Matrix-Based Evolutionary Computation

Zhi-Hui Zhan, *Senior Member, IEEE*, Jun Zhang, *Fellow, IEEE*, Ying Lin, *Member, IEEE*, Jian-Yu Li, *Student Member, IEEE*, Ting Huang, *Student Member, IEEE*, Xiao-Qi Guo, *Student Member, IEEE*, Feng-Feng Wei, *Student Member, IEEE*, Sam Kwong, *Fellow, IEEE*, Xin-Yi Zhang, *Student Member, IEEE*, and Rui You, *Student Member, IEEE*

***Abstract*—Computational intelligence (CI), including artificial neural network, fuzzy logic, and evolutionary computation (EC), has rapidly developed nowadays. Especially, EC is a kind of algorithm for knowledge creation and problem solving, playing a significant role in CI and artificial intelligence (AI). However, traditional EC algorithms have faced great challenge of heavy computational burden and long running time in large-scale (e.g., with many variables) problems. How to efficiently extend EC algorithms to solve complex problems has become one of the most significant research topics in CI and AI communities. To this aim, this paper proposes a matrix-based EC (MEC) framework to extend traditional EC algorithms for efficiently solving large-scale or super large-scale optimization problems. The proposed framework is an entirely new perspective on EC algorithm, from the solution representation to the evolutionary operators. In this framework, the whole population (containing a set of individuals) is defined as a matrix, where a row stands for an individual and a column stands for a dimension (decision variable). This way, the parallel computing functionalities of matrix can be directly and easily carried out on the high performance computing resources to accelerate the computational speed of evolutionary operators. This paper gives two typical examples of MEC algorithms, named matrix-based genetic algorithm and matrix-based particle swarm optimization. Their matrix-based solution representations are presented and the evolutionary operators based on the matrix are described. Moreover, the time complexity is analyzed and the experiments are conducted to show that these MEC algorithms are efficient in reducing the computational time on large scale of decision variables. The MEC is a promising way to extend EC to complex optimization problems in big data environment, leading to a new research direction in CI and AI.**

***Index Terms*—Evolutionary computation (EC), matrix-based evolutionary computation (MEC), genetic algorithm (GA), particle swarm optimization (PSO).**

I. Introduction

COMPUTATIONAL intelligence (CI) is a kind of biologically and linguistically motivated computational paradigm that mainly contains three branches as artificial neural network (ANN), logic inference, and evolutionary computation (EC), which has lots of overlaps with artificial intelligence (AI) [1]. In fact, the IEEE CI Society (CIS) is directly related to AI and covers the main researches of AI. The three big branches of CI (i.e., logic, ANN, and EC) are mainly corresponding to the three branches of AI to approach the human intelligence, i.e., *Symbolisms*, *Connectionism*, and *Evolutionism*. Nowadays CI and AI have extremely fast developed, including the great success of deep learning (DL) and deep neural network (DNN) in a large number of real-world applications, such as speech recognition [2], chemical syntheses planning [3], and smart city [4]. The success of DL and DNN are strongly related to the big data (BD) resources and the high performance computing (HPC) resources that substantially extend traditional ANN to DNN and the applications [5]. Fig. 1 shows the relationship among the algorithm, data, and computational power to illustrate the development of CI. Specially, the BD resources make it possible for DNN to be trained with good quality, while the HPC resources like cloud computing, supercomputing, and graphic process unit (GPU) make it possible for DNN to be executed in an acceptable time. Therefore, the integration of *A*lgorithms (like ANN), *B*ig data resources, and *C*omputing resources (like HPC) results in the A-B-C to make new CI techniques (like DNN) greatly successful in complex problems in real-world applications like industry, government, environment, economy, finance, medicine, education, management, and ecology.

However, the DL and DNN are only parts but not all of the CI and AI. As shown in Fig. 1, the three branches in CI achieve the abilities of *Knowledge Learning*, *Knowledge Reasoning*, and *Knowledge Creation* by approaches like ANN, logic inference, and EC, respectively. During the history of CI, these three branches on the one hand develop their own different techniques and on the other hand also interact with each other. As shown in Fig. 1, with the help of BD and HPC resources/techniques, not only the ANN has been promoted to DL and DNN with greater

Manuscript received March 17, 2020; revised September 29, 2020 and December 2, 2020; accepted December 18, 2020. Date of publication January 21, 2021; date of current version March 25, 2022. This work was supported in part by the National Natural Science Foundations of China (NSFC) under Grants 61873097, 61822602, and 61772207, in part by the National Key Research and Development Program of China under Grant 2019YFB2102102, in part by the Key-Area Research and Development of Guangdong Province under Grant 2020B010166002, in part by the Guangdong Natural Science Foundation Research Team under Grant 2018B030312003, in part by the Guangdong-Hong Kong Joint Innovation Platform under Grant 2018B050502006, and in part by the Hong Kong GRF-RGC General Research Fund 9042816 (CityU 11209819) *(Corresponding author: Jun Zhang.)*

Zhi-Hui Zhan, Jian-Yu Li, Ting Huang, Xiao-Qi Guo, Feng-Feng Wei, Xin-Yi Zhang, and Rui You are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China.

Jun Zhang is with the Hanyang University, Ansan 15588, Korea and also with the Victoria University, Melbourne, VIC 8001, Australia (e-mail: junzhang@ieee.org).

Ying Lin is with the Department of Psychology, Sun Yat-sen University, Guangzhou 510006, China.

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

This article has supplementary downloadable material available at https://doi.org/10.1109/TETCI.2020.3047410, provided by the authors.

Digital Object Identifier 10.1109/TETCI.2020.3047410

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

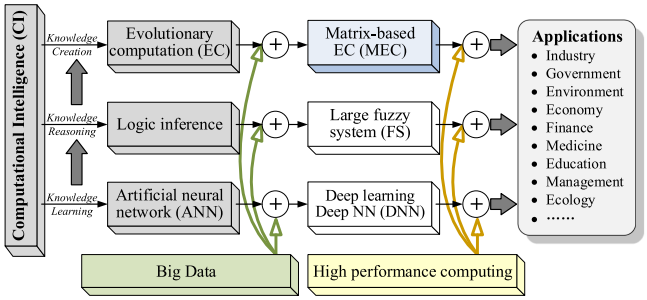

Fig. 1. Development of computational intelligence.

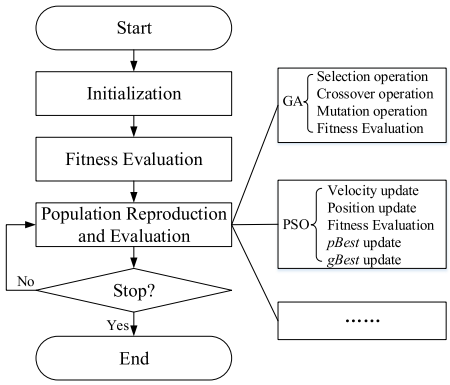

Fig. 2. Generic framework of evolutionary computation algorithms.

knowledge learning ability that as reported in *Nature* [6] and *Science* [7], but also the researches in the knowledge reasoning have turned from simple logic inference to complex fuzzy system (FS) to deal with complex problems. Moreover, being the approaches for knowledge creation, which act as more significant intelligent role in CI and AI, the EC algorithms have interacted with the approaches in knowledge learning and knowledge reasoning for a long time, resulting in evolutionary DNN [8], evolutionary DL [9], evolutionary FS [10], fuzzy-based EC [11], and machine learning enhanced EC [12]. More significantly, it is noted that knowledge reasoning and knowledge learning mainly lead to lower-level of AI, e.g., if the knowledge has been mastered by human being, one can use ANN or DL to enable machine to learn this knowledge so that the machine can replace the human being to do something. However, EC is a nature-inspired approach for knowledge creation and problem-solving, which can help human to solve highly complicated and NP-hard problems. Therefore, the EC can obtain something that human does not know, for example, finding the most optimized solutions for the real-world problems, such as large-scale supply chain network design [13], mineral processing [14], airline crew rostering problem [15], and virtual machine placement [16] and workflow management [17] in cloud, which can be regarded as higher-level of AI.

Therefore, in the future development of CI and AI, more attention would be paid to EC algorithms because they are a kind of general problem-solving tools to achieve higher-level AI. Moreover, the EC algorithms are independent on exact problem model, making them one of the promising ways for problem-solving and knowledge creation when there are no exact approaches/models for the complex problems in nowadays' complex environments. Therefore, this paper focuses on the EC algorithms.

Traditional EC algorithms share a common framework like Fig. 2, including initialization, fitness evaluation, and new population reproduction, to search for the global optimum generation by generation. Therefore, ECs are population-based and iteration-based algorithms which have generally better global search ability than other single-solution search algorithms. The EC family includes evolutionary algorithms (EA) [18], e.g., genetic algorithm (GA) [19], differential evolution (DE) [20], and estimation of distribution algorithm (EDA) [21], and swarm intelligence (SI) algorithms [22] like particle swarm optimization (PSO) [23] and ant colony optimization (ACO) [24]. Under the generic framework of Fig. 2, different kinds of EC algorithms can have their own population reproduction strategies. For example, GA is a typical EA that reproduces the new population by selection, crossover, and mutation operations, while PSO is a typical SI that reproduces the new population by learning from personal and global guidance information, including the velocity and position update, and the personally best and globally best experiences update. Therefore, this paper mainly focuses on these two EC algorithms.

Due to the fact that EC algorithms are less sensitive to the mathematical model of problems, they have been successfully applied to many kinds of science and engineering optimization problems that are difficult to be solved by traditional mathematical methods, such as in computer network [25], electromagnetic engineering [26], Blockchain applications [27], mobile computing [28], and information security [29], which have speeded up the development of the EC community in recent years. These successes are significant in real-world applications for small- or medium-scale problems. However, as the practical problems become increasingly complex nowadays, the number of decision variables can reach thousands or hundreds of thousand, resulting in large-scale or super large-scale optimization problems. Moreover, the population size can be set very large to increase the diversity when dealing with complex problems [30]. In this sense, being a kind of population-based and iterative-based algorithms, the EC algorithms may result in very heavy computational burdens due to the large population or the large-scale decision variables, or both. This is an obstacle that prevents EC algorithms from being widely used nowadays. Therefore, like the large-scale FS and DNN that are based on BD and HPC resources/techniques to efficiently solve complex problems [31], Fig. 1 shows the researches in designing efficient EC algorithms with the help of BD and HPC for solving complex problems have also become a pioneer approach.

