functions. On the high-dimensional problems, KL-based algorithms also achieve promising performance: When $D = 50$, KL-JADE outperforms JADE on 16 functions; KL-ADDE dominates ADDE on 17

TABLE II
RESULTS FOR MEAN ERROR OF THE DE-BASED ALGORITHMS AND THEIR KL-BASED VARIANTS AT $D = 30$

Func	KL-JADE	JADE		KL-ADDE	ADDE		KL-jSO	jSO		KL-HyDE-DF	HyDE-DF	
F_1	**0.00E+00**	**0.00E+00**	≈	6.18E-02	**3.72E-02**	≈	**0.00E+00**	**0.00E+00**	≈	**8.07E+00**	4.98E+02	+
F_2	**1.12E+05**	1.62E+05	+	1.20E+05	**1.19E+05**	≈	**0.00E+00**	**0.00E+00**	≈	**8.99E-02**	2.99E+01	+
F_3	**2.51E+01**	3.79E+01	+	**5.86E+01**	5.87E+01	+	**5.86E+01**	5.91E+01	+	6.36E+01	**5.65E+01**	≈
F_4	**6.36E+01**	1.83E+02	+	**3.03E+01**	3.21E+01	≈	2.20E+01	**1.02E+01**	−	**3.56E+01**	4.43E+01	+
F_5	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	4.03E-08	**1.48E-08**	−	**1.91E-02**	5.94E-02	+
F_6	**9.65E+01**	2.18E+02	+	7.14E+01	**6.96E+01**	≈	5.32E+01	**3.91E+01**	−	**6.76E+01**	8.45E+01	+
F_7	**5.99E+01**	1.75E+02	+	3.61E+01	**3.48E+01**	≈	2.49E+01	**1.12E+01**	−	**3.40E+01**	4.14E+01	+
F_8	1.13E+01	**7.19E-01**	−	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**9.79E-02**	3.41E+01	+
F_9	**3.98E+03**	8.07E+03	+	**3.68E+03**	4.03E+03	+	1.88E+03	**1.75E+03**	−	**2.00E+03**	2.42E+03	+
F_{10}	9.06E+01	**8.31E+01**	≈	3.57E+01	**3.10E+01**	≈	**3.80E+00**	8.04E+00	+	**3.17E+01**	4.50E+01	+
F_{11}	4.91E+03	**3.17E+03**	−	6.53E+03	**4.90E+03**	−	3.20E+02	**1.44E+02**	−	2.14E+04	**6.12E+03**	−
F_{12}	**5.19E+02**	4.30E+03	+	**2.48E+02**	2.25E+04	≈	**1.13E+01**	2.14E+01	+	6.55E+03	**3.80E+02**	≈
F_{13}	5.93E+04	**2.69E+04**	−	**3.17E+03**	3.32E+04	+	2.37E+01	**2.30E+01**	−	**7.76E+01**	1.03E+02	+
F_{14}	8.58E+03	**3.99E+03**	≈	**1.72E+02**	4.74E+04	+	**1.88E+00**	3.25E+00	+	**1.17E+02**	1.24E+02	+
F_{15}	**9.40E+02**	1.87E+03	+	7.49E+02	**7.47E+02**	≈	**6.58E+01**	1.25E+02	+	4.24E+02	**3.80E+02**	≈
F_{16}	**3.19E+02**	4.31E+02	+	**2.17E+02**	2.53E+02	+	**2.93E+01**	3.88E+01	+	**9.21E+01**	1.02E+02	+
F_{17}	**2.13E+05**	7.27E+05	≈	**5.74E+05**	6.75E+05	+	**2.10E+01**	2.13E+01	+	**2.37E+02**	3.50E+02	+
F_{18}	6.92E+03	**6.14E+02**	−	**4.88E+01**	3.44E+04	≈	**4.20E+00**	7.37E+00	+	6.06E+01	**5.77E+01**	−
F_{19}	**2.87E+02**	5.19E+02	+	**2.34E+02**	2.98E+02	+	**2.49E+01**	3.53E+01	+	**7.31E+01**	1.33E+02	+
F_{20}	**2.63E+02**	4.01E+02	+	**2.35E+02**	2.36E+02	≈	2.24E+02	**2.11E+02**	−	**2.32E+02**	2.41E+02	+
F_{21}	**6.66E+02**	1.36E+03	+	**1.00E+02**	**1.00E+02**	+	**1.00E+02**	**1.00E+02**	+	**1.00E+02**	**1.00E+02**	+
F_{22}	**4.17E+02**	5.33E+02	+	**3.85E+02**	3.86E+02	≈	3.73E+02	**3.48E+02**	−	**3.85E+02**	3.96E+02	+
F_{23}	**5.04E+02**	6.34E+02	+	**4.60E+02**	**4.60E+02**	−	4.46E+02	**4.24E+02**	−	**4.60E+02**	4.67E+02	+
F_{24}	**3.87E+02**	**3.87E+02**	−	**3.87E+02**	**3.87E+02**	+	**3.87E+02**	**3.87E+02**	+	3.93E+02	**3.86E+02**	−
F_{25}	**1.55E+03**	2.49E+03	+	1.24E+03	**1.21E+03**	−	1.20E+03	**9.13E+02**	−	**1.06E+03**	1.27E+03	+
F_{26}	**5.03E+02**	5.07E+02	+	5.06E+02	**5.02E+02**	≈	4.98E+02	**4.94E+02**	−	5.11E+02	**5.08E+02**	≈
F_{27}	3.69E+02	**3.47E+02**	≈	3.69E+02	**3.25E+02**	−	**3.00E+02**	3.09E+02	+	**3.14E+02**	3.45E+02	+
F_{28}	**6.40E+02**	1.27E+03	+	**5.70E+02**	5.83E+02	≈	**4.33E+02**	4.46E+02	+	5.34E+02	**5.28E+02**	≈
F_{29}	2.23E+03	**2.20E+03**	≈	**3.46E+03**	7.91E+03	+	1.98E+03	**1.97E+03**	−	3.04E+03	**2.94E+03**	−
Number of + / ≈ / −		17 / 7 / 5		+ / ≈ / −	10 / 15 / 4		+ / ≈ / −	13 / 3 / 13		+ / ≈ / −	20 / 5 / 4	

functions; KL-HyDE-DF dominates HyDE-DF on 17 functions. Although the performance of KL-jSO is inferior to jSO at $D = 50$ (KL-jSO achieves better performance than jSO on 9 cases, and jSO dominates KL-jSO on 11 problems), KL-jSO outperforms jSO at $D = 10$ and $D = 30$. Therefore, with the help of the KL framework, the KL-based DE variants can achieve significantly better performance than the original algorithms.

