# Adaptive Granularity Learning Distributed Particle Swarm Optimization for Large-Scale Optimization

AQ1 Zi-Jia Wang, *Student Member, IEEE*, Zhi-Hui Zhan, *Senior Member, IEEE*, Sam Kwong, *Fellow, IEEE*, Hu Jin, *Senior Member, IEEE*, and Jun Zhang, *Fellow, IEEE*

***Abstract*—Large-scale optimization has become a significant and challenging research topic in the evolutionary computation (EC) community. Although many improved EC algorithms have been proposed for large-scale optimization, the slow convergence in the huge search space and the trap into local optima among massive suboptima are still the challenges. Targeted to these two issues, this article proposes an adaptive granularity learning distributed particle swarm optimization (AGLDPSO) with the help of machine-learning techniques, including clustering analysis based on locality-sensitive hashing (LSH) and adaptive granularity control based on logistic regression (LR). In AGLDPSO, a master–slave multisubpopulation distributed model is adopted, where the entire population is divided into multiple subpopulations, and these subpopulations are co-evolved. Compared with other large-scale optimization algorithms with single population evolution or centralized mechanism, the multisubpopulation distributed co-evolution mechanism will fully exchange the evolutionary information among different subpopulations to further enhance the population diversity. Furthermore, we propose an adaptive granularity learning strategy (AGLS) based on LSH and LR. The AGLS is helpful to determine an appropriate subpopulation size to control the learning granularity of the distributed subpopulations in different evolutionary states to balance the exploration ability for escaping from massive suboptima and the exploitation ability for converging in the huge search space. The experimental results show that AGLDPSO performs better than or at least comparable with some other state-of-the-art large-scale optimization algorithms, even the winner of the competition on large-scale optimization, on all the 35 benchmark functions from both IEEE Congress on Evolutionary Computation (IEEE CEC2010) and IEEE CEC2013 large-scale optimization test suites.**

***Index Terms*—Adaptive granularity learning distributed particle swarm optimization (AGLDPSO), large-scale optimization, locality-sensitive hashing (LSH), logistic regression (LR), master–slave multisubpopulation distributed.**

AQ2 Manuscript received August 20, 2019; revised December 10, 2019 and February 18, 2020; accepted February 27, 2020. This work was supported in part by the Outstanding Youth Science Foundation under Grant 61822602, in part by the National Key Research and Development Program of China under Grant 2019YFB2102100, in part by the National Natural Science Foundations of China under Grant 61772207 and Grant 61873097, in part by the Guangdong Natural Science Foundation Research Team under Grant 2018B030312003, in part by the Ministry of Science and ICT through the National Research Foundation of Korea under Grant NRF-2019H1D3A2A01101977, and in part by the Hong Kong GRF-RGC General Research Fund under Grant 9042489 and Grant CityU 11206317. This article was recommended by Associate Editor Y. Yang. *(Corresponding authors: Zhi-Hui Zhan; Jun Zhang.)*

Zi-Jia Wang is with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China.

Zhi-Hui Zhan is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, and also with the Guangdong Provincial Key Laboratory of Computational Intelligence and Cyberspace Information, South China University of Technology, Guangzhou 510006, China (e-mail: zhanapollo@163.com).

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

Hu Jin and Jun Zhang are with the Department of Division of Electrical Engineering, Hanyang University, Ansan 15588, South Korea (e-mail: junzhang@ieee.org).

This article has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the author.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCYB.2020.2977956

## I. Introduction

EVOLUTIONARY computation (EC) algorithms, including evolutionary algorithms (EAs) and swarm intelligence algorithms (SIs) [1]–[9], such as genetic algorithm (GA) [10], [11]; differential evolution (DE) [12]–[15]; particle swam optimization (PSO) [16]–[18]; and ant colony optimization (ACO) [19]–[21], have been widely studied and applied in many real-world optimization problems. However, with the increasing scale of problems, the traditional EC algorithms lose their effectiveness and advantages rapidly when the dimension of problem increases, which is so-called "curse of dimensionality" [22]–[26]. The main reason for this phenomenon can be due to two points. On the one hand, the search space drastically and exponentially increases with the growing dimensionality, traditional EC algorithms have to find the optimal solution in the huge search space, causing the slow convergence. On the other hand, there are massive local optimal regions in the search space, traditional EC algorithms have to further improve the population diversity so as to escape from the local optima in such a complex environment.