In order to relieve the computational burden when dealing with complex optimization problems, a promising way is to design parallel or distributed EC (DEC) algorithms [32]. This has become an emerging topic and has aroused great attention from the EC researchers in recent years, like the distributed GA [33], distributed DE [34], distributed PSO [35], and distributed memetic algorithm [36]. The DEC always uses distributed computational resources to execute the multiple populations or all the individuals in parallel, which can reduce the running time

[37]. However, designing an effective DEC algorithm is ad hoc based on the computational resources and is relied on the hardware. Moreover, communication burden has to be considered. This is difficult for practical applications. Therefore, a more general way to extend the traditional EC framework is in great need. Also, the existing DEC algorithms almost only consider how to distribute the multiple populations or the individuals to distributed computational resources [38]. Recently, a novel generation-level parallelism PSO is proposed to enable the DEC being executed in parallel in *generation level* [39]. However, most existing DEC algorithms do not consider the potential parallelism on the decision variables. That is, the DEC are parallel in *population level* or *individual level*, or even in *generation level*, but not in *dimension level*. Therefore, more generic and efficient DEC algorithms are in great need to fully utilize the HPC resources like cloud computing, supercomputing, and GPU.

In fact, the parallelism on dimension level is a significant research topic in EC community but has not been studied intensively. This may be due to the fact that it is not easy for implementation. Refer to Fig. 2 again, the evolutionary operators such as the crossover and mutation in GA or the velocity and position update in PSO are executed on the decision variables dimension by dimension. However, one may also further consider whether the evolutionary operators can be executed by manipulating all the dimensions simultaneously. Someone may propose to also use multiple distributed computational resources to parallel execute these dimensions. However, under the traditional solution representation, this way is difficult if not impossible for implementation. For example, although one can separate all dimensions of the problem and deal with the variables simultaneously when updating the velocity/position of PSO, sophisticated programming skills are required. Moreover, when executing the operators in GA like selection and crossover, there are interactions among the variables vector, making it not easy or impossible to separate and distribute the dimensions to different resources for parallel computing. Therefore, a new approach is needed if researchers want to have such a kind of parallelism on dimension level.

This paper focuses on this particular issue and proposes a matrix-based EC (MEC) algorithm to efficiently extend traditional EC to faster solve large-scale or super large-scale optimization problems. Although there exists matrix-based implementation of DE [40], it uses the matrix scheme to enhance the search ability but not to reduce the running time. Differently, the MEC proposed in this paper focuses on the running time, which is an entirely new perspective on EC algorithm, from the solution representation to the evolutionary operators. Therefore, the MEC is a new emerging and promising topic of EC and CI/AI community. It is common that in traditional EC, an individual is represented by a vector and the population is formed by a set of individuals (vectors). Differently, the MEC does not represent an individual alone, but define the whole population (contains a set of individuals) as a matrix, where a row stands for one individual and a column stands for one dimension (decision variable). Matrix is easy for parallel computing. If the population

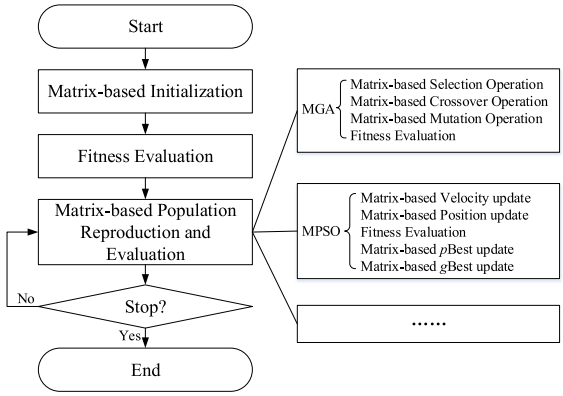

Fig. 3. Generic framework of matrix-based evolutionary computation.

of EC algorithm is represented by the matrix, MEC can use the parallel computing functionalities of matrix to accelerate the computational speed directly and easily. That is, the matrix operations [41] such as addition (+), subtraction (–), multiplication (×), find maximization or minimization, and other linear operations [42] have been widely studied in the parallel and distributed computing community [43]. Consequently, the evolutionary operators can be directly carried out in parallel due to the functionality of matrix. More importantly, researchers have developed many ripe parallel routines for matrix operations on HPC resources like GPU, cloud computing platform, and supercomputing platform [44]. This allows us to deploy the MEC to these HPC platforms naturally. In this way, the MEC can relive the heavy computational burden of large population size and large scale of decision variables. Based on the above advantages, the MEC can be a promising way to extend EC algorithms to complex optimization problems in a very complex big data environment. Similar to Fig. 2, the generic framework of MEC algorithms is illustrated as Fig. 3. Moreover, the MEC algorithms can be benefitted from the following three aspects:

Firstly, MEC is suitable for solving complex problems with any size of scale. The dimensions (variables) do not need to be decomposed and can be processed simultaneously because the matrix can be naturally distributed on sufficient HPC computational resources. Therefore, the computational time of MEC is not limited to the problem size.

Secondly, MEC can be beneficial from the bonuses of HPC. It has no need to consider the manipulation of the distributed resources to deal with the fussy issues like communication and transformation. The MEC can be directly deployed on HPC platforms like supercomputing and cloud computing. Therefore, the computational time of MEC can be further promoted with the development of supercomputing.

Thirdly, MEC is actually a true parallel and distributed algorithm. The MEC is not only parallel on population level and individual level, but also on dimension level. Therefore, the MEC is more suitable for population-based algorithm to solve large-scale or super large-scale optimization problems in shorter computational time.

This "EC-to-MEC" would be even more significant than that of "ANN-to-DNN" because EC algorithms are more suitable for optimization, knowledge creation, and problem solving, so that the MEC is a promising way to solve complex optimization problems in big data environment. The MEC will lead a new development in EC to achieve higher-level CI and AI.

The rest of the paper is organized as follows. Section II introduces the basic knowledge of matrix and some common matrix-based representation or operators in MEC. Section III and Section IV describe the MEC based on the two typical EC algorithms, one is the matrix-based GA (MGA) and the other is the matrix-based PSO (MPSO), respectively, including how to implement the evolutionary operators according to the new matrix-based representation. The time complexity of MEC (e.g., MGA and MPSO) is analyzed in Section V and the experiments are conducted in Section VI to show the efficiency of MEC in reducing computational burden. Finally, conclusions and future works are given in Section VII.

II. Basic Knowledge

A. Representations and Notations

Assume that there is a population with N individuals to solve a D-dimensional problem (i.e., with D variables). In traditional EC, an individual is represented by a vector as:

$$X_i = (x_{i1}, x_{i2}, \ldots, x_{iD}) \tag{1}$$

where $1 \leq i \leq N$ is the index of the individual. However, in MEC, the population is directly represented by a $N \times D$ matrix $\boldsymbol{X} = (x_{ij})_{N \times D}$ as:

$$\boldsymbol{X} = \begin{pmatrix} x_{11} & \ldots & x_{1D} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{ND} \end{pmatrix} \tag{2}$$

where a row represents an individual and a column represents a dimension (variable).

Accommodated with the matrix-based representation, lower bound and upper bound of the variables are represented by two $1 \times D$ matrices $\boldsymbol{L}$ and $\boldsymbol{U}$, respectively, while the fitness values of all the individuals are recorded by an $N \times 1$ matrix ***Fit***. Moreover, two matrices named ***Ones*** and $\boldsymbol{R}$ are also defined to help carry out the evolutionary operator in MEC. The ***Ones*** is a matrix whose elements are all 1, while the $\boldsymbol{R}$ is a random matrix whose elements are randomly generated between [0, 1]. For convenience, the matrices used for MEC are notated and described in Table I.

B. Common Operations

Under the above matrix-based representation, the implementation of MEC relates to different typical matrix operations. Table II lists these related operations and gives their descriptions. For simplicity, the matrices used in Table II, $\boldsymbol{A}$ and $\boldsymbol{B}$, are $N \times D$ if they do not have subscripts.

TABLE I
Notations of the Matrices Used in Matrix-Based Evolutionary Computation

Notations	Descriptions
$\boldsymbol{X}_{N\times D}$	An $N\times D$ matrix represents the position of all the individuals in the population
$\boldsymbol{Fit}_{N\times 1}$	An $N\times 1$ matrix represents the fitness values of all the individuals in the population
$\boldsymbol{L}_{1\times D}$	A $1\times D$ matrix represents the lower bounds of all the dimensions (variables)
$\boldsymbol{U}_{1\times D}$	A $1\times D$ matrix represents the upper bounds of all the dimensions (variables)
$\boldsymbol{Prob}_{N\times 1}$	An $N\times 1$ matrix represents the probability of selecting each individual in MGA
$\boldsymbol{V}_{N\times D}$	An $N\times D$ matrix represents the velocity of all the individuals in the MPSO population
$\boldsymbol{pBest}_{N\times D}$	An $N\times D$ matrix represents the personal best positions of all the individuals in the MPSO population
$\boldsymbol{gBest}_{1\times D}$	A $1\times D$ matrix represents the globally best position (the individual with globally best fitness value)
$\boldsymbol{pBest\_Fit}_{N\times 1}$	An $N\times 1$ matrix represents the fitness values of the personal best positions of all the individuals in the MPSO population
$\boldsymbol{Vmax}_{1\times D}$	A $1\times D$ matrix represents the maximum velocity constraint of all the dimensions (variables) in MPSO
$\boldsymbol{Vmin}_{1\times D}$	A $1\times D$ matrix represents the minimum velocity constraint of all the dimensions (variables) in MPSO
Ones	A matrix whose elements are all 1
$\boldsymbol{R}$	A matrix whose elements are randomly between [0, 1]

III. Matrix-Based Genetic Algorithm

A. Representation and Initialization

The solution (individual) representation in MGA is shown as the matrix $\boldsymbol{X}$ in (2). To initialize the population $\boldsymbol{X}$ randomly in the feasible domain of each dimension, the $\boldsymbol{X}$ is initialized within the $\boldsymbol{L}$ and $\boldsymbol{U}$ by the help of the matrices ***Ones*** and $\boldsymbol{R}$ as

$$\boldsymbol{X} = \boldsymbol{Ones}_{N\times 1} \times (\boldsymbol{U} - \boldsymbol{L}) \circ \boldsymbol{R}_{N\times D} + \boldsymbol{Ones}_{N\times 1} \times \boldsymbol{L} \tag{3}$$

Herein the (3) is explained in details. Firstly, the subtraction operation (–) between $\boldsymbol{U}$ and $\boldsymbol{L}$ results in a $1 \times D$ matrix that constrains the initialization range. Then, the multiplication operation ($\times$) between $\boldsymbol{Ones}_{N\times 1}$ and $(\boldsymbol{U}–\boldsymbol{L})_{1\times D}$ results in an $N \times D$ matrix

$$\begin{pmatrix} u_1 - l_1 & \ldots & u_D - l_D \\ \vdots & \ddots & \vdots \\ u_1 - l_1 & \cdots & u_D - l_D \end{pmatrix}_{N\times D},$$

which is similar to that $\boldsymbol{Ones}_{N\times 1} \times \boldsymbol{L}_{1\times D}$ as

$$\begin{aligned} \boldsymbol{Ones}_{N\times 1} \times \boldsymbol{L}_{1\times D} &= \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \times \begin{pmatrix} l_1 \cdots l_D \end{pmatrix} \\ &= \begin{pmatrix} l_1 & \ldots & l_D \\ \vdots & \ddots & \vdots \\ l_1 & \cdots & l_D \end{pmatrix}_{N\times D} \end{aligned} \tag{4}$$