2) Comparisons of PSO-Based Algorithms against Their KL-Based Variants

The experimental results for mean error values at $D = 30$ of PSO-based state-of-the-art algorithms (i.e., SaDPSO, HPSO-TVAC, TAPSO, and AWPSO) with their KL-based variants can be found in Table III, while the results at $D = 10$ and $D = 50$ are shown in Table S.V and Table S.VI of the supplementary material.

For the comparison between KL-SaDPSO and SaDPSO, the performance of KL-SaDPSO dominates that of SaDPSO at $D = 30$ and $D = 50$ (KL-SaDPSO defeats SaDPSO on 4 functions both at $D = 30$ and $D = 50$). For the comparison between KL-HPSO-TVAC and HPSO-TVAC, in most functions, the performance of KL-HPSO-TVAC dominates that of HPSO-TVAC whatever at $D = 10$, 30, or 50. Besides, KL-TAPSO and KL-AWPSO also outperform TAPSO and AWPSO, respectively. Specifically, at $D = 30$, KL-TAPSO significantly outperforms TAPSO at 5 problems, and KL-AWPSO significantly outperforms AWPSO at 11 problems. Moreover, comparing the experimental results of these algorithms at different dimensions, the gap between the performance of the original algorithms and the KL-based variants increases with the increasing of problem dimension. For example, the number of functions on which the experimental results of KL-AWPSO are better than those of AWPSO is 3 at $D = 5$, at $D = 30$ the number is 11, and at $D = 50$ the number is 17.

D. Comparisons among KLDE, DMDE, and DVR-DE

The detailed results for mean error values at $D = 30$ obtained by KLDE, DMDE, and DVR-DE are given in Table IV, and the results obtained by KLDE, DMDE, and DVR-DE at $D = 10$ and 50 are given in Table S.VII and Table S.VIII of the supplementary material. The KLDE obtains generally superior results than these two historical data utilizing methods. The KLDE dominates DMDE on 13, 20, and 19 functions at $D = 10$, 30, and 50, respectively. KLDE defeats DVR-DE on 11, 17, and 14 functions at $D = 10$, 30, and 50, respectively.

Due to the above experiments, the efficiency and effectiveness of the KL framework are shown. We conclude that, firstly, with the help of the KL framework, the performance of the KL-based algorithm is greatly improved when compared with the original algorithm. Secondly, if the dimension of function becomes large and the population can not cover the full landscape to extract useful data from the current population, the KL framework can still provide extra knowledge. From the convergence graphs, when the KL-based algorithm meets the stagnant of pre-mature that the basic algorithm can not jump out, the KL-based EC algorithms still can improve the fitness of offspring to further evolve. Thirdly, by the comparison among KLDE, DMDE, and DVR-DE, we can conclude that the KL framework dominates the state-of-the-art historical data utilizing methods.

E. Sensitivity Analysis

1) Sensitivity Analysis on Parameter lr

The learning rate *lr* in the KL framework determines the proportion of individuals that learn the knowledge of KLM. Intuitively, the learning rate *lr* can influence the exploration and exploitation abilities of KL-based algorithms. If the *lr* is relatively large, most of the individuals acquire knowledge from KLM, and only a small proportion of individuals can

TABLE III
RESULTS FOR MEAN ERROR OF THE PSO-BASED ALGORITHMS AND THEIR KL-BASED VARIANTS AT $D = 30$

Func	KL-SaDPSO	SaDPSO		KL-HPSO-TVAC	HPSO-TVAC		KL-TAPSO	TAPSO		KL-AWPSO	AWPSO	
F_1	**8.82E+01**	4.98E+02	≈	**1.15E+07**	2.01E+07	+	**2.19E+03**	2.46E+03	≈	**4.85E+03**	1.16E+08	≈
F_2	1.32E-05	**7.10E-06**	≈	**1.91E+02**	3.46E+02	+	**9.89E+00**	3.48E+01	≈	2.71E-02	**3.51E-04**	−
F_3	**1.56E+00**	1.72E+00	≈	**9.94E+01**	1.03E+02	≈	**4.91E+01**	6.26E+01	+	**8.58E+01**	1.03E+02	+
F_4	**4.82E+01**	4.85E+01	≈	**1.26E+02**	1.29E+02	≈	4.62E+01	**4.16E+01**	≈	**4.60E+01**	5.21E+01	+
F_5	**9.58E-03**	3.35E-02	+	**1.23E+01**	1.33E+01	≈	1.96E-06	**3.51E-07**	−	**1.59E-01**	5.43E-01	+
F_6	7.93E+01	**7.41E+01**	≈	**1.96E+02**	2.16E+02	+	**6.64E+01**	6.95E+01	≈	**6.62E+01**	8.41E+01	+
F_7	5.27E+01	**5.22E+01**	≈	**1.00E+02**	1.04E+02	≈	**4.30E+01**	4.57E+01	≈	**4.10E+01**	5.14E+01	+
F_8	**3.17E+00**	8.87E+00	+	1.20E+03	**8.88E+02**	≈	**1.10E-01**	2.65E-01	+	**3.02E+00**	5.57E+00	+
F_9	2.54E+03	**2.29E+03**	−	3.23E+03	**3.13E+03**	≈	**2.37E+03**	2.48E+03	≈	**2.48E+03**	2.69E+03	≈
F_{10}	**7.55E+01**	8.12E+01	≈	1.24E+02	**1.20E+02**	≈	5.26E+01	**4.93E+01**	≈	9.56E+01	**9.02E+01**	≈
F_{11}	**1.84E+03**	2.19E+03	≈	**3.27E+06**	3.67E+06	≈	1.65E+05	**3.09E+04**	−	**1.24E+05**	1.35E+06	+
F_{12}	1.46E+03	**1.42E+03**	≈	**5.01E+05**	1.19E+06	+	**1.31E+04**	1.39E+04	≈	**2.04E+04**	7.22E+05	+
F_{13}	4.75E+02	**4.38E+02**	≈	**9.44E+03**	1.26E+04	≈	**2.67E+03**	6.90E+03	+	1.00E+04	**5.99E+03**	≈
F_{14}	**3.23E+02**	8.64E+02	≈	**1.03E+05**	1.41E+05	≈	**2.11E+03**	3.12E+03	≈	**9.23E+03**	1.16E+04	≈
F_{15}	5.52E+02	**5.14E+02**	≈	**9.53E+02**	1.02E+03	≈	7.24E+02	**7.03E+02**	≈	**5.57E+02**	6.34E+02	≈
F_{16}	1.35E+02	**1.34E+02**	≈	**3.04E+02**	4.06E+02	+	**1.46E+02**	1.62E+02	≈	2.16E+02	**2.10E+02**	≈
F_{17}	2.69E+03	**2.67E+03**	≈	**2.17E+05**	2.26E+05	≈	1.08E+05	**1.05E+05**	≈	2.37E+05	**1.48E+05**	−
F_{18}	**3.25E+02**	3.44E+02	+	**1.28E+05**	1.37E+05	≈	**4.01E+03**	5.83E+03	+	**7.59E+03**	3.00E+04	≈
F_{19}	1.96E+02	**1.94E+02**	≈	**3.32E+02**	3.34E+02	≈	**1.97E+02**	2.10E+02	≈	2.56E+02	**2.37E+02**	≈
F_{20}	**2.50E+02**	2.53E+02	≈	**3.29E+02**	3.44E+02	+	**2.41E+02**	2.42E+02	≈	**2.51E+02**	2.54E+02	≈
F_{21}	1.43E+02	**1.00E+02**	−	1.33E+03	**8.49E+02**	≈	**1.00E+02**	1.60E+02	+	1.19E+03	**7.21E+02**	≈
F_{22}	**4.05E+02**	4.10E+02	≈	**6.65E+02**	7.34E+02	+	**3.99E+02**	4.02E+02	≈	**4.49E+02**	4.61E+02	+
F_{23}	**4.76E+02**	4.84E+02	≈	**8.54E+02**	9.53E+02	+	4.76E+02	**4.74E+02**	≈	**5.37E+02**	5.45E+02	≈
F_{24}	3.83E+02	**3.81E+02**	≈	**3.95E+02**	3.99E+02	≈	3.89E+02	**3.88E+02**	≈	**3.88E+02**	3.91E+02	+
F_{25}	**6.06E+02**	8.30E+02	≈	**2.53E+03**	2.55E+03	≈	**1.35E+03**	1.40E+03	≈	**1.13E+03**	1.16E+03	≈
F_{26}	**5.07E+02**	5.17E+02	+	**6.85E+02**	7.37E+02	≈	5.10E+02	**5.09E+02**	≈	**5.33E+02**	5.41E+02	≈
F_{27}	3.32E+02	**3.23E+02**	≈	4.12E+02	**4.07E+02**	≈	3.43E+02	**3.35E+02**	−	**4.24E+02**	4.44E+02	+
F_{28}	**5.90E+02**	6.09E+02	≈	**9.03E+02**	9.29E+02	≈	**5.15E+02**	5.30E+02	≈	6.48E+02	**6.43E+02**	≈
F_{29}	1.35E+04	**6.37E+03**	−	**1.45E+05**	1.91E+05	≈	**3.83E+03**	3.92E+03	≈	9.05E+03	**7.77E+03**	≈
Number of +/≈/−		4 / 22 / 3		+/≈/−	8 / 21 / 0		+/≈/−	5 / 22 / 3		+/≈/−	11 / 16 / 2	