To tackle with the large-scale optimization problems (often more than 500 dimensions), a common method is utilizing a cooperative coevolution (CC) framework in EC algorithms to decompose the high-dimensional problem into several mid/lower dimensional subproblems and to solve each subproblem separately. The CC framework was first proposed by Potter and Jong in 1994 [27]. Due to its effectiveness, researchers have developed many variants by introducing the CC framework into different EC algorithms (CCECs), including CCGA [27], [28]; CCPSO [29], [30]; and DECC [31]–[34]. Even though the CCECs are intensively studied and have achieved a considerable success, but undeniably, they also have some limitations. First, the decomposition strategy is the most crucial component

for CCECs. Therefore, the performance of CCECs is highly sensitive to the decomposition strategies. Second, a considerable number of fitness evaluations (FEs) are required since we have to solve each subproblem separately, especially when the dimension of the problem is large. Also, in order to improve the accuracy of decomposition, additional FEs are often needed to detect the dependency among variables [33]–[35]. Third, CCECs are only effective and helpful for separable problems. When dealing with the partially separable problems or nonseparable problems, the performance of CCECs deteriorates greatly.

Taking the above limitations of CCECs into account, researchers proposed some novel search strategies for EC algorithms to tackle with all dimensions as a whole [36], [37]. Moreover, studies have shown that coevolutionary [38], [39] and distributed [40], [41] mechanisms with multiple populations can fully improve optimization efficiency.

However, when using the multipopulation coevolutionary mechanism, the size of the population (i.e., the granularity) is also necessary and needed to be set appropriately. Large population (coarse granularity) and small population (fine-granularity) may be suitable for different evolutionary states in different problems. Many current multipopulation coevolutionary algorithms only use the simple dynamic-changed mechanism. For example, the dynamic multiswarm PSO (DMS-L-PSO) is proposed in [42], where the swarm size is randomly changed in every certain generation. However, random change is difficult for providing the appropriate population size and lacks the adaptive granularity control. If we can further introduce the adaptive granularity control and find the appropriate population size to meet the search requirement of the different evolutionary states in different problems, the search process will be more effective.

Machine learning (ML) is one of the most promising research areas in artificial intelligence, which has become a powerful tool in a wide range of applications [43]. Since EC algorithms have stored ample data about the search space, problem features, and population information during the iterative search process, the ML technique is helpful in analyzing these data to further enhance the search performance. In this way, useful information can be extracted to analyze the evolutionary state and to achieve the adaptive granularity control. Therefore, to further improve the searching ability and achieve an adaptive algorithm, this article develops a novel adaptive granularity learning distributed PSO (AGLDPSO) with the help of ML techniques, including clustering analysis based on locality-sensitive hashing (LSH) and adaptive granularity control based on logistic regression (LR). More specifically, the novelties and advantages of AGLDPSO contain the following two aspects.

1) The master–slave multisubpopulation distributed model is adopted in AGLDPSO, where the entire population is randomly divided into multiple subpopulations and these subpopulations are co-evolved. Compared with other large-scale optimization algorithms with single population evolution or centralized mechanism, the multisubpopulation distributed co-evolution mechanism will fully exchange the evolutionary information among different subpopulations to further enhance the population diversity.
2) An adaptive granularity learning strategy (AGLS) is further proposed based on LSH and LR. The AGLS is helpful to determine an appropriate subpopulation size to control the learning granularity of the distributed subpopulations in different evolutionary states to balance the exploration ability for escaping from massive suboptima and the exploitation ability for converging in the huge search space.