That is, the ***Ones*** matrix is used to help the $1 \times D$ ($\boldsymbol{U}–\boldsymbol{L}$) matrix and $\boldsymbol{L}$ matrix to be extended to $N \times D$ matrices in which

TABLE II
TYPICAL OPERATIONS IN MATRIX AND THEIR NOTATIONS

Name	Description
Addition operation (+)	$\boldsymbol{A}+\boldsymbol{B}=\begin{pmatrix} a_{11}+b_{11} & \cdots & a_{1D}+b_{1D} \\ \vdots & \ddots & \vdots \\ a_{N1}+b_{N1} & \cdots & a_{ND}+b_{ND} \end{pmatrix}$
Subtraction operation (–)	$\boldsymbol{A}-\boldsymbol{B}=\begin{pmatrix} a_{11}-b_{11} & \cdots & a_{1D}-b_{1D} \\ \vdots & \ddots & \vdots \\ a_{N1}-b_{N1} & \cdots & a_{ND}-b_{ND} \end{pmatrix}$
Multiplication operation (×)	$\boldsymbol{A}_{N\times D}\times\boldsymbol{B}_{D\times N}=\begin{pmatrix} \sum_{i=1}^{D} a_{1i}\times b_{i1} & \cdots & \sum_{i=1}^{D} a_{1i}\times b_{iN} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^{D} a_{Ni}\times b_{i1} & \cdots & \sum_{i=1}^{D} a_{Ni}\times b_{iN} \end{pmatrix}$
Scalar multiplication ($\cdot$)	$c\cdot\boldsymbol{A}=\begin{pmatrix} c\times a_{11} & \cdots & c\times a_{1D} \\ \vdots & \ddots & \vdots \\ c\times a_{N1} & \cdots & c\times a_{ND} \end{pmatrix}$
Hadamard product ($\circ$)	$\boldsymbol{A}\circ\boldsymbol{B}=\begin{pmatrix} a_{11}\times b_{11} & \cdots & a_{1D}\times b_{1D} \\ \vdots & \ddots & \vdots \\ a_{N1}\times b_{N1} & \cdots & a_{ND}\times b_{ND} \end{pmatrix}$
Transposition operation ($\boldsymbol{X}^T$)	$\boldsymbol{A}^T=\begin{pmatrix} a_{11} & \cdots & a_{D1} \\ \vdots & \ddots & \vdots \\ a_{1N} & \cdots & a_{DN} \end{pmatrix}$
Logical operation ($\leq$)	$\boldsymbol{A}\leq\boldsymbol{B}=\boldsymbol{C}, c_{i,j}=\begin{cases}1, \text{if } a_{i,j}\leq b_{i,j}\\ 0, \text{ otherwise}\end{cases}$
Maximum operation (max)	$a=\max(\boldsymbol{A})$, where a is the maximum element in $\boldsymbol{A}$
Minimum operation (min)	$a=\min(\boldsymbol{A})$, where a is the minimum element in $\boldsymbol{A}$
Maximum indexing (maxind)	$k=\max\text{ind}(\boldsymbol{A}_{N\times 1})$, where k is the row index of the maximum element in $\boldsymbol{A}_{N\times 1}$
Minimum indexing (minind)	$k=\min\text{ind}(\boldsymbol{A}_{N\times 1})$, where k is the row index of the minimum element in $\boldsymbol{A}_{N\times 1}$
Index operation ($\boldsymbol{X}[\boldsymbol{I}\mid\boldsymbol{J}]$)	$\boldsymbol{X}[\boldsymbol{I}\mid\boldsymbol{J}]=\begin{pmatrix} X_{I_1,J_1} & \cdots & X_{I_1,J_j} \\ \vdots & \ddots & \vdots \\ X_{I_i,J_1} & \cdots & X_{I_i,J_j} \end{pmatrix}$

all the rows are the same. Later, the formed $N \times D$ matrix

$$\begin{pmatrix} u_1-l_1 & \cdots & u_D-l_D \\ \vdots & \ddots & \vdots \\ u_1-l_1 & \cdots & u_D-l_D \end{pmatrix}_{N\times D}$$

carries out the Hadamard product with a random $N \times D$ matrix $\boldsymbol{R}_{N\times D}$, also results in an $N \times D$ matrix. This matrix pluses with the

$$\begin{pmatrix} l_1 & \cdots & l_D \\ \vdots & \ddots & \vdots \\ l_1 & \cdots & l_D \end{pmatrix}_{N\times D}$$

and at last forms the randomly initialized $\boldsymbol{X}$.

Therefore, (3) does not initialize the individuals one by one like traditional GA, but performs on all the individuals and all the dimensions simultaneously. More importantly, such an implementation of parallelism has been supported by the parallel routines of matrix toolbox and does not require the programming skills of users.

After the initialization, all the individuals go through the fitness evaluation and their fitness values are recorded in the $N \times 1$ matrix ***Fit*** as:

$$\boldsymbol{Fit}=f(\boldsymbol{X}) \tag{5}$$

Moreover, the globally best fitness value (*gBest_Fit*) can be directly determined by the matrix operation min or max as

$$gBest\_Fit=\begin{cases}\min(\boldsymbol{Fit}), \text{ if it is a minimum problem}\\ \max(\boldsymbol{Fit}), \text{ if it is a maximum problem}\end{cases} \tag{6}$$

B. Selection

After the initialization and fitness evaluation, the MGA goes through the evolutionary process by performing selection, crossover, and mutation generation by generation. To select N individuals from the population by a roulette wheel selection operator, the roulette should be built and the individuals are chosen based on it. The following steps illustrate this process.

Step 1: Calculate the probability matrix $\boldsymbol{Prob}_{N\times 1}$, which indicates the selected chance of each individual. In the minimum problems, the individual with lower fitness value has more chance to be selected. At the same time, to ensure that the sum of the selected probability of all individuals equals to 1, ***Prob*** is calculated through the sum of fitness value subtracts each fitness value and then divided by N-1 times of the sum of fitness. In the maximum problems, the individual with higher fitness value has more chance to be selected. ***Prob*** is calculated through each fitness value divided by the sum of all fitness values. Therefore, the ***Prob*** is calculated as

$$\boldsymbol{Prob}=\begin{cases}\frac{\boldsymbol{Ones}_{1\times N}\times\boldsymbol{Fit}-\boldsymbol{Fit}}{(N-1)\times(\boldsymbol{Ones}_{1\times N}\times\boldsymbol{Fit})}, \text{ minimum problem}\\ \frac{\boldsymbol{Fit}}{\boldsymbol{Ones}_{1\times N}\times\boldsymbol{Fit}}, \text{maximum problem}\end{cases} \tag{7}$$

where the multiplication operation between $\boldsymbol{Ones}_{1\times N}$ and ***Fit*** is used to calculate the sum of fitness.

Step 2: Define a $N \times 1$ roulette matrix ***ROU_MAT*** to record the accumulated probability of each individual. The ***ROU_MAT*** can be obtained by left multiply ***Prob*** with an $N \times N$ lower triangular one-matrix ***LTRI*** as

$$\begin{aligned}\boldsymbol{ROU\_MAT} &= \boldsymbol{LTRI} \times \boldsymbol{Prob} \\ &= \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix} \times \begin{pmatrix} Prob_1 \\ Prob_2 \\ \vdots \\ Prob_N \end{pmatrix} \\ &= \begin{pmatrix} Prob_1 \\ Prob_1 + Prob_2 \\ \vdots \\ Prob_1 + \cdots + Prob_N \end{pmatrix} \end{aligned} \tag{8}$$

Therefore, the last element in ***ROU_MAT*** is $Prob_1 + Prob_2 + \cdots + Prob_N = 1$. Similar to the roulette wheel selection operation in traditional GA, the ***ROU_MAT*** is used to determine which individual is selected. Given a random number between 0 and 1, then the first individual whose accumulated probability larger than this number is selected. In order to select N individuals at the same time, the Step 3 is designed based on the matrix operation.

Step 3: Generate an $N \times 1$ probability matrix ***R*** in which each element is a random number between 0 and 1. To find the first individual with the larger accumulated probability than each of these random numbers, the algorithm has to compare the values of elements in ***ROU_MAT*** and ***R***. To do this, the algorithm right multiplies a matrix $\boldsymbol{Ones}_{1\times N}$ to ***R*** to make the ***R*** an $N \times N$ matrix whose all columns are the same. For ***ROU_MAT***, the algorithm firstly transposes it and then extend it to $N \times N$ where all rows are the same. Then making the logical operation between the extended ***R*** and the extended ***ROU_MAT*** to get a $N \times N$ selected matrix named ***SEL_MAT***:

$$\begin{aligned}\boldsymbol{SEL\_MAT} &= (\boldsymbol{Ones}_{N\times 1} \times \boldsymbol{ROU\_MAT}^T) \\ &\leq (\boldsymbol{R}_{N\times 1} \times \boldsymbol{Ones}_{1\times N}) \end{aligned} \tag{9}$$

which is composed of 0 and 1, where 1 means the value in roulette is not larger than the random value (i.e., $\leq$) and 0 means larger. In (9), the left part of the logical operator is

$$\boldsymbol{Ones}_{N\times 1} \times \boldsymbol{ACCUM}^T = \begin{pmatrix} Prob_1 & Prob_1 + Prob_2 & \cdots & Prob_1 + \cdots + Prob_N \\ \vdots & & \ddots & \vdots \\ Prob_1 & Prob_1 + Prob_2 & \cdots & Prob_1 + \cdots + Prob_N \end{pmatrix}_{N\times N}.$$

Obviously, the values in each row are in ascending order. The right part of (9) is

$$\boldsymbol{R}_{N\times 1} \times \boldsymbol{Ones}_{1\times N} = \begin{pmatrix} R_1 & \cdots & R_1 \\ \vdots & \ddots & \vdots \\ R_N & \cdots & R_N \end{pmatrix}_{N\times N}.$$

Therefore, the selected matrix ***SEL_MAT*** has the characteristic that in each row the first several elements are 1 and the rest elements are 0. In this sense, the index of the first 0 is the individual that to be selected. It is also interesting to observe that the sum of ones before this element is actually the same as its index. Therefore, right multiplying an $N \times 1$ matrix ***Ones*** can get the index matrix ***SET_IND*** as

$$\boldsymbol{SEL\_IND} = \boldsymbol{SEL\_MAT} \times \boldsymbol{Ones}_{N\times 1} \tag{10}$$

Step 4: Replace the population with the selected individuals. The matrix ***SET_IND*** determines which individuals are selected and using index operation can extract them as (11). After that, just copy ***X_SEL*** to ***X*** to replace the original population.

$$\boldsymbol{X\_SEL} = \boldsymbol{X}[\boldsymbol{SEL\_IND}|1, \ldots, D] \tag{11}$$

$$\boldsymbol{X} = \boldsymbol{X\_SEL} \tag{12}$$

C. Crossover

In the crossover operator for GA, there is a crossover probability P_c that controls whether an individual takes part in the crossover. In fact, the expected number of crossover individuals is $NC = N \times P_c$. In the MGA, to make the algorithm easier and more suitable for the matrix-based representation, the MGA chooses the first NC individuals for crossover from the population formed by the above selection operation. It should be noted that the population has been randomly shuffled in the selection operation and therefore these NC chosen individuals are without loss of generality. If NC is odd, the MGA chooses one more individual so that $NC = NC + 1$. Then the first half of the individuals correspondingly mate with the second half of the individuals one by one. This operation is legal out of two considerations. Firstly, keeping the number of crossover individuals to be a constant has less influence on the performance. Traditionally, the number of crossover individuals in each generation is dynamic. However, with a large number of independent runs, the expected value is $N \times P_c$. Therefore, the MGA sets the number of crossover individuals NC as the fixed value $N \times P_c$ during the evolution. Secondly, the randomness is kept. Although the MGA chooses the first NC individuals and mate the first half individuals with the second half individuals directly, the roulette selection provides enough randomness to the selected population in each generation. Roulette selection is a totally random process and the order of selected individuals is also random.