explore undiscovered regions via evolutionary operations. In this case, the exploration ability of KL-based algorithms will be weakened. If the *lr* is specified as a very small value, most individuals are evolved according to the evolutionary operation. In this case, the KL framework takes no effect, thus the exploitation ability reduces.

Therefore, the learning rate *lr* should be studied and carefully set. In the above experiments, *lr* is set as 0.2. To study the performance of the KL framework with different settings of *lr* and find the optimal setting of *lr*, we compare the performance of the KLDE and KLPSO with $lr = 0.2$ and the variants with $lr = 0.1$, 0.3, 0.4, and 0.5. The comparison results obtained by these KLDE variants and KLPSO variants with different settings of *lr* at $D = 30$ and $ep = 10$ are given in Table V. We can find that by considering the results on both KLDE and KLPSO, the $lr = 0.2$ achieves a generally better performance than other *lr* values. Therefore, both considering the results and the famous "80%/20% rule" [47], [48], *lr* is recommended to be set as 0.2 in the KL framework.

2) Sensitivity Analysis on Parameter ep

The *ep* indicates the training epochs of the FNN-based KLM in each generation. Similar to the setting of *lr*, *ep* also affects the performance of the KL framework. On the one hand, if the *ep* is too small, the KLM will be under-fitting and show poor performance. On the other hand, if the *ep* is set as a relatively large number, the KLM may be over-fitting and provides the wrong evolutionary directions.

In the above experiments, the value of *ep* is set as 10. Herein, to show whether the setting $ep = 10$ is superior to other settings of *ep*, we compare the performance of $ep = 10$ and $ep = 1$, 5, and 20. The experimental results of these KLDEs and KLPSOs with different values of *ep* at $D = 30$ and $lr = 0.2$ are given in Table VI. The KLDE with $ep = 1$ shows the worst performance since KLM in KLDE is under-fitting. In the other aspect, with the $ep = 5$ or $ep = 20$, KLDE can achieve comparative performance to the KLDE with $ep = 10$. However, KLPSO with $ep = 10$ can achieve better performance than KLPSO with $ep = 5$ or 20. Moreover, $ep = 10$ means the KLM only needs to be trained for 10 epochs, whose time complexity is lower than that of the KL framework with $ep = 20$. Therefore, the *ep* is set as 10 by both considering the performance and the time complexity.

F. Analysis of KLM Architecture

In the proposed KL framework, the KLM is based on an FNN, and the architecture of the KLM is shown in Fig. 3. The architecture of the KLM can influence the performance of the KL framework. To analyze the settings of KLM architecture, we compare the performance of the KLM architecture in Fig. 3 and that of the other KLM architectures. Specifically, KLDE and KLPSO variants with different KLM architectures are compared, where the node number of each hidden layer is set as 4, 8, 16, and 32 (denoted as $NN = 4$, 8, 16, and 32), respectively.

The experimental results obtained by KLDE variants with $NN = 4$, 8, 16, and 32 at $D = 30$ are shown in Table S.IX, while results obtained by KLPSO and KLPSO variants with $NN = 4$, 8, 16, and 32 at $D = 30$ are shown in Table S.X of supplementary material. To analyze the results, the number of best results (denoted as NoB) and the mean rank of each algorithm are adopted as statical metrics. According to the results, we can find that the settings of $NN = 16$ and 32 achieve relatively promising performance. Specifically, according to NoB, the KLDE with $NN = 32$ and KLPSO with $NN = 16$ achieves the best performance among the variants, while the KLDE with $NN = 16$ and KLPSO with $NN = 32$ get the best results according to the mean rank. However, when $NN = 32$, the time complexity of the KLM can be relatively large, and executing the KL framework can consume much time. Therefore, both considering the experimental results and the time complexity, $NN = 16$ is the optimal setting.