Experiments are conducted on all the 35 benchmark functions from both IEEE Congress on Evolutionary Computation (IEEE CEC2010) and IEEE CEC2013 large-scale optimization test suites. The results show that AGLDPSO is better than, or at least comparable to some other state-of-the-art large-scale optimization algorithms, even the winner of the competition on large-scale optimization, showing the effectiveness and superiority of our AGLDPSO algorithm.

The remainder of this article is organized as follows. Section II reviews the traditional PSO algorithm and some PSO variants for large-scale optimization. Section III presents the AGLDPSO algorithm in detail. The experimental results from both IEEE CEC2010 and IEEE CEC2013 large-scale optimization test suites between AGLDPSO and some other state-of-the-art large-scale optimization algorithms are shown in Section IV. Finally, the conclusion will be drawn in Section V.

## II. Related Work

### A. PSO

In PSO [44], the member of the population is called particle. Each particle $P_i$ has two vectors. The vector $\boldsymbol{V}_i = [v_i^1, v_i^2, \ldots, v_i^D]$ means the velocity of $P_i$, while the vector $\boldsymbol{X}_i = [x_i^1, x_i^2, \ldots, x_i^D]$ means the position of $P_i$, where $D$ stands for the dimensions of the search space. Moreover, each particle $P_i$ has its own historical best position, called personal best $\boldsymbol{pbest}_i = [\text{pbest}_i^1, \text{pbest}_i^2, \ldots, \text{pbest}_i^D]$. The best one of all the $\boldsymbol{pbest}_i$ is treated as the globally best of the entire population, called $\boldsymbol{gbest} = [\text{gbest}^1, \text{gbest}^2, \ldots, \text{gbest}^D]$. In every generation, each particle $P_i$ updates its velocity and position based on its own $\boldsymbol{pbest}_i$ and the $\boldsymbol{gbest}$ of the entire population. The velocity $\boldsymbol{V}_i$ and position $\boldsymbol{X}_i$ of each particle are updated according to the following formulas:

$$v_i^d = \omega \times v_i^d + c_1 \times r_{1i}^d \times \left(\text{pbest}_i^d - x_i^d\right) + c_2 \times r_{2i}^d \times \left(\text{gbest}^d - x_i^d\right) \quad (1)$$

$$x_i^d = x_i^d + v_i^d \quad (2)$$

where $\omega$ is the inertia weight to balance the global and local search abilities. $c_1$ and $c_2$ are the acceleration coefficients, where $c_1$ pulls the particle to its own $\boldsymbol{pbest}$, ensuring the diversity of the population; while $c_2$ pushes the particle to the current $\boldsymbol{gbest}$, ensuring the speed of convergence. $r_{1i}^d$ and $r_{2i}^d$ are two uniformly distributed random numbers within [0, 1]. A particle's velocity and position on each dimension are clamped in $[-V_{\max}^d, V_{\max}^d]$ and $[X_{\min}^d, X_{\max}^d]$, respectively.

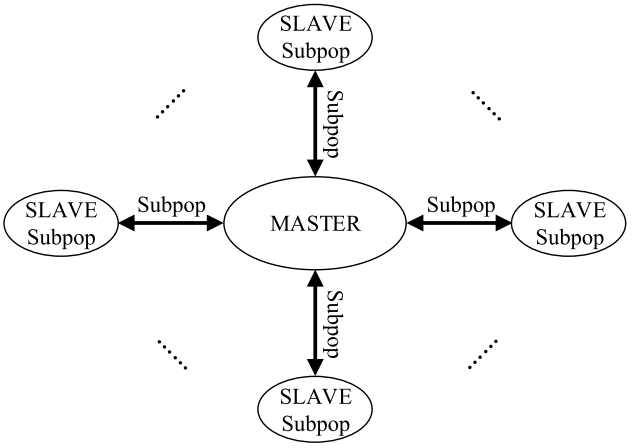


Fig. 1. Master–slave multisubpopulation distributed model.