Crossover is that the variables before the crossover point in the first parent combine with the variables after the crossover point in the second parent to form one offspring and the other variables of parents form the second offspring. In order to do this operation using matrix, the MGA can build two matrices where one matrix is composed of part of original values and part of zeros and the other matrix consists of correspondingly part of zeros and part of original values. Adding these two matrices can directly result in the offspring matrix. The MGA achieves the crossover by following procedure.

Step 1: Generate a $\frac{NC}{2} \times 1$ matrix ***C_POS*** randomly, where each element is an integer between 1 and D, representing the crossover position of each pair of individuals.

Step 2: Generate a $1 \times D$ permutation matrix ***PERM*** $= (1, 2, \ldots, D)$.

Step 3: Right multiply a $1 \times D$ ***Ones*** matrix to extend ***C_POS*** to $\frac{NC}{2} \times D$, with all the columns the same. Left multiply a $\frac{NC}{2} \times 1$ ***Ones*** matrix to extend ***PERM*** to $\frac{NC}{2} \times D$, with all the

rows the same. Make a logical operation between the extended ***C_POS*** and ***PERM*** to get the mask matrix ***MASK*** as

$$\begin{aligned}\boldsymbol{MASK}_{\frac{NC}{2}\times D} &= (\boldsymbol{C\_POS}_{\frac{NC}{2}\times 1} \times \boldsymbol{Ones}_{1\times D}) \\ &\leq (\boldsymbol{Ones}_{\frac{NC}{2}\times 1} \times \boldsymbol{PERM}_{1\times D}) \end{aligned} \quad (13)$$

This process is similar to (9) and ***MASK*** has the same characteristic with ***SEL_MAT*** which consists of the first several zeros and the remaining ones, but the meaning is different. In ***MASK***, the first several zeros mean the variables before the crossover position will be discarded while the remaining ones mean the variables after the crossover position need to be retained. Meanwhile, the complemented matrix $\overline{\boldsymbol{MASK}}$ of the ***MASK*** is obtained by $\overline{\boldsymbol{MASK}_{\frac{NC}{2}\times D}} = \boldsymbol{Ones}_{\frac{NC}{2}\times D} - \boldsymbol{MASK}_{\frac{NC}{2}\times D}$.

Step 4: The offspring individuals are obtained by two steps. Firstly, in this step the first *NC*/2 offspring individuals are obtained as

$$\begin{aligned}&\boldsymbol{OffSpring}\left[1,\ldots,\frac{NC}{2}\middle|1,\ldots,D\right] \\ &= \boldsymbol{X}\left[1,\ldots,\frac{NC}{2}\middle|1,\ldots,D\right] \circ \boldsymbol{MASK} \\ &\quad + \boldsymbol{X}\left[\frac{NC}{2}+1,\ldots NC\middle|1,\ldots,D\right] \circ \overline{\boldsymbol{MASK}} \end{aligned} \quad (14)$$

Herein, the Hadamard product between the first half of *NC* individuals and ***MASK*** (i.e., $\boldsymbol{X}[1,\ldots,\frac{NC}{2}|1,\ldots,D] \circ \boldsymbol{MASK}$) is a matrix where each row stands for a pre-offspring individual with the first several elements are zero and the following elements are all from the corresponding positions of ***X***. Herein the MGA use the "pre-offspring" because the first several elements of the individuals are still needed to be crossed from other parents. Therefore, the Hadamard product between the second half of *NC* individuals and $\overline{\boldsymbol{MASK}}$ (i.e., $\boldsymbol{X}[\frac{NC}{2}+1,\ldots NC|1,\ldots,D] \circ \overline{\boldsymbol{MASK}}$) is to provide this information. The sum of these two matrices results in the first *NC*/2 offspring individuals.

Step 5: Secondly, in this step, the second *NC*/2 offspring individuals are obtained as

$$\begin{aligned}&\boldsymbol{OffSpring}\left[\frac{NC}{2}+1,\ldots,NC\middle|1,\ldots,D\right] \\ &= \boldsymbol{X}\left[\frac{NC}{2}+1,\ldots,NC\middle|1,\ldots,D\right] \circ \boldsymbol{MASK} \\ &\quad + \boldsymbol{X}\left[1,\ldots,\frac{NC}{2}\middle|1,\ldots,D\right] \circ \overline{\boldsymbol{MASK}} \end{aligned} \quad (15)$$

whose explanations are similar to that of (14).

Step 6: Using the index operation to replace the parent individuals using the offspring.

$$\boldsymbol{X}[1,\ldots NC|1,\ldots,D] = \boldsymbol{OffSpring} \quad (16)$$

It should be noted that the individuals that do not take the crossover operations remain the same as their previous generation.

D. Mutation

The mutation operator is done on genes rather than the individuals, which means each bit has a chance to mutate. The process is to judge which bits need to mutate and then keep the unchanged bits and reinitialize the mutated bits. The procedure is as follows.

Step 1: Generate a random decimal matrix $\boldsymbol{R}_{N\times D}$, where each element is between 0 and 1, meaning the mutation rate of each gene. If the mutation rate of a bit is smaller than P_m, it will mutate. Otherwise, the gene is kept.

Step 2: Left multiply an $N \times 1$ ***Ones*** matrix and right multiply a $1 \times D$ ***Ones*** matrix to extend the mutation probability P_m to an $N \times D$ matrix ***PM***. Each element in ***PM*** is P_m. Make a logical operation between ***R*** and ***PM*** to determine which variables need to mutate.

$$\boldsymbol{PM} = \boldsymbol{Ones}_{N\times 1} \times P_m \times \boldsymbol{Ones}_{1\times D} \quad (17)$$

$$\boldsymbol{MUT\_IND} = \boldsymbol{R} \leq \boldsymbol{PM} \quad (18)$$

The result ***MUT_IND*** consists of most zeros and several ones, where ones represent the mutation variables.

Step 4: Using Hadamard product to retain the unchanged part and reinitialize the mutation positions. The reinitialization process in (19) is the same as the initialization process in (3).

$$\boldsymbol{RE\_X} = \boldsymbol{Ones}_{N\times 1} \times (\boldsymbol{U} - \boldsymbol{L}) \circ \boldsymbol{R}_{N\times D} + \boldsymbol{Ones}_{N\times 1} \times \boldsymbol{L} \quad (19)$$

$$\boldsymbol{X} = \boldsymbol{X} \circ (1 - \boldsymbol{MUT\_IND}) + \boldsymbol{RE\_X} \circ \boldsymbol{MUT\_IND} \quad (20)$$

In (20), matrix ($1-$***MUT_IND***) consists of most of the ones which mean the unchanged variables and several zeros which are to be reset. Unchanged index matrix Hadamard products with population ***X*** and the mutation index matrix Hadamard products with the reinitialization matrix. At last, the sum of these two results is the population ***X*** after mutation.

E. Whole Algorithm

The pseudocode of MGA is outlined in Algorithm 1.

IV. Matrix-Based Particle Swarm Optimization

A. Representation and Initialization

The representation in MPSO is little different from MGA because MPSO not only has the population matrix ***X*** that represents the positions of the individuals, but also has the matrices ***V*** and ***pBest*** that represents the velocity and personal best position, respectively. Herein, the ***X*** is initialized the same as (3) and the ***V*** is initialized similar to the ***X***, except that ***U*** is replaced by ***Vmax*** and ***L*** is replaced by ***Vmin***.

After the initialization of ***X*** and ***V***, the MPSO can obtain the fitness values of all the individuals through fitness evaluation and record them in the $N \times 1$ matrix ***Fit*** like (5). Moreover, the ***pBest*** is set directly the same as ***X*** and the ***pBest_Fit*** is the same as ***Fit***. At last, the globally best fitness value (*gBest_Fit*) is determined like (6). Moreover, assume that the optimization problem is a maximization problem, the MPSO can use maxind

Algorithm 1: Matrix-Based Genetic Algorithm.

Input: The size of population N, the dimension of the problem D, the crossover rate P_c, the mutation rate P_m, the maximal generation *max_gen*

Begin

/*Initialization*/

$\boldsymbol{X}_{N\times D} = \boldsymbol{Ones}_{N\times 1} \times (\boldsymbol{U}-\boldsymbol{L})_{1\times D} \circ \boldsymbol{R}_{N\times D} + \boldsymbol{Ones}_{N\times 1} \times \boldsymbol{L}_{1\times D}$

/*Evaluation*/

$\boldsymbol{Fit}_{N\times 1} = f(\boldsymbol{X})$

/*Update the best solution*/

$gBest\_Fit =$ $\begin{cases} \min(\boldsymbol{Fit}), \text{if it is a minimum problem} \\ \max(\boldsymbol{Fit}), \text{if it is a maximum problem} \end{cases}$

For $g = 1$ to *max_gen* **Do**

/*Selection*/

$\boldsymbol{Prob} = \begin{cases} \frac{\boldsymbol{Ones}_{1\times N}\times \boldsymbol{Fit} - \boldsymbol{Fit}}{(N-1)\times(\boldsymbol{Ones}_{1\times N}\times \boldsymbol{Fit})}, \text{minimum problem} \\ \frac{\boldsymbol{Fit}}{\boldsymbol{Ones}_{1\times N}\times \boldsymbol{Fit}}, \text{maximum problem} \end{cases}$

$\boldsymbol{ROU\_MAT}_{N\times 1} = \boldsymbol{LTRI}_{N\times N} \times \boldsymbol{Prob}$

$\boldsymbol{SEL\_MAT} = (\boldsymbol{Ones}_{N\times 1} \times \boldsymbol{ROU\_MAT}^T) \le (\boldsymbol{R}_{N\times 1} \times \boldsymbol{Ones}_{1\times N})$

$\boldsymbol{SEL\_IND}_{N\times 1} = \boldsymbol{SEL\_MAT} \times \boldsymbol{Ones}_{N\times 1}$

$\boldsymbol{X\_SEL}_{N\times D} = \boldsymbol{X}[\boldsymbol{SEL\_IND}|1,\ldots,D]$