TABLE IV
RESULTS FOR MEAN ERROR OF KLDE, DMDE, DVR-DE AT $D = 30$

Func	KLDE	DMDE		DVR-DE	
F_1	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	≈
F_2	**1.68E-02**	6.09E+00	+	6.11E+00	+
F_3	5.58E+01	**5.36E+01**	≈	5.92E+01	≈
F_4	**2.66E+01**	1.66E+02	+	6.13E+01	+
F_5	6.46E-03	**4.95E-08**	−	5.04E-03	≈
F_6	**5.80E+01**	1.96E+02	+	9.99E+01	+
F_7	**2.92E+01**	1.66E+02	+	6.21E+01	+
F_8	1.44E-01	3.56E-02	+	**1.78E-02**	−
F_9	**2.27E+03**	6.31E+03	+	4.16E+03	+
F_{10}	**2.56E+01**	2.67E+01	≈	2.61E+01	≈
F_{11}	1.46E+04	1.46E+04	≈	**1.06E+04**	≈
F_{12}	**2.39E+01**	8.63E+01	+	4.14E+01	+
F_{13}	**1.99E+01**	5.69E+01	+	3.82E+01	+
F_{14}	**5.10E+00**	1.39E+01	+	1.28E+01	+
F_{15}	**3.69E+02**	4.30E+02	≈	1.07E+03	+
F_{16}	**5.74E+01**	1.09E+02	+	3.68E+02	+
F_{17}	**2.38E+01**	3.27E+01	+	3.05E+01	+
F_{18}	**6.33E+00**	8.87E+00	+	1.13E+01	+
F_{19}	7.92E+01	**6.75E+01**	−	3.47E+02	+
F_{20}	**2.30E+02**	3.54E+02	+	2.57E+02	+
F_{21}	**8.72E+02**	3.57E+03	+	1.45E+03	≈
F_{22}	**3.86E+02**	5.39E+02	+	4.07E+02	+
F_{23}	**4.60E+02**	7.00E+02	+	4.67E+02	≈
F_{24}	**3.87E+02**	**3.87E+02**	≈	**3.87E+02**	−
F_{25}	**1.32E+03**	2.38E+03	+	1.50E+03	+
F_{26}	5.05E+02	5.57E+02	+	**5.03E+02**	≈
F_{27}	**3.29E+02**	1.78E+03	+	3.41E+02	≈
F_{28}	**4.58E+02**	5.57E+02	+	5.45E+02	+
F_{29}	2.07E+03	2.04E+03	≈	**2.03E+03**	≈
Number of + / ≈ / −		20 / 7 / 2		17 / 10 / 2	

G. Runtime Analysis

This section analyzes the runtime of the KL-based algorithms. In the runtime analysis, the average cost of CPU time consumed by each algorithm is used as the metric. The comparison of the runtime is conducted on KLDE, DE, KLPSO, PSO, KL-JADE, JADE, KL-ADDE, ADDE, KL-TAPSO, KL-AWPSO, and AWPSO. The experimental results with respect to the runtime obtained by these compared algorithms and their KL-based variants at $D = 10$, 30, and 50 are shown in Table S.XI of the supplementary material. We can find that the KL-based algorithms consume more time than the original algorithms. This observation is intuitive since the KL framework contains an FNN-based KLM and utilizing this KLM can consume some time.

Besides, to evaluate whether the time cost is worth it, the "*e*rror reduction versus *t*ime increase *r*ate" (ETR) proposed by Zhan et al. [17] is adopted. ETR is used to calculate the percentage of improvement in algorithm performance caused by each percentage of additional time consumption, which is shown as:

$$\mathrm{ETR}(A, A_{KL}) = \frac{\mathrm{ERR}(A, A_{KL})}{\mathrm{TIR}(A, A_{KL})} \tag{10}$$

where A and A_{KL} indicate the original algorithm and its KL-based variant, respectively. ERR(A, A_{KL}) is the error reduction rate, which is calculated via

$$\mathrm{ERR}(A, A_{KL}) = \begin{cases} \dfrac{E(A) - E(A_{KL})}{E(A)}, & \text{if } E(A) > E(A_{KL}) \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

where $E(A)$ and $E(A_{KL})$ indicate the mean error obtained by algorithm A and its KL-based variant A_{KL}. TIR(A, A_{KL}) is the time cost increase rate, which is calculated via

TABLE V
RESULTS FOR NUMBER OF + / ≈ / − OBTAINED BY COMPARING KLDE AND KLPSO WITH $lr = 0.2$ AND OTHER lr SETTINGS AT $D = 30$ AND $ep = 10$

Algorithms	$lr = 0.2$ *vs* $lr = 0.1$	$lr = 0.2$ *vs* $lr = 0.3$
	+ / ≈ / −	+ / ≈ / −
KLDE	8 / 12 / 9	9 / 17 / 3
KLPSO	4 / 22 / 3	6 / 21 / 2
Algorithms	$lr = 0.2$ *vs* $lr = 0.4$	$lr = 0.2$ *vs* $lr = 0.5$
	+ / ≈ / −	+ / ≈ / −
KLDE	14 / 13 / 2	20 / 7 / 2
KLPSO	9 / 14 / 6	12 / 15 / 2

TABLE VI
RESULTS FOR NUMBER OF + / ≈ / − OBTAINED BY COMPARING KLDE AND KLPSO WITH $ep = 10$ AND OTHER ep SETTINGS AT $D = 30$ AND $lr = 0.2$

Algorithms	$ep = 10$ *vs* $ep = 1$	$ep = 10$ *vs* $ep = 5$	$ep = 10$ *vs* $ep = 20$
	+ / ≈ / −	+ / ≈ / −	+ / ≈ / −
KLDE	16 / 12 / 1	5 / 19 / 5	3 / 21 / 5
KLPSO	6 / 19 / 4	3 / 24 / 2	6 / 19 / 4

$$\mathrm{TIR}(A, A_{KL}) = \frac{T(A_{KL}) - T(A)}{T(A)} \tag{12}$$

where $T(A)$ and $T(A_{KL})$ indicate the cost of CPU time of A and A_{KL}. The value of ETR(A, A_{KL}) represents the percentage of error reduction divided by the percentage of additional time cost, and thus ETR can be used to assess whether the time consumption is worth it.