### B. PSO Variants for Large-Scale Optimization

The traditional PSO algorithm or its variants may be promising in solving small-scale or low-dimensional problems [45]–[47]. However, when the scale or dimension of the problem increases, PSO and its variants may lose their effectiveness and feasibility. The huge search space and the exponentially increasing local optima are two of the most important challenges.

When using PSO to tackle with the large-scale optimization problems, a common method is combining the CC framework with PSO, called CCPSO [29], [30]. For instance, in CCPSO-$S_K$ [29], it randomly divides the entire problem into $K$ subcomponents, while each subcomponent contains $D/K$ dimensions. After that, PSO is applied to optimize each subcomponent. Meanwhile, its variant CCPSO-$H_K$ is also proposed [29], which combines PSO and CCPSO-$S_K$, where PSO and CCPSO-$S_K$ are evolved alternately. In CCPSO2 [30], Li and Yao developed a new scheme to dynamically determine the coevolving subcomponent size by randomly choosing from a predefined size pool. However, different subcomponent sizes are suitable for different problems. Using fixed subcomponent size or using dynamic subcomponent size may be difficult to provide the suitable subcomponent size for different problems.

Although CCPSOs have achieved a considerable success, they also have some limitations. First, the performance of CCPSOs is highly sensitive to the decomposition strategies. Second, since we have to solve each subproblem separately, a considerable number of FEs are required, especially when the dimension of the problem is large. Third, the current decomposition strategies [33]–[35] also need extra FEs to detect the dependency among variables. In addition, CCPSOs are only effective and helpful for separable problems. When dealing with the partially separable problems or nonseparable problems, their performance deteriorates greatly.

Taking the above limitations of CCPSOs into consideration, some researchers design the novel search strategies for PSO to effectively explore in the huge search space and avoid local optima. In SL-PSO, proposed by Cheng and Jin [36], each particle learns from other better particles in the entire population after the fitness sorting. Besides, they also propose CSO [37], where two particles are randomly selected from the population for competing, then the loser learns from the winner. Different from the traditional PSO, both SL-PSO and CSO use some other better particles and the mean position of the entire population to guide the evolution rather than using only the personal best ***pbest*** and ***gbest***. Dynamic segment-based predominant learning swarm optimizer (DSPLSO) [48], proposed by Yang *et al.*, follows the framework of CSO, where the worse particles will learn from different better particles through the segment-based predominant learning strategy. Moreover, they also develop dynamic level-based learning swarm optimizer (DLLSO) [49], which first separates particles into a number of levels, then each particle guides by two particles from different higher levels. These two methods use a different dynamic mechanism based on the *softmax* function and probabilistic scheme rather than the simple random mechanism. In DMS-L-PSO [42], it generates several subswarms randomly in every certain generation for exchanging information among different subswarms to improve the population diversity. Meanwhile, it utilizes a quasi-Newton-based local search strategy to further refine the solutions.

Although many improved PSOs have been proposed to deal with the large-scale optimization problems, the slow convergence in the huge search space and the trap into local optima among massive suboptima are still the main challenges in large-scale optimization. Therefore, AGLDPSO is proposed to relieve these issues.

## III. AGLDPSO

### A. Master–Slave Multisubpopulation Distributed Framework

The master–slave multisubpopulation distributed framework is illustrated in Fig. 1, where the master node dominates multiple slave nodes in the parallel hardware.

During the evolution process, the master randomly divides the entire population into $N/M$ subpopulations of the same size and sends each subpopulation to its corresponding slave, where $N$ is the population size and $M$ is the size of the subpopulation. Note that if $N\%M \neq 0$, the last subpopulation will have $M + N\%M$ particles (where % stands for the modulo operation). Then, different subpopulations are co-evolved concurrently on their slave nodes. After the evolution, each slave sends the updated subpopulation back to the master. Different from the traditional master–slave distributed framework, the number of slaves is adaptively changed since the size (learning granularity) of subpopulation is adaptively controlled according to the LSH and LR (shown in Section III-C).