$\boldsymbol{X} = \boldsymbol{X\_SEL}$

/*Crossover*/

$\boldsymbol{PERM} = (1, 2, \ldots, D)$

$\boldsymbol{MASK}_{\frac{NC}{2}\times D} = (\boldsymbol{C\_POS}_{\frac{NC}{2}\times 1} \times \boldsymbol{Ones}_{1\times D}) \le (\boldsymbol{Ones}_{\frac{NC}{2}\times 1} \times \boldsymbol{PERM}_{1\times D})$

$\overline{\boldsymbol{MASK}_{\frac{NC}{2}\times D}} = \boldsymbol{Ones}_{\frac{NC}{2}\times D} - \boldsymbol{MASK}_{\frac{NC}{2}\times D}$

$\boldsymbol{OffSpring}[1,\ldots,\frac{NC}{2}|1,\ldots,D] = \boldsymbol{X}[1,\ldots,\frac{NC}{2}|1,\ldots,D] \circ \boldsymbol{MASK} + \boldsymbol{X}[\frac{NC}{2}+1,\ldots NC|1,\ldots,D] \circ \overline{\boldsymbol{MASK}}$

$\boldsymbol{OffSpring}[\frac{NC}{2}+1,\ldots,NC|1,\ldots,D] = \boldsymbol{X}[\frac{NC}{2}+1,\ldots,NC|1,\ldots,D] \circ \boldsymbol{MASK} + \boldsymbol{X}[1,\ldots,\frac{NC}{2}|1,\ldots,D] \circ \overline{\boldsymbol{MASK}}$

$\boldsymbol{X}[1,\ldots NC|1,\ldots,D] = \boldsymbol{OffSpring}$

/*Mutation*/

$\boldsymbol{PM}_{N\times D} = \boldsymbol{Ones}_{N\times 1} \times P_m \times \boldsymbol{Ones}_{1\times D}$

$\boldsymbol{MUT\_IND}_{N\times D} = \boldsymbol{R}_{N\times D} \le \boldsymbol{PM}$

$\boldsymbol{RE\_X}_{N\times D} = \boldsymbol{Ones}_{N\times 1} \times (\boldsymbol{U}-\boldsymbol{L}) \circ \boldsymbol{R}_{N\times D} + \boldsymbol{Ones}_{N\times 1} \times \boldsymbol{L}$

$\boldsymbol{X} = \boldsymbol{X} \circ (1 - \boldsymbol{MUT\_IND}) + \boldsymbol{RE\_X} \circ \boldsymbol{MUT\_IND}$

/*Evaluation*/

$\boldsymbol{Fit}_{N\times 1} = f(\boldsymbol{X})$

/*Update the best solution fitness*/

$gBest\_Fit = \{ \begin{matrix} \min(\boldsymbol{Fit}), \text{minimumproblem} \\ \max(\boldsymbol{Fit}), \text{maximumproblem} \end{matrix}$

End for

Output: The found best solution fitness $gBest\_Fit$.

End

operator to obtain the index of individual with the best ***pBest*** fitness as

$$index = \max \text{ind}(\boldsymbol{pBest\_Fit}) \tag{21}$$

where the maxind() is defined in Table II.

B. Velocity and Position Update

During the iteration of MPSO, the individuals perform the velocity update and position update generation by generation, so as to gradually approach the global optimum. The velocity and position can be updated by (22) and (23), respectively.

$$\boldsymbol{V} = w \times \boldsymbol{V} + c_1 \times \boldsymbol{R}_1 \circ (\boldsymbol{pBest} - \boldsymbol{X}) + c_2 \times \boldsymbol{R}_2 \circ (\boldsymbol{Ones} \times \boldsymbol{gBest} - \boldsymbol{X}) \tag{22}$$

$$\boldsymbol{X} = \boldsymbol{X} + \boldsymbol{V} \tag{23}$$

where w is the inertia weight, c_1 and c_2 are the acceleration coefficients, $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ are two $N \times D$ matrices with uniformly distributed random numbers.

It should be noted that ***gBest*** is $1 \times D$ matrix that is actually a row from the $N \times D$ matrix ***pBest***. That is, the ***pBest*** of the *index* (i.e., the globally best in (21)) individual is actually the ***gBest*** $=$ ***pBest***(*index*). However, in order to make the $N \times D$ matrix $\boldsymbol{X}$ can learn from the $1 \times D$ matrix ***gBest***, the (22) will left multiply an $N \times 1$ matrix ***Ones*** so that ***gBest*** is extended to an $N \times D$ matrix where all the rows are the same, similar to the process of Eq. (4).

As the elements in $\boldsymbol{V}$ and $\boldsymbol{X}$ should be not out of the bound, the detection and handling of the bounds will be performed on $\boldsymbol{V}$ and $\boldsymbol{X}$ immediately after they have been updated. This can be carried out through logic operation and Hadamard product. For better descriptions, the MPSO takes the $\boldsymbol{X}$ and its upper bound $\boldsymbol{U}$ as an example. The detection and handling for the upper bound can be represented as

$$\boldsymbol{LOGIC}_{N\times D} = \boldsymbol{X} > (\boldsymbol{Ones} \times \boldsymbol{U}) \tag{24}$$

Herein, the $1 \times D$ matrix $\boldsymbol{U}$ is firstly expanded to an $N \times D$ matrix, where each row is the same with each other, similar as the process of Eq. (4), and then compared with the $N \times D$ matrix $\boldsymbol{X}$. The ***LOGIC*** records the elements in $\boldsymbol{X}$ that are larger than the corresponding upper bound. That is, an element in ***LOGIC***, say l_{ij}, is 1 if the element of row i and column j in $\boldsymbol{X}$ is larger than j^{th} value of $1 \times D$ matrix $\boldsymbol{U}$. Otherwise, it will be 0. In this way, the handling of upper bound can be achieved by:

$$\boldsymbol{X} = \boldsymbol{LOGIC} \circ \boldsymbol{U} + (\boldsymbol{1} - \boldsymbol{LOGIC}) \circ \boldsymbol{X} \tag{25}$$

The aim of this operation is to set the value out of the upper bound as the upper bound. To be more specifically, the element of $\boldsymbol{X}$ will be set as the value of the corresponding upper bound if the element in the same position of ***LOGIC*** is 1, while the element of X will not be changed if the element in the same position of ***LOGIC*** is 0, which is similar to Eq. (20). It should be noted that the operation to deal with the lower bound of $\boldsymbol{X}$ is similar to the operation to deal with the upper bound, and therefore is not repeated herein.

C. Update of Personal and Global Best Positions

After the update of velocities and positions, the ***Fit*** will be re-calculated through fitness evaluation. Then the personal best position ***pBest*** should be updated according to the relationship between ***Fit*** and ***pBest_Fit***. The relationship can be determined by using logic operation and Hadamard product operation. Firstly, to find which individuals have found positions that are better than their previous ***pBest***, an $N \times 1$ matrix ***LOGIC*** is computed through logic operation as (26) or (27) for maximization or minimization problem, respectively.

$$\boldsymbol{LOGIC} = \boldsymbol{pBest\_Fit} > \boldsymbol{Fit} \tag{26}$$

$$\boldsymbol{LOGIC} = \boldsymbol{pBest\_Fit} < \boldsymbol{Fit} \tag{27}$$

Then, the personal best positions matrix and the personal best fitness matrix can be updated through Hadamard product, which are illustrated in (28) and (29).

$$\begin{aligned}\boldsymbol{pBest} = {} & \boldsymbol{LOGIC} \times \boldsymbol{Ones} \circ \boldsymbol{X} \\ & + (\mathbf{1} - \boldsymbol{LOGIC} \times \boldsymbol{Ones}) \circ \boldsymbol{pBest}\end{aligned} \tag{28}$$

$$\begin{aligned}\boldsymbol{pBest\_Fit} = {} & \boldsymbol{LOGIC} \circ \boldsymbol{Fit} \\ & + (\mathbf{1} - \boldsymbol{LOGIC}) \circ \boldsymbol{pBest\_Fit}\end{aligned} \tag{29}$$

The main idea of (28) and (29) are the same, which are similar to Eq. (20). The difference between them is that a $1 \times D$ matrix ***Ones*** is employed in (28) to expand the ***LOGIC*** because the size of ***X*** and ***pBest*** are $N \times D$.

After the update of personal best positions and fitness values, the new global best position and its corresponding index can be determined again by (21).

The MPSO will finish if the stop criteria (i.e., the maximum of generation is reached) are met. Otherwise, the algorithm will run into next generation.

D. Whole Algorithm

To provide a demonstration of the complete MPSO algorithm, its pseudo code is given in Algorithm 2. For simplicity, this pseudo code is the version of MPSO for maximization problems.

Algorithm 2: Matrix-Based Particle Swarm Optimization.

Input: The size of population N, the dimension of the problem D, the parameters w, the c_1 and c_2, maximal generation *max_gen*

Begin

/*Initialization*/

$\boldsymbol{X}_{N\times D} = \boldsymbol{Ones}_{N\times 1} \times (\boldsymbol{U} - \boldsymbol{L})_{1\times D} \circ \boldsymbol{R}_{N\times D} + \boldsymbol{Ones}_{N\times 1} \times \boldsymbol{L}_{1\times D}$

$\boldsymbol{V}_{N\times D} = \boldsymbol{Ones}_{N\times 1} \times (\boldsymbol{Vmax} - \boldsymbol{Vmin})_{1\times D} \circ \boldsymbol{R}_{N\times D} + \boldsymbol{Ones}_{N\times 1} \times \boldsymbol{Vmin}_{1\times D}$

/*Evaluation*/

$\boldsymbol{Fit}_{N\times 1} = f(\boldsymbol{X})$

/*Update the best solution*/

$\boldsymbol{gBest} = \boldsymbol{X}[\max \mathrm{ind}(\boldsymbol{Fit})|1, \ldots, D]$

/*Update the best fitness*/

$gBest\_Fit = \max(\boldsymbol{Fit})$

/*Update the personal best position*/

$\boldsymbol{pBest} = \boldsymbol{X}$

/*Update the personal best fitness*/

$\boldsymbol{pBest\_Fit} = \boldsymbol{Fit}$

For $g = 1$ to *max_gen* **Do**

/*Update velocity*/

$\boldsymbol{V} = w \times \boldsymbol{V} + c_1 \times \boldsymbol{R}_1 \circ (\boldsymbol{pBest} - \boldsymbol{X}) + c_2 \times \boldsymbol{R}_2 \circ (\boldsymbol{Ones} \times \boldsymbol{gBest} - \boldsymbol{X})$

/*Perform bound detection and handling on $\boldsymbol{V}$ */

$\boldsymbol{LOGIC}_{N\times D} = \boldsymbol{V} > (\boldsymbol{Ones} \times \boldsymbol{Vmax})$

$\boldsymbol{V} = \boldsymbol{LOGIC} \circ \boldsymbol{Vmax} + (\mathbf{1} - \boldsymbol{LOGIC}) \circ \boldsymbol{V}$

$\boldsymbol{LOGIC}_{N\times D} = \boldsymbol{V} < (\boldsymbol{Ones} \times \boldsymbol{Vmin})$