Herein, the results with respect to ETR are obtained in six settings of the time cost of each fitness evaluation (i.e., each fitness evaluation costs 1, 2, 4, 6, 8, and 10ms, respectively). The curve of the ETR obtained by DE and KLDE, JADE and KL-JADE, ADDE and KL-ADDE on $D = 50$ is shown in Fig. S5 (a) of the supplementary material, while the curve of the ETR obtained by PSO and KLPSO, TAPSO and KL-TAPSO, AWPSO and KL-AWPSO on $D = 50$ is shown in Fig. S5 (b). According to the curves, we can find the ETR of these DE-based algorithms is up to 2.0, while the ETR of these PSO-based algorithms is up to 1.5, when the time cost of each fitness evaluation is 10ms. In such a case, for every percent of extra time consumed, the KL framework is able to improve the performance of the DE-based algorithm by 2.0 percent and improve the performance of the PSO-based algorithm by 1.5 percent. In many real-world expensive optimization problems, the time cost of each fitness evaluation is very high, e.g., a single fitness evaluation of some problems can consume a few days [49]-[51], which is much higher than 10ms. Therefore, the additional time cost of the KL framework is worth it, especially in solving the expensive optimization problem.

Based on the results on runtime and solution accuracy, we can conclude both the advantages and disadvantages of the proposed KL framework. On the one hand, the advantages of the KL framework are that it can help the population jump out of the local optima and accelerate the convergence speed of the population to fast approach the global optimum. Combing the KL framework with the state-of-the-art EC algorithms can significantly enhance the accuracy of the finally obtained solutions. On the other hand, the disadvantage of the KL framework is that the training and utilizing processes of the KLM will consume additional time. Nevertheless, the additional time cost of the KL framework is worthy for the

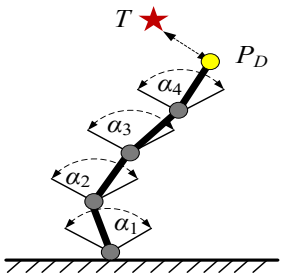


Fig. 7. An example of the planar kinematic arm control problem.

TABLE VII
RESULTS FOR MNF OF DE, HYDE-DF, PSO, AWPSO, AND THEIR KL-BASED VARIANTS ON LOW-DIMENSIONAL PROBLEMS

Dim	KLDE	DE	KL-HyDE-DF	HyDE-DF
D = 10	**1.80E-01**	2.74E-01 (≈)	**1.56E-01**	6.89E-01 (+)
D = 50	**1.99E-01**	5.20E-01 (+)	**2.90E-01**	7.09E-01 (+)
Number of + / ≈ / −		1 / 1 / 0	+ / ≈ / −	2 / 0 / 0
Dim	KLPSO	PSO	KL-AWPSO	AWPSO
D = 10	**2.80E-01**	4.97E-01 (+)	**2.63E-01**	5.13E-01 (+)
D = 50	**3.57E-01**	6.17E-01 (+)	**3.73E-01**	6.81E-01 (+)
Number of + / ≈ / −		2 / 0 / 0	+ / ≈ / −	2 / 0 / 0

TABLE VIII
RESULTS FOR MNF OF DE, HYDE-DF, PSO, AWPSO, AND THEIR KL-BASED VARIANTS ON HIGH-DIMENSIONAL PROBLEMS

Dim	KLDE	DE	KL-HyDE-DF	HyDE-DF
D = 100	**2.25E-01**	6.40E-01 (+)	**4.65E-01**	5.10E-01 (≈)
D = 500	**4.41E-01**	5.60E-01 (≈)	**4.68E-01**	7.37E-01 (+)
Number of + / ≈ / −		1 / 1 / 0	+ / ≈ / −	1 / 1 / 0
Dim	KLPSO	PSO	KL-AWPSO	AWPSO
D = 100	**4.74E-01**	7.91E-01 (+)	**5.22E-01**	8.29E-01 (+)
D = 500	**6.22E-01**	7.35E-01 (+)	**6.40E-01**	6.67E-01 (≈)
Number of + / ≈ / −		2 / 0 / 0	+ / ≈ / −	1 / 1 / 0

accuracy improvement, especially in solving complex and expensive optimization problems.

H. Comparison on Real-World Application

In this section, the performance of the KL framework is evaluated in a real-world optimization problem, which is called the planar kinematic arm control problem [52], [53]. Fig. 7 illustrates an example of the planar kinematic arm control problem. The planar kinematic arm control problem contains a kinematic arm with several links and joints, and the objective is to control the angle of each joint to make the tip of the arm P_D as close as possible to the target T. Without loss of generosity, the objective function $f_p(.)$ of the planar kinematic arm control problem with d dimensions (i.e., with d joints) is shown as:

$$f_p\left(\alpha_1,\alpha_2,...,\alpha_d,[L,\alpha_{\max}]\right)=\left\|P_D-T\right\| \tag{13}$$

where α_1, α_2, …, α_d denote the angle of the d joints, L denotes the total length of the links, $\alpha_{\max}$ denotes the sum of the maximum angle of all the joints. The values of the parameters are set as the same as those in [53]. In the experimental results, the mean normalized fitness (MNF) is utilized as the metric to more clearly evaluate the performance of each algorithm, which is calculated via

$$MNF=\frac{f-\min(F)}{\max(F)-\min(F)} \tag{14}$$

where f denotes the obtained fitness value, F denotes all the fitness values obtained by both the KL-based algorithm and the original algorithm on the current problem.

First, the performance of four compared algorithms (i.e., DE, HyDE-DF, PSO, and AWPSO) and their KL-based variants are evaluated on two low-dimensional planar kinematic arm control problems (i.e., D = 10 and 50). The experimental results with respect to MNF obtained by the compared algorithms and the KL-based algorithms on D = 10 and 50 are shown in Table VII. According to the results, we can find the KL-based algorithms generally outperforms the original algorithms. Therefore, we can conclude that the KL framework achieves promising performance on the low-dimensional planar kinematic arm control problems.

Second, the performance of the compared algorithms and their KL-based variants are evaluated on two high-dimensional planar kinematic arm control problems (i.e., D = 100 and 500). The experimental results with respect to MNF obtained by the compared algorithms and the KL-based algorithms on D = 100 and 500 are shown in Table VIII. On both D = 100 and 500, the performance of the KL-based algorithms is generally better than that of the original algorithms. Specifically, on D = 100, the performance of KLDE, KLPSO, and KL-AWPSO is significantly greater than HyDE-DF, PSO, and AWPSO, respectively. On D = 500, the performance of KL-HyDE-DF and KLPSO is significantly superior to HyDE-DF and PSO, respectively. Therefore, we can also conclude that the achieves encouraging performance on the high-dimensional planar kinematic arm control problems.