### B. Velocity and Position Update

Before we introduce AGLS, we first describe the velocity and position update operators in AGLDPSO. The formulas of updating the velocity and position are shown as (3) and (4), which are similar to (1) and (2), but with the following two differences:

$$v_w^d = \omega^d \times v_w^d + c_1 \times r_1^d \times \left(\text{sbest}^d - x_w^d\right) + c_2 \times r_2^d \times \left(\text{gbest}^d - x_w^d\right) \quad (3)$$

$$x_w^d = x_w^d + v_w^d. \quad (4)$$

1) *Strategy Difference:* After the population partition, only the worst particle $P_w$ in each subpopulation is updated by learning from the best particle in the current subpopulation (called subpopulation best ***sbest***) and the best particle from the entire population (called global best ***gbest***), while other particles in the subpopulation will enter to the next generation directly. This velocity update strategy only updates the worst particle in each subpopulation, being helpful to save more FEs to prolong the evolutionary process to further increase the solution accuracy.
2) *Parameter Difference:* The inertia weight $\omega$ in PSO is often set as linearly decreasing from 0.9 to 0.4, while herein it is replaced by the random number in AGLDPSO. The random mechanism is beneficial to maintain the learning diversity and the population diversity. Meanwhile, acceleration coefficients $c_1$ and $c_2$ that control the convergence speed to the ***sbest*** and ***gbest*** are often set as 2.0 in the traditional PSO, while in AGLDPSO, both $c_1$ and $c_2$ are set smaller than that in traditional PSO to avoid premature convergence, which is replaced by 1.0 and 0.1, respectively. The new setting parameters $\omega$, $c_1$, and $c_2$ are beneficial for AGLDPSO to maintain the population diversity, which can effectively avoid being trapped in the local optima.

We have also compared the experimental results between AGLDPSO and the traditional PSO on the IEEE CEC2010 large-scale optimization test suite, the results listed in Table S.I in the supplementary file fully show the effectiveness and superiority of our velocity updating strategy and parameter setting in AGLDPSO.

### C. AGLS

The subpopulation size $M$ is important for AGLDPSO because it affects the learning granularity for the particle in each subpopulation and further affects the diversity for exploration and the convergence for exploitation, which is needed to be set appropriately. For a given population size, if the subpopulation size $M$ is large, it means the coarse granularity because the number of particles in each subpopulation is large. Therefore, the ***sbest*** is the best particle from a large subpopulation, so that $P_w$ is learning from a large neighborhood and therefore, is helpful for accelerating the convergence speed. In contrast, if $M$ is small, it means the fine granularity to learn from small subpopulation. Therefore, the number of subpopulations is large, so that many ***sbest*** can be learned to increase the diversity. Although the subpopulation size $M$ has a significant influence on the algorithm performance, how to precisely determine a suitable $M$ in different evolutionary states for a given problem is still hard to solve.

In the literature, researchers have found that the exploration and exploitation abilities of EC algorithms can be adaptively adjusted according to the evolutionary state [50]–[52]. In this sense, clustering analysis has been widely used to estimate the evolutionary state and the parameters of EC algorithms can be adjusted adaptively. However, when the dimension of the problem increases, the traditional clustering analysis approach will be very time consuming and ineffective.

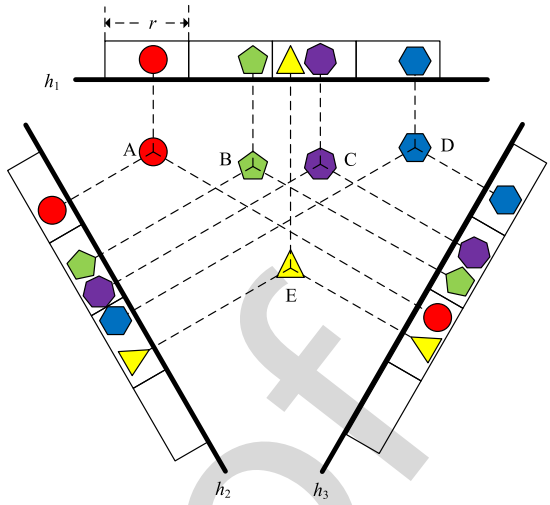


Fig. 2. Example of LSH in a 2-D Euclidean space.