$\boldsymbol{V} = \boldsymbol{LOGIC} \circ \boldsymbol{Vmin} + (\mathbf{1} - \boldsymbol{LOGIC}) \circ \boldsymbol{V}$

/*Update position*/

$\boldsymbol{X} = \boldsymbol{X} + \boldsymbol{V}$

/* Perform bound detection and handling on $\boldsymbol{X}$*/

$\boldsymbol{LOGIC}_{N\times D} = \boldsymbol{X} > (\boldsymbol{Ones} \times \boldsymbol{U})$

$\boldsymbol{X} = \boldsymbol{LOGIC} \circ \boldsymbol{U} + (\mathbf{1} - \boldsymbol{LOGIC}) \circ \boldsymbol{X}$

$\boldsymbol{LOGIC}_{N\times D} = \boldsymbol{X} < (\boldsymbol{Ones} \times \boldsymbol{L})$

$\boldsymbol{X} = \boldsymbol{LOGIC} \circ \boldsymbol{L} + (\mathbf{1} - \boldsymbol{LOGIC}) \circ \boldsymbol{X}$

/*Evaluation*/

$\boldsymbol{Fit}_{N\times 1} = f(\boldsymbol{X})$

/*Update the personal best position*/

$\boldsymbol{LOGIC} = \boldsymbol{pBest\_Fit} > \boldsymbol{Fit}$

$\boldsymbol{pBest} = \boldsymbol{LOGIC} \cdot \boldsymbol{Ones} \circ \boldsymbol{X} + (\mathbf{1} - \boldsymbol{LOGIC} \cdot \boldsymbol{Ones}) \circ \boldsymbol{pBest}$

/*Update the personal best fitness*/

$\boldsymbol{pBest\_Fit} = \boldsymbol{LOGIC} \circ \boldsymbol{Fit} + (\mathbf{1} - \boldsymbol{LOGIC}) \circ \boldsymbol{pBest\_Fit}$

/*Update the best solution*/

$\boldsymbol{gBest} = \boldsymbol{pBest}[\max \mathrm{ind}(\boldsymbol{pBest\_Fit})|1, \ldots, D]$

/*Update the best solution fitness*/

$gBest\_Fit = \max(\boldsymbol{pBest\_Fit})$

End for

Output: The found best solution fitness $gBest\_Fit$.

End

V. Time Complexity Analyses

A. Time Complexity of Common Operators

Before discussing the time complexity of MGA and MPSO, this paper firstly analyzes the time complexity (TC) of common operators, so that the later descriptions can be clearer. Among the common operators defined in Section II, there are only three kinds of operators that require different TCs, they are bitwise operator, multiplication operator, and max(min) operator.

1) Bitwise Operator: The bitwise operator performs calculations in a bit-by-bit fashion, which includes matrix addition, matrix subtraction, and Hadamard product. Therefore, these operators on $N \times D$ matrix needs $N \times D$ bit-by-bit calculations, resulting in the time complexity as $O(N \times D)$. However, as MEC

can use the parallel routine of matrix to execute all these $N \times D$ bit-by-bit calculations simultaneously on a set of resources, the time complexity can be further reduced to $O(\frac{N \times D}{P})$, where P is the number of the parallel processes [41].

2) Multiplication Operator: For the multiplication between a $J \times K$ matrix and a $K \times L$ matrix, the TC is $O(J \times K \times L)$. However, the matrix multiplication can be accelerated by many methods, such as block matrix multiplication [45], Fox algorithm [46], and Cannon algorithm. Therefore, in MEC, the TC of the multiplication operator can be reduced to $O(\frac{J \times K \times L}{P})$, where P is also the number of parallel processes [47].

3) Max (Min) and Maxind (Minind) Operator: The max (min) operator is to find the maximum (minimum) value in an array. By employing enough processes and divide-and-conquer mechanism, its TC is $O(\log_2 N)$ on single instruction multiple data with exclusive read exclusive write (SIMD-EREW) machine, and is $O(1)$ on SIMD with concurrent read concurrent write (SIMD-CRCW) machine [47]. As SIMD-EREW is commonly used, this paper employ the $O(\log_2 N)$ as the TC for max (min) operator. Similarly, the maxind (minind) operator is to get the index of maximum (minimum) value in an array. Therefore, they share the same TC with the max and min operators.

B. Time Complexity of Matrix-Based Genetic Algorithm

This part gives the theoretical analysis of the TC of each part of MGA, including initialization, roulette selection, crossover, mutation, and the globally best solution update. Finally, this part will make also a comprehensive analysis of MGA.

1) Initialization: According to (3), the random initialization uses the matrix multiplication and bitwise operation. Multiplication's TC is $O(N \times 1 \times D)$. For Hadamard product, subtraction, and addition operations, the TC is $O(N \times D)$. Since the fitness evaluation is problem-dependent, its TC is not taken into considerations. Therefore, the TC of initialization is $O(\frac{N \times D}{P})$ with P processors.

2) Roulette Selection: In the selection, the matrix operations include multiplication, subtraction, and logical operations. For multiplication, the TC is $O(1 \times N \times 1 + N \times N \times 1 + N \times 1 \times N + N \times N \times 1)$, which is $O(N^2)$. Subtraction and logical operation are both bitwise operations, so the TC of is $O(N \times D)$. Therefore, the TC of the roulette selection is $O(\frac{N \times D + N^2}{P})$ with P processors.

3) Crossover: There are multiplication operation and three kinds of bitwise operations (i.e., logical, Hadamard product, and complementation) in the crossover. The TC of multiplication is $O(\frac{NC}{2} \times 1 \times D + 1 \times D \times \frac{NC}{2})$, which is $O(D \times \frac{NC}{2})$. The TC of the bitwise operations in the crossover is $O(D \times \frac{NC}{2})$. Therefore, the total TC of crossover is $O(\frac{D \times \frac{NC}{2}}{P})$. with P processors.

4) Mutation: The reinitialization in the mutation is the same with the initialization of the algorithm. The TC of the multiplication operator in mutation is $O(1 \times N \times D)$ while the TC of the bitwise operators in mutation is $O(N \times D)$. Therefore, the TC of the mutation is $O(\frac{N \times D}{P})$ with P processors.

5) Update the Globally Best Solution: This operation is to find and update the globally best solution. Its TC is the same as max (min) operator with $O(\log_2 N)$.

From the analyses above, the total TC of initialization, selection, crossover, and mutation in MGA is $O(\frac{N \times D + N^2 + D \times \frac{NC}{2}}{P})$. Since $NC = N \times Pc$ (i.e., the expected number of crossover individuals as defined in Section III-C) and $Pc < 1$, the final TC can be written as $O(\frac{N \times D + N^2}{P})$. Suppose that $P \geq \frac{NC}{2}$, the TC of finding the best solution is $O(\log_2 N)$ and the TC of MGA is $O(\frac{N \times D + N^2}{P} + \log_2 N)$.

C. Time Complexity of Matrix-Based Particle Swarm Optimization

This part gives the theoretical analysis of the TC of each part of MPSO, including initialization, velocity and position update, and personal and global best position update. Finally, this part will also make a comprehensive analysis of MPSO.

1) Initialization: Like the initialization in MGA, the MPSO performs initializations for $\boldsymbol{X}$ and $\boldsymbol{V}$ through the matrix multiplication and bitwise operations. The TC for multiplication is $O(N \times 1 \times D)$, while for Hadamard product, subtraction, and addition operations, the TC is $O(N \times D)$. Therefore, the total TC of initialization in MPSO is $O(\frac{N \times D}{P})$ with P processors. It should be noted that the TC of fitness evaluation has not been considered due to its problem-dependent feature.

2) Velocity and Position Update: As shown in (22) to (25), the operations for velocity and position update are all bitwise operations, which have the TC of $O(N \times D)$. Therefore, the TC for velocity and position update is also $O(\frac{N \times D}{P})$ with P processors.

3) Personal and Global Best Position Update: When computing the matrix of ***gBest*** and ***pBest***, the TC for (26), (27), and (29) are $O(N \times D)$ since they are bitwise operations, while the TC for the maxind operation in (21) is $O(\log_2 N)$. In addition, the matrix multiplication in (28) have the TC of $O(N \times 1 \times D)$. Hence, the TC for computing ***gBest*** and ***pBest*** is $O(\frac{N \times D}{P} + \log_2 N)$ with P processors.

Based on the above analyses, the total TC of MPSO is $O(\frac{N \times D}{P} + \log_2 N)$ when using N individuals and P processers to optimize the D-dimensional problems.

At last, the TC of the common operators and the MGA and MPSO are summarized in Table III.

VI. Experimental Studies

A. Experimental Configurations

The efficiency of MEC is that it can accelerate the computational speed but not improve the problem-solving ability. Therefore, this part conducts the experiments based on a typical Sphere function herein to evaluate the computational efficiency of MEC. The Sphere function is defined as

$$f(x) = \sum_{j=1}^{D} x_j^2, \ -5.12 \leq x_j \leq 5.12 \tag{30}$$

where D is the dimension of the problem and is set as 100.

To ensure the fairness of the comparisons, the EC and MEC algorithms use the same parameters and run on the same machines. The population size N is 100 and the maximal generation is 3000. The experiments are implemented by Matlab 2014 and

TABLE III
TC OF THE COMMON OPERATORS AND THE MATRIX-BASED GENETIC ALGORITHM AND MATRIX-BASED PARTICLE SWARM OPTIMIZATION

Objects	Name	TC
The TC of the Common Operators	Bitwise Operator	$O(\frac{N\times D}{P})$
	Multiplication Operator	$O(\frac{J\times K\times L}{P})$.
	Max (Min) and Maxind (Minind) Operator	$O(\log_2 N)$
The TC of MGA	Initialization	$O(\frac{N\times D}{P})$
	Roulette selection	$O(\frac{N\times D+N^2}{P})$
	Crossover	$O(\frac{D\times \frac{NC}{2}}{P})$
	Mutation	$O(\frac{N\times D}{P})$
	Update the globally best solution	$O(\log_2 N)$
	Total	$O(\frac{N\times D+N^2}{P}+\log_2 N)$
The TC of MPSO	Initialization	$O(\frac{N\times D}{P})$
	Velocity and position update	$O(\frac{N\times D}{P})$
	pbest and ***gbest*** update	$O(\frac{N\times D}{P}+\log_2 N)$
	Total	$O(\frac{N\times D}{P}+\log_2 N)$

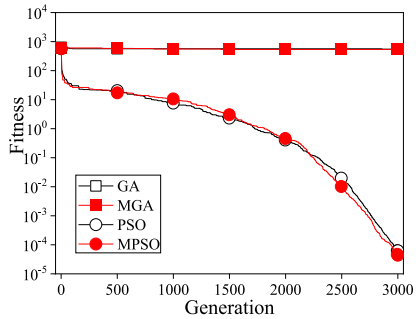

Fig. 4. Convergence curves of different EC and MEC algorithms.

are run in machine with 4 cores, configured as Intel(R) Xeon(R) CPU E3-1225 with 3.3 GHz, 8G memory, and Ubuntu 16.04.2 LTS operation system.