V. CONCLUSION

In this paper, an innovative KL framework for EC algorithms has been proposed. The KL framework learns the knowledge based on successful experiences and guides the evolution according to the learned knowledge and the positions of the individuals. In the KL framework, an FNN-based KLM is created to learn the successful experience and get knowledge about the relationship between the position of the individual and the optimal direction. During the evolution, each individual can acquire a suitable evolutionary direction from the KLM according to its current position and the knowledge of KLM.

To evaluate the functionality of the KL framework, the traditional DE and PSO, and several DE-based and PSO-based state-of-the-art algorithms are compared with that of their KL-based variants. In addition, the KL framework is also compared with two recently proposed historical data utilizing methods. In these experiments conducted on both the benchmark functions and the real-world optimization problems, the KL framework achieves a more promising performance than the compared algorithms.

Although the concept and core idea of the proposed KL framework is simple and easy to understand, it is very effective and efficient. In future work, we hope to extend the idea of KL to more research aspects of EC, such as large-scale optimization problems [54]-[56], multimodal optimization problems [57]-[59], multi-/many-objective optimization [60]-[62], and multi-task optimization [63]-[65]. Additionally, we will combine the KL framework with other kinds of EC algorithms, such as genetic algorithms [66] and memetic

algorithms [67], [68].

REFERENCES

[1] Z. H. Zhan, L. Shi, K. C. Tan, and J. Zhang, "A survey on evolutionary computation for complex continuous optimization," *Artif. Intell. Rev.*, vol. 55, no. 1, pp. 59-110, Jan. 2022.

[2] K. Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, and Q. Pan, "A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems," *IEEE/CAA J. Autom. Sin.*, vol. 6, no. 4, pp. 904-916, July 2019.

[3] Z. G. Chen, Z. H. Zhan, S. Kwong, and J. Zhang, "Evolutionary computation for intelligent transportation in smart cities: A survey," *IEEE Comput. Intell. Mag*., vol. 17, no. 2, pp. 83-102, May, 2022.

[4] Z. H. Zhan *et al.*, "Matrix-based evolutionary computation," *IEEE Trans. Emerging Topics Comp. Intell*., vol. 6, no. 2, pp. 315-328, Apr. 2022.

[5] R. Storn and K. Price, "Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces," *Int. Comput. Sci. Inst*., Berkeley, CA, USA, Rep. TR–95–012, 1995.

[6] A. Karbassi Yazdi, M. A. Kaviani, T. Hanne, and A. Ramos, "A binary differential evolution algorithm for airline revenue management: A case study," *Soft Comput.*, vol. 24, no. 18, pp. 14221–14234, Sep. 2020.

[7] C. Bu, W. Luo, T. Zhu, R. Yi, and B. Yang, "Species and memory enhanced differential evolution for optimal power flow under double-sided uncertainties," *IEEE Trans. Sust. Comp.*, vol. 5, no. 3, pp. 403-415, Sept. 2020.

[8] J. H. Holland, "Genetic algorithm," *Scientific American*, vol. 267, no. 1, pp. 66-83, Jul. 1992.

[9] S. C. Liu, Z. G. Chen, Z. H. Zhan, S. W. Jeon, S. Kwong, and J. Zhang, "Many-objective job shop scheduling: A multiple populations for multiple objectives-based genetic algorithm approach," *IEEE Trans. Cybern.*, vol. 53, no. 3, pp. 1460-1474, Mar. 2023.

[10] Q. T. Yang, Z. H. Zhan, S. Kwong, and J. Zhang, "Multiple populations for multiple objectives framework with bias sorting for many-objective optimization," *IEEE Trans. Evol. Comput*., to be published, doi: 10.1109/TEVC.2022.3212058.

[11] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, 1995, pp. 1942–1948.

[12] W. Luo, J. Sun, C. Bu, and H. Liang, "Species-based particle swarm optimizer enhanced by memory for dynamic optimization," *Appl. Soft Comput.*, vol. 47, pp. 130–140, Oct. 2016.

[13] J. R. Jian, Z. G. Chen, Z. H. Zhan, and J. Zhang, "Region encoding helps evolutionary computation evolve faster: A new solution encoding scheme in particle swarm for large-scale optimization," *IEEE Trans. Evol. Comput.*, vol. 25, no. 4, pp. 779–793, Aug. 2021.

[14] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53-66, April 1997.

[15] X. Zhang, Z. H. Zhan, W. Fang, P. Qian, and J. Zhang, "Multi-population ant colony system with knowledge-based local searches for multiobjective supply chain configuration," *IEEE Trans. Evol. Comput.*, vol. 26, no. 3, pp. 512-526, Jun. 2022.

[16] L. Shi, Z. H. Zhan, D. Liang, and J. Zhang, "Memory-based ant colony system approach for multi-source data associated dynamic electric vehicle dispatch optimization," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 10, pp. 17491-17505, Oct. 2022.

[17] Z. H. Zhan, J. Y. Li, S. Kwong, and J. Zhang, "Learning-aided evolution for optimization," *IEEE Trans. Evol. Comput.*, to be published, doi: 10.1109/TEVC.2022.3232776.

[18] K. C. Tan, L. Feng, and M. Jiang, "Evolutionary transfer optimization - A new frontier in evolutionary computation research," *IEEE Comput. Intell. Mag.*, vol. 16, no. 1, pp. 22–33, Feb. 2021.

[19] Y. Jiang, Z. H. Zhan, K. C. Tan, and J. Zhang, "Block-level knowledge transfer for evolutionary multi-task optimization," *IEEE Trans. Cybern.*, to be published, doi: 10.1109/TCYB.2023.3273625.

[20] H. Han, X. Bai, Y. Hou, and J. Qiao, "Multi-task particle swarm optimization with dynamic on-demand allocation," *IEEE Trans. Evol. Comput.*, to be published, doi: 10.1109/TEVC.2022.3187512.

[21] X. Zhang and S. Y. Yuen, "A directional mutation operator for differential evolution algorithms," *Appl. Soft Comput.*, vol. 30, pp. 529–548, May. 2015.

[22] A. Ghosh, S. Das, A. K. Das, and L. Gao, "Reusing the past difference vectors in differential evolution—A simple but significant improvement," *IEEE Trans. Cybern.*, vol. 50, no. 11, pp. 4821–4834, Nov. 2020.

[23] A. Gupta, Y.-S. Ong, and L. Feng, "Multifactorial evolution: Toward evolutionary multitasking," *IEEE Trans. Evol. Comput.*, vol. 20, no. 3, pp. 343–357, Jun. 2016.