B. Comparisons on Problem-Solving Ability

The fitness values of different EC and MEC algorithms during the evolutionary process are compared in Fig. 4. The comparisons results show that the problem-solving abilities of MGA and GA are the same, and so are the PSO and MPSO. This is because that the MEC can use matrix-based population representation and operator to perform efficient population reproduction in EC algorithms, which can improve the computational time efficiency without influencing the problem-solving ability of original EC algorithms.

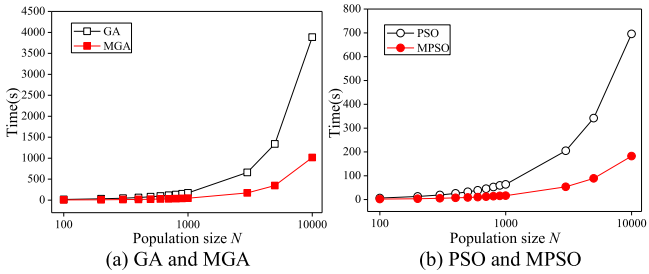

Fig. 5. Computational time efficiency of MEC algorithms with different population size N. (a) GA and MGA. (b) PSO and MPSO.

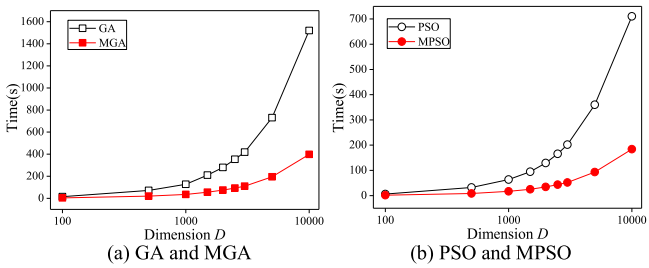

Fig. 6. Computational time efficiency of MEC algorithms on different dimensions *D*. (a) GA and MGA. (b) PSO and MPSO.

C. Comparisons on Computational Speed

As for the comparisons on the efficiency of computational speed, this part conducts experiments on different population size *N*, different dimension *D*, and different number of CPU cores. It should be noted that the computational time for fitness evaluation is not included in the comparisons because the MEC only accelerates the computational efficiency of evolutionary operators.

Firstly, the experiment fixes the dimension *D* as 100 and the number of CPU cores as 4, while the population size *N* varies from 100 to 10000. The experimental results are compared in Fig. 5. The experimental results show that the efficiency of MEC in computational time becomes more evident as the number of population size increases, no matter for MGA or MPSO. This indicates that when the algorithm uses a large population size to deal with large-scale or super large-scale optimization, the advantages of MEC will become more significant in reducing the computational time.

Secondly, the experiment fixes the population size *N* as 100 and the number of CPU cores as 4, while the dimension *D* varies from 100 to 10000. The experimental results are compared in Fig. 6. The experimental results show that the computational time efficiency of MEC also becomes more evident as the number of problem dimension increases, no matter for MGA or MPSO. Therefore, the MEC algorithms are much more suitable for solving large-scale or super large-scale optimization problems by reducing the computational time.

Finally, the experiment fixes the population size *N* as 100 and the dimension *D* as 100, while the number of CPU cores varies from 1 to 64. Herein, the experimental environments is changed to the machine with 68 cores, configured as Intel(R) Xeon Phi (TM) CPU 7250 with 1.40 GHz, and with 112G memory. The speedup results are compared in Fig. 7. The experimental

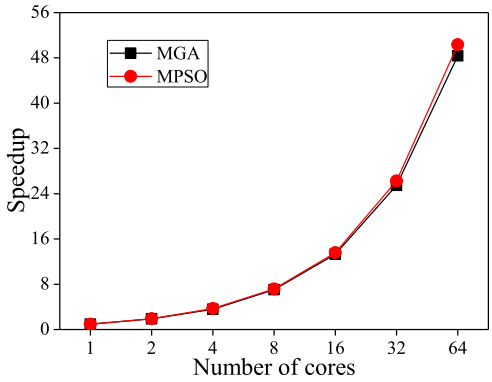

Fig. 7. Speedup of MEC algorithms with different number of CPU cores.

results show that the speedup of MEC becomes more evident as the more CPU cores are used, no matter for MGA or MPSO. Therefore, the advantages of MEC algorithms will be more significant if one can deploy the algorithms in rich-resources computational environments.

D. Discussion

The above contents have shown the great superiority and advantage of the MEC. However, to be honest, the matrix-based idea for EC algorithms is still a new methodology and there are some issues that need further researches and studies in the future.

As shown in the Fig. 3, when an existing EC algorithm is extended to its MEC version, there are two key issues that should be addressed. One is the matrix-based solution representation and the other is the matrix-based population reproduction. In fact, the matrix-based solution representation of different MEC algorithms is similar, which has been clearly described in Section III-A where MGA is used as an example. Similarly, as described in Section IV-A for the matrix-based solution representation in MPSO, the implementation is the same as that in MGA. While for the matrix-based population reproduction, the implementations of different EC algorithms may be somewhat different, which may result in some difficulties in the implementation.

This paper has fully described the implementations of MGA and MPSO in detail because they are the two typical evolutionary algorithm and swarm intelligence algorithm respectively. Moreover, in the Supplemental Material, this paper has also provided the implementations of matrix-based DE (MDE) and matrix-based EDA (MEDA) detailly because they are also two widely-used EC algorithms for optimization problems. The implementations detail of these four algorithms can serve as helpful references when extending other EC algorithms into matrix-based versions. However, it is difficult to give all the implementations of all different EC algorithms in this single paper. Therefore, this paper has put these issues in the future works and hope more and more related works can appear in the near future.

VII. Conclusion

The MEC proposed in this paper is an entirely new and groundbreaking perspective to extend traditional EC by introducing the matrix-based operation. The MEC uses a matrix to represent the whole population of the algorithm, where a row stands for an individual and a column stands for a dimension (variable). In this way, this paper has successfully developed the MGA and MPSO. Their solution representations are similar and their evolutionary operators, e.g., the selection, crossover, and mutation operators in MGA, and the velocity update, position update, and personal best position update in MPSO, have been fully designed and described based on the matrix representations. Moreover, two other EC algorithms named DE and EDA are extended to their matrix-based versions, resulting in the MDE and MEDA, respectively, and are detailed in the Supplemental Material. The time complexity analyses on both MGA and MPSO also show that MEC has greatly reduced the computational time of traditional EC algorithms, especially on very large-scale optimization problems. Therefore, the MEC can be very promising in the computational time when solving complex optimization problems in big data environment. The future work will on the one hand develop MEC algorithms based on other typical EC algorithms and on the other hand try to apply MEC algorithms to various kinds of complex problems in real-world applications. For other MEC algorithms, apart from the MGA, MPSO, MDE, and MEDA detailed in this paper, the implementation of matrix-based ACO and others are worthy studied. Moreover, when based on special EC algorithms for special optimization problems, the special matrix-based evolutionary operators in dealing with large-scale [48], dynamic [49], multimodal [50], multi-objective [51], many-objective [52], and constrained issues [53] are also worthy studied.

References

[1] J. C. Bezdek, "(Computational) intelligence: What's in a name?," *IEEE Syst., Man, Cybern. Mag.*, vol. 2, no. 2, pp. 4–14, Apr. 2016.
[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[3] M. H. S. Segler, M. Preuss, and M. P. Waller, "Planning chemical syntheses with deep neural networks and symbolic AI," *Nature*, vol. 555, pp. 604–610, 2018.
[4] Q. Chen *et al.*, "A survey on an emerging area: Deep learning for smart city data," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 3, no. 5, pp. 392–410, Oct. 2019.
[5] Y. Ong and A. Gupta, "AIR5: Five pillars of artificial intelligence research," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 3, no. 5, pp. 411–415, Oct. 2019.
[6] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
[7] D. Silver *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.
[8] Y. Sun, G. G. Yen, and Y. Zhang, "Evolving unsupervised deep neural networks for learning meaningful representations," *IEEE Trans. Evol. Comput.*, vol. 23, no. 1, pp. 89–103, Feb. 2019.
[9] Y. Guo, J. Y. Li, and Z. H. Zhan, "Efficient hyperparameter optimization for convolution neural networks in deep learning: A distributed particle swarm optimization approach," *Cybern. Syst.*, to be published, doi: 10.1080/01969722.2020.1827797.
[10] S. M. Nekooei and G. Chen, "Cooperative coevolution design of multilevel fuzzy logic controllers for media access control in wireless body area networks," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 4, no. 3, pp. 336–350, Jun. 2020.

[11] Z. H. Zhan, J. Zhang, Y. Li, and H. S. H. Chung, "Adaptive particle swarm optimization," *IEEE Trans. Syst. Man, Cybern. B*, vol. 39, no. 6, pp. 1362–1381, Dec. 2009.

[12] J. Zhang *et al.*, "Evolutionary computation meets machine learning: A survey," *IEEE Comput. Intell. Mag.*, vol. 6, no. 4, pp. 68–75, Nov. 2011.

[13] X. Zhang, K. J. Du, Z. H. Zhan, S. Kwong, T. L. Gu, and J. Zhang, "Cooperative co-evolutionary bare-bones particle swarm optimization with function independent decomposition for large-scale supply chain network design with uncertainties," *IEEE Trans. Cybern.*, vol. 50, no. 10, pp. 4454–4468, Oct. 2020.

[14] J. Ding, C. Yang, Q. Xiao, T. Chai, and Y. Jin, "Dynamic evolutionary multiobjective optimization for raw ore allocation in mineral processing," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 3, no. 1, pp. 36–48, Feb. 2019.

[15] S. Zhou, Z. H. Zhan, Z. Chen, S. Kwong, and J. Zhang, "A multi-objective ant colony system algorithm for airline crew rostering problem with fairness and satisfaction," *IEEE Trans. Intell. Transp. Syst.*, to be published, doi: 10.1109/TITS.2020.2994779.

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

[17] X. Xu, S. Fu, W. Li, F. Dai, H. Gao, and V. Chang, "Multi-objective data placement for workflow management in cloud infrastructure using NSGA-II," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 4, no. 5, pp. 605–615, Oct. 2020.

[18] 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.

[19] J. Y. Li, Z. H. Zhan, H. Wang, and J. Zhang, "Data-driven evolutionary algorithm with perturbation-based ensemble surrogates," *IEEE Trans. Cybern.*, to be published, doi: 10.1109/TCYB.2020.3008280.

[20] Z. J. Wang *et al.*, "Automatic niching differential evolution with contour prediction approach for multimodal optimization problems," *IEEE Trans. Evol. Comput.*, vol. 24, no. 1, pp. 114–128, Feb. 2020.

[21] Z. G. Chen, Y. Lin, Y. Gong, Z. H. Zhan, and J. Zhang, "Maximizing lifetime of range-adjustable wireless sensor networks: A neighborhood-based estimation of distribution algorithm," *IEEE Trans. Cybern.*, to be published, doi: 10.1109/TCYB.2020.2977858.

[22] J. Kennedy, R. C. Eberhart, and Y. H. Shi, *Swarm Intelligence*. San Mateo, CA: Morgan Kaufmann, 2001.

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

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

[25] V. Šešum-Čavić, E. Kühn, and L. Fleischhacker, "Efficient search and lookup in unstructured P2P overlay networks inspired by swarm intelligence," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 4, no. 3, pp. 351–368, Jun. 2020.

[26] M. O. Akinsolu, B. Liu, V. Grout, P. I. Lazaridis, M. E. Mognaschi, and P. D. Barba, "A parallel surrogate model assisted evolutionary algorithm for electromagnetic design optimization," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 3, no. 2, pp. 93–105, Apr. 2019.

[27] Z. Zhou, B. Wang, Y. Guo, and Y. Zhang, "Blockchain and computational intelligence inspired incentive-compatible demand response in internet of electric vehicles," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 3, no. 3, pp. 205–216, Jun. 2019.

[28] T. Wei, J. Zhong, and J. Zhang, "An energy-efficient partition-based framework with continuous ant colony optimization for target tracking in mobile sensor networks," *IEEE Trans. Emerg. Top. Comput. Intell.*, to be published, doi: 10.1109/TETCI.2019.2940978.

[29] R. Wang and W. Ji, "Computational intelligence for information security: A survey," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 4, no. 5, pp. 616–629, Oct. 2020.

[30] P. Huang, Y. Wang, K. Wang, and K. Yang, "Differential evolution with a variable population size for deployment optimization in a UAV-assisted IoT data collection system," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 4, no. 3, pp. 324–335, Jun. 2020.

[31] M. Asim, Y. Wang, K. Wang, and P. Huang, "A review on computational intelligence techniques in cloud and edge computing," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 4, no. 6, pp. 742–763, Dec. 2020.

[32] Z. H. Zhan *et al.*, "Cloudde: A heterogeneous differential evolution algorithm and its distributed cloud version," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 704–716, Mar. 2017.

[33] Z. J. Wang, Z. H. Zhan, and J. Zhang, "Solving the energy efficient coverage problem in wireless sensor networks: A distributed genetic algorithm approach with hierarchical fitness evaluation," *Energies*, vol. 11, no. 12, pp. 1–14, Dec. 2018.

[34] 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.

[35] 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.*, to be published, doi: 10.1109/TCYB.2020.2977956.

[36] Y. F. Ge *et al.*, "Distributed memetic algorithm for outsourced database fragmentation," *IEEE Trans. Cybern.*, to be published, doi: 10.1109/TCYB.2020.3027962.

[37] Y. J. Gong *et al.*, "Distributed evolutionary algorithms and their models: A survey of the state-of-the-art," *Appl. Soft Comput. J.*, vol. 34, no. 2013, pp. 286–300, 2015.

[38] Z. J. Wang *et al.*, "Dynamic group learning distributed particle swarm optimization for large-scale optimization and its application in cloud workflow scheduling," *IEEE Trans. Cybern.*, vol. 50, no. 6, pp. 2715–2729, Jun. 2020.

[39] J. Y. Li, Z. H. Zhang, R. D. Liu, C. Wang, S. Kwong, and J. Zhang, "Generation-level parallelism for evolutionary computation: A pipeline-based parallel particle swarm optimization," *IEEE Trans. Cybern.*, to be published, doi: 10.1109/TCYB.2020.3028070.

[40] J. S. Pan, Z. Meng, H. Xu, and X. Li, "A matrix-based implementation of DE algorithm: The compensation and deficiency," in *Proc. Int. Conf. Ind., Eng. Other Appl. Appl. Intell. Syst.*, 2017, pp. 72–81.

[41] L. García, J. Cuenca, and D. Giménez, "On optimization techniques for the matrix multiplication on hybrid {CPU+GPU} platforms," *Ann. Multicore GPU Prog.*, vol. 1, no. 1, pp. 10–18, 2014.

[42] G. Bernabé, J. Cuenca, L. P. García, and D. Giménez, "Tuning basic linear algebra routines for hybrid CPU+GPU platforms," *Procedia Comput. Sci.*, vol. 29, pp. 30–39, 2014.

[43] G. Bernabé, J. Cuenca, L. P. García, and D. Giménez, "Auto-tuning techniques for linear algebra routines on hybrid platforms," *J. Comput. Sci.*, vol. 10, pp. 299–310, 2015.

[44] P. Alonso, R. Reddy, and A. Lastovetsky, "Experimental study of six different implementations of parallel matrix multiplication on heterogeneous computational clusters of multicore processors," in *Proc. Euromicro Conf. Parallel, Distrib. Netw.-Based Process.*, 2010, pp. 263–270.

[45] J. Choi, "A new parallel matrix multiplication algorithm on distributed-memory concurrent computers," *Concurrency Pract. Exp.*, vol. 10, no. 8, pp. 655–670, 1998.

[46] G. C. Fox, S. W. Otto, and A. J. G. Hey, "A matrix algorithms on a hypercube I: Matrix multiplication," *Parallel Comput*, vol. 4, no. 1, pp. 17–31, Feb. 1987.

[47] S. Ubéda, "Pyramidal thinning algorithm for SIMD parallel machines," *Pattern Recognit*, vol. 28, no. 12, pp. 1993–2000, 1995.

[48] J. R. Jian, Z. H. Zhan, and J. Zhang, "Large-scale evolutionary optimization: A survey and experimental comparative study," *Int. J. Mach. Learn. Cybern.*, vol. 11, no. 3, pp. 729–745, Mar. 2020.

[49] X. F. Liu *et al.*, "Neural network-based information transfer for dynamic optimization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 5, pp. 1557–1570, May 2020.

[50] 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.

[51] Z. H. Zhan, J. Li, J. Cao, J. Zhang, H. Chung, and Y. H. Shi, "Multiple populations for multiple objectives: A coevolutionary technique for solving multiobjective optimization problems," *IEEE Trans. Cybern.*, vol. 43, no. 2, pp. 445–463, Apr. 2013.

[52] 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.

[53] W. Xu, J. Xu, D. He, and K. C. Tan, "An evolutionary constraint-handling technique for parametric optimization of a cancer immunotherapy model," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 3, no. 2, pp. 151–162, Apr. 2019.

Zhi-Hui Zhan (Senior Member, IEEE) received the bachelor's and Ph.D. degrees in computer science from 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 algorithms, swarm intelligence algorithms, deep learning, and their applications in real-world problems, and in environments of cloud computing and big data. Dr. Zhan was the recipient of the IEEE Computational Intelligence Society, Outstanding Ph.D. Dissertation, and the China Computer Federation (CCF) Outstanding Ph.D. Dissertation. He was also the recipient of the Outstanding Youth Science Foundation from the National Natural Science Foundations of China in 2018 and the Wu Wen-Jun Artificial Intelligence Excellent Youth from the Chinese Association for Artificial Intelligence in 2017. He is listed as one of the most cited Chinese researchers in computer science. He is currently an Associate Editor for the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the *Neurocomputing*, and the *International Journal of Swarm Intelligence Research*.

Jun Zhang (Fellow, IEEE) received the Ph.D. degree from the City University of Hong Kong, Kowloon, Hong Kong, in 2002. He is currently a Korea Brain Pool Fellow Professor with Hanyang University, South Korea, and a Visiting Scholar with Victoria University, Australia. He has authored or coauthored more than 300 technical papers in his research fields, which include computational intelligence algorithms and applications. Dr. Zhang was the recipient of the Changjiang Chair Professor from the Ministry of Education, China, in 2013 and the China National Funds for Distinguished Young Scientists from the National Natural Science Foundation of China in 2011. He is currently an Associate Editor for the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and the IEEE TRANSACTIONS ON CYBERNETICS.

Ying Lin (Member, IEEE) received the Ph.D. degree in 2012 in computer science from Sun Yat-sen University, Guangzhou, China, where she is currently an Associate Professor with the Department of Psychology. Her main research interests include computational intelligence and its applications in psychometrics and neuroimaging.

Jian-Yu Li (Student Member, IEEE) received the bachelor's degree in 2018 in computer science and technology from the South China University of Technology, Guangzhou, China, where he is currently working toward the Ph.D. degree in computer science and technology with the School of Computer Science and Engineering. His research interests mainly include computational intelligence, data-driven optimization, machine learning, and their applications in real-world problems, and in environments of distributed computing and big data.

Ting Huang (Student Member, IEEE) is currently working toward the Ph.D. degree from the School of Computer Science and Engineering, South China University of Technology, China. Her current research interests include evolutionary computation, swarm intelligence, multisolution optimization, and their real-world applications.

Xiao-Qi Guo (Student Member, IEEE) received the bachelor's degree in 2018 in computer science and technology from the South China University of Technology, Guangzhou, China, where she is currently working toward the Ph.D. degree. Her current research interests include evolutionary computation and their applications on expensive optimization problems or in distributed networks.

Feng-Feng Wei (Student member, IEEE) received the bachelor's degree in 2019 in computer science and technology from the South China University of Technology, Guangzhou, China, where she is currently working toward the Ph.D. degree in computer science and technology. Her current research interests include data-driven evolutionary computation, evolutionary constrained optimization, and cooperative coevolutionary algorithms.

Sam Kwong (Fellow, IEEE) received the B.S. degree in electrical engineering from the State University of New York at Buffalo, Buffalo, NY, USA, in 1983, the M.S. degree in electrical engineering from the University of Waterloo, Waterloo, ON, Canada, in 1985, and the Ph.D. degree from the University of Hagen, Hagen, Germany, in 1996. From 1985 to 1987, he was a Diagnostic Engineer with Control Data Canada, where he designed the diagnostic software to detect the manufactured faults of the VLSI chips in the cyber 430 machine. Then, he joined the Bell Northern Research, Canada, as a Member of the Scientific Staff. In 1990, he joined as a Lecturer with the Department of Electronics Engineering, City University of Hong Kong, Hong Kong. He is currently the Chair Professor with the Department of Computer Science. His research interests include pattern recognition, evolutionary computations, and video analytics. Prof. Kwong was elevated to an IEEE Fellow for his contributions to optimization techniques for cybernetics and video coding in 2014. He is the Vice President of the IEEE Systems, Man, and Cybernetics. He was also appointed as an IEEE Distinguished Lecturer of the IEEE SMC Society in March 2017. He is currently an Associate Editor for the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION.

Xin-Yi Zhang (Student Member, IEEE) received the bachelor's degree in 2018 in computer science and technology from the South China University of Technology, Guangzhou, China, where she is currently working toward the master's degree in computer science and technology. Her current research interests include data-driven evolutionary computation, fuzzy system, and combinational optimization in intelligent transportation.

Rui You (Student Member, IEEE) received the bachelor's degree in 2018 in computer science and technology from the South China University of Technology, Guangzhou, China, where he is currently working toward the master's degree in computer science and technology. His current research interests include the estimation of distribution algorithms, data-driven optimization, and their applications.