[24] L. Zhou *et al.*, "Toward adaptive knowledge transfer in multifactorial evolutionary computation," *IEEE Trans. Cybern.*, vol. 51, no. 5, pp. 2563–2576, May 2021.

[25] L. Feng *et al.*, "Evolutionary multitasking via explicit autoencoding," *IEEE Trans. Cybern.*, vol. 49, no. 9, pp. 3457–3470, Sep. 2019.

[26] H. Xu, A. K. Qin, and S. Xia, "Evolutionary multi-task optimization with adaptive knowledge transfer," *IEEE Trans. Evol. Comput*., to be published, doi: 10.1109/TEVC.2021.3107435.

[27] J. Yin, A. Zhu, Z. Zhu, Y. Yu, and X. Ma, "Multifactorial evolutionary algorithm enhanced with cross-task search direction," in *Proc. IEEE Congr. Evol. Comput.*, 2019, pp. 2244–2251.

[28] J. Zhang and A. C. Sanderson, "JADE: Adaptive differential evolution with optional external archive," *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 945–958, 2009.

[29] R. Tanabe and A. Fukunaga, "Success-history based parameter adaptation for differential evolution," in *Proc. IEEE Congr. Evol. Comput.*, 2013, pp. 71–78.

[30] R. Tanabe and A. S. Fukunaga, "Improving the search performance of SHADE using linear population size reduction," in *Proc. IEEE Congr. Evol. Comput.*, 2014, pp. 1658–1665.

[31] J. Brest, M. S. Maučec, and B. Bošković, "Single objective real-parameter optimization: Algorithm jSO," in *Proc. IEEE Congr. Evol. Comput.*, 2017, pp. 1311–1318.

[32] Z. H. Zhan, Z. J. Wang, H. Jin, and J. Zhang, "Adaptive distributed differential evolution," *IEEE Trans. Cybern.*, vol. 50, no. 11, pp. 4633–4647, Nov. 2020.

[33] J. J. Liang, L. Guo, R. Liu, and B. Y. Qu, "A self-adaptive dynamic particle swarm optimizer," in *Proc. IEEE Congr. Evol. Comput.*, 2015, pp. 3206–3213.

[34] Z. H. Zhan and J. Zhang, "Self-adaptive differential evolution based on PSO learning strategy," in *Proc. Genet. Evol. Comput. Conf.*, 2010, pp. 39–46.

[35] B. Xue, A. K. Qin, and M. Zhang, "An archive-based particle swarm optimisation for feature selection in classification," in *Proc. IEEE Congr. Evol. Comput.*, 2014, pp. 3119-3126.

[36] X. Xia *et al.*, "Triple archives particle swarm optimization," *IEEE Trans. Cybern*., vol. 50, no. 12, pp. 4862–4875, Dec. 2020.

[37] T. Zhu, W. Luo, C. Bu, and H. Ning, "Making use of observable parameters in evolutionary dynamic optimization," *Inf. Sci.*, vol. 512, pp. 708–725, Feb. 2020.

[38] X. Zhang and X. Zhang, "Improving differential evolution by differential vector archive and hybrid repair method for global optimization," *Soft Comput.*, vol. 21, no. 23, pp. 7107–7116, 2017.

[39] Z. H. Zhan, J. Y. Li, and J. Zhang, "Evolutionary deep learning: A survey," *Neurocomputing*, vol. 483, pp. 42-58, April 2022.

[40] C. Wang, C. Xu, X. Yao, and D. Tao, "Evolutionary generative adversarial networks," *IEEE Trans. Evol. Comput.*, vol. 23, no. 6, pp. 921–934, Dec. 2019.

[41] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.

[42] I. Goodfellow, A. Courville, and Y. Bengio, *Deep Learning*, Cambridge: MIT press, 2016.

[43] F. Lezama, J. Soares, R. Faia, and Z. Vale, "Hybrid-adaptive differential evolution with decay function (HyDE-DF) applied to the 100-digit challenge competition on single objective numerical optimization," in *Proc. Genet. Evol. Comput. Conf.*, 2019, pp. 7–8.

[44] A. Ratnaweera, S. K. Halgamuge, and H. C. Watson, "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 240–255, Jun. 2004.

[45] W. Liu, Z. Wang, Y. Yuan, N. Zeng, K. Hone, and X. Liu, "A novel sigmoid-function-based adaptive weighted particle swarm optimizer," *IEEE Trans. Cybern.*, vol. 51, no. 2, pp. 1085-1093, Feb. 2021.

[46] N. H. Awad, M. Z. Ali, J. J. Liang, B. Y. Qu, and P. N. Suganthan, "Problem definitions and evaluation criteria for the CEC 2017 special session and competition on single objective bound constrained real-parameter numerical optimization," *Nanyang Technol. Univ., Singapore, Rep*, 2016.

[47] B. Li and W. S. Jiang, "Heuristics genetic algorithm using 80/20 rule," in *Proc IEEE Int. Conf. Ind. Technol.*, 1996, pp. 436-438.

[48] M. Iqbal and M. Rizwan, "Application of 80/20 rule in software

engineering waterfall model," in *Proc. Int. Conf. Inf. Commun. Technol.*, 2009, pp. 223-228.

[49] J. Y. Li, Z. H. Zhan, and J. Zhang, "Evolutionary computation for expensive optimization: A survey," *Mach. Intell. Res*., vol. 19, no. 1, pp. 3– 23, 2022.

[50] S. H. Wu, Z. H. Zhan, and J. Zhang, "SAFE: Scale-adaptive fitness evaluation method for expensive optimization problems," *IEEE Trans. Evol. Comput.*, vol. 25, no. 3, pp. 479-491, Jun. 2021.

[51] J. Y. Li, Z. H. Zhan, C. Wang, H. Jin, and J. Zhang, "Boosting data-driven evolutionary algorithm with localized data generation," *IEEE Trans. Evol. Comput.*, vol. 24, no. 5, pp. 923-937, Oct. 2020.

[52] H. Xu, A. K. Qin, and S. Xia, "Evolutionary multitask optimization with adaptive knowledge transfer," *IEEE Trans. Evol. Comput.*, vol. 26, no. 2, pp. 290-303, Apr. 2022.

[53] Y. Jiang, Z. H. Zhan, K. C. Tan, and J. Zhang, "A bi-objective knowledge transfer framework for evolutionary many-task optimization," *IEEE Trans. Evol. Comput.*, to be published, doi: 10.1109/TEVC.2022.3210783.

[54] Z. J. Wang, Z. H. Zhan, S. Kwong, H. Jin, and J. Zhang, "Adaptive granularity learning distributed particle swarm optimization for large-scale optimization," *IEEE Trans. Cybern.*, vol. 51, no. 3, pp. 1175-1188, Mar. 2021.

[55] Z. J. Wang, J. R. Jian, Z. H. Zhan, Y. Li, S. Kwong, and J. Zhang, "Gene targeting differential evolution: A simple and efficient method for large scale optimization," *IEEE Trans. Evol. Comput.*, to be published, doi: 10.1109/TEVC.2022.3185665.

[56] J. Y. Li, Z. H. Zhan, K. C. Tan, and J. Zhang, "Dual differential grouping: A more general decomposition method for large-scale optimization," *IEEE Trans. Cybern.*, to be published, doi:10.1109/TCYB.2022.3158391.

[57] Y. Jiang, Z. H. Zhan, K. C. Tan, and J. Zhang, "Optimizing niche center for multimodal optimization problems," *IEEE Trans. Cybern.*, to be published, vol. 53, no. 4, pp. 2544-2557, Apr. 2023.

[58] Z. G. Chen, Z. H. Zhan, H. Wang, and J. Zhang, "Distributed individuals for multiple peaks: A novel differential evolution for multimodal optimization problems," *IEEE Trans. Evol. Comput.*, vol. 24, no. 4, pp. 708-719, Aug. 2020.

[59] 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.

[60] S. C. Liu, Z. H. Zhan, K. C. Tan, and J. Zhang, "A multi-objective framework for many-objective optimization," *IEEE Trans. Cybern.*, vol. 52, no. 12, pp. 13654-13668, Dec. 2022.

[61] 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.*, vol. 23, no. 4, pp. 587-602, Aug. 2019.

[62] J. Y. Li *et al.*, "A multi-population multi-objective ant colony system considering travel and prevention costs for vehicle routing in COVID-19-like epidemics," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 12, pp. 25062-25076, Dec. 2022.

[63] S. H. Wu, Z. H. Zhan, K. C. Tan, and J. Zhang, "Orthogonal transfer for multitask optimization," *IEEE Trans. Evol. Comput.*, vol. 27, no. 1, pp. 185-200, Feb. 2023.

[64] J. Y. Li, Z. H. Zhan, K. C. Tan, and J. Zhang, "A meta-knowledge transfer-based differential evolution for multitask optimization," *IEEE Trans. Evol. Comput.*, vol. 26, no. 4, pp. 719-734, Aug. 2022.

[65] S. H. Wu, Z. H. Zhan, K. C. Tan, and J. Zhang, "Transferable adaptive differential evolution for many-task optimization," *IEEE Trans. Cybern.*, to be published, doi: 10.1109/TCYB.2023.3234969. Dec. 2022.

[66] J. E. Lehner, R. Dornberger, R. Simić, and T. Hanne, "Optimization of multi-robot sumo fight simulation by a genetic algorithm to identify dominant robot capabilities," in *Proc. IEEE Congr. Evol. Comput.*, 2019, pp. 490-496.

[67] J. Luo, Y. Yang, Q. Liu, X. Li, M. Chen, and K. Gao, "A new hybrid memetic multi-objective optimization algorithm for multi-objective optimization," *Inf. Sci.*, vol. 448–449, pp. 164–186, Jun. 2018.

[68] V. A. Shim, K. C. Tan, and H. Tang, "Adaptive memetic computing for evolutionary multiobjective optimization," *IEEE Trans. Cybern.*, vol. 45, no. 4, pp. 610-621, Apr. 2015.

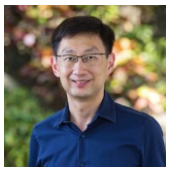
Yi Jiang (Student Member, IEEE) received the B.S. degree in computer science and technology from South China University of Technology, Guangzhou, China, in 2020, where he is currently pursuing the Ph.D. degree in computer science and technology with the School of Computer Science and Engineering.

His research interests mainly include computational intelligence, evolutionary computation, machine learning, and their applications in real-world problems.

Zhi-Hui Zhan (Senior Member, IEEE) received the Bachelor's degree and the Ph. D. degree in Computer Science from the Sun Yat-Sen University, Guangzhou China, in 2007 and 2013, respectively.

He is currently the Changjiang Scholar Young Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. His current research interests include evolutionary computation, swarm intelligence, and their applications in real-world problems and in environments of cloud computing and big data.

Dr. Zhan was a recipient of the IEEE Computational Intelligence Society (CIS) Outstanding Early Career Award in 2021, the Outstanding Youth Science Foundation from 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 one of the World's Top 2% Scientists for both Career-Long Impact and Year Impact in Artificial Intelligence and one of the Highly Cited Chinese Researchers in Computer Science. He is currently the Chair of Membership Development Committee in IEEE Guangzhou Section and the Vice-Chair of IEEE CIS Guangzhou Chapter. He is currently an Associate Editor of the *IEEE Transactions on Evolutionary Computation*, the *Neurocomputing*, the *Memetic Computing*, and the *Machine Intelligence Research*.



Kay Chen Tan (Fellow, IEEE) received the B.Eng. degree (First Class Hons.) and the Ph.D. degree from the University of Glasgow, U.K., in 1994 and 1997, respectively.

He is currently a Chair Professor (Computational Intelligence) of the Department of Computing, The Hong Kong Polytechnic University. He has published over 300 refereed articles and seven books.

Prof. Tan is currently the Vice-President (Publications) of IEEE Computational Intelligence Society, USA. He served as the Editor-in-Chief of the *IEEE Computational Intelligence Magazine* from 2010 to 2013 and the *IEEE Transactions on Evolutionary Computation* from 2015 to 2020. He currently serves as the Chief Co-Editor of Springer Book Series on Machine Learning: Foundations, Methodologies, and Applications, and as an Editorial Board Member for more than ten journals.

Jun Zhang (Fellow, IEEE) obtained his PhD degree in Electrical Engineering from the City University of Hong Kong in 2002.

He is a BP professor at Hanyang University, ERICA, South Korea, and an honorary professor at Key Laboratory of Intelligent Education Technology and Application of Zhejiang Province, Zhejiang Normal University, China. Prof. Zhang's research contributions span over 300 peer-reviewed publications, of which more than 180 appear in IEEE Transactions. His research interests include Computational Intelligence, cloud computing, Big data mining, and Power Electronic Circuits.

Professor Zhang was a recipient of the China National Funds for Distinguished Young Scientists from the National Natural Science Foundation of China in 2011 and was appointed as a Cheung Kong Chair Professor in 2013 by the Ministry of Education, China. Presently, Prof. Zhang serves as an associate editor for both the *IEEE Transactions on Artificial Intelligence* and the *IEEE Transactions on Cybernetics*.