In order to design a more lightweight and efficient evolutionary state estimation approach, the LSH approximation method is adopted to ensure the estimation quality and to overcome the time bottleneck [53]. After the clustering based on LSH, the learning granularity $M$ is adaptively controlled based on the LR. Therefore, we first introduce a novel clustering analysis method based on LSH and then introduce the adaptive granularity control based on LR.

*1) Clustering Analysis Based on LSH:* LSH has been successfully applied in EC algorithms for solving multimodal optimization problems [54]. Here, we modified and applied this method into AGLDPSO for tackling with the large-scale optimization problems.

The basic idea of LSH is that two neighboring individuals in the original space have a large probability to be adjacent in the new space by the same mapping. Thus, the distance of individuals in the high-dimensional space can be calculated approximately and quickly by mapping these individuals into the low-dimensional space. A simple example of LSH in a 2-D Euclidean space is shown in Fig. 2. Each hash function is defined as a projected line in the space, and there are infinite hash functions. Then, each projected line is divided into several equal segments with size $r$. Each segment represents a "bucket" for storing the individuals. We randomly generate a hash function (projected line) and project all the individuals onto the line. Then, two close individuals will be hashed into the same bucket with a large probability. This property can be used to simplify the distance computing in clustering. Suppose that there are five individuals (A, B, C, D, and E) in the space. The similar individuals B and C can be very likely hashed into the same bucket using the hash functions $h_2$ and $h_3$.

Now, we extend LSH into any dimensional problems. In a $D$-dimensional problem, each individual $\boldsymbol{x}_i$ can be expressed as a vector $(x_{i,1}, x_{i,2}, \ldots, x_{i,D})$. In every generation, we first randomly generate a $D$-dimensional vector $\boldsymbol{O}$ within the search space and calculate the dot product $(\boldsymbol{x}_i \boldsymbol{O}^T)$. The dot product projects this individual (vector) onto a line. Therefore, if the individuals $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are close ($\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ is small), the distance between their projections $(\boldsymbol{x}_1 - \boldsymbol{x}_2)\, \boldsymbol{O}^T$ is very likely small and

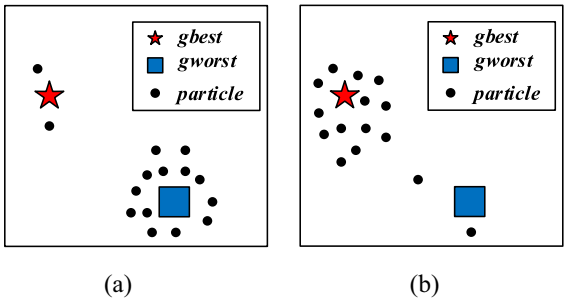


Fig. 3. Illustration of the evolutionary state based on the population distribution. (a) Exploration. (b) Exploitation.

they are in the same bucket with a large probability. In contrast, if they are far from each other ($\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ is large), the distance between their projections $(\boldsymbol{x}_1 - \boldsymbol{x}_2)\,\boldsymbol{O}^T$ is very likely large and they are in the same bucket with a small probability.

When all these individuals have been projected, we record the maximal and minimal coordinates ($h_{\max}$ and $h_{\min}$) of all the projected points. Then, the bucket size $r$ is calculated as

$$r = \frac{h_{\max} - h_{\min}}{nb} \tag{5}$$

where $nb$ is the number of buckets. In our method, $nb$ is set as $0.1 \times N$. Next, we divide the hash line into $nb$ equal-width segments with size $r$, the hash values of these individual are the segments they are projected to. The hash function is defined as

$$h(\boldsymbol{x}_i) = \left\lfloor \frac{\boldsymbol{x}_i O^T + b}{r} \right\rfloor \tag{6}$$

where $b$ is a real number called shifted projection, which is randomly generated within the interval $[0, r]$. Assume that the distance between the projections of two individuals is $d$, if $d$ is larger than $r$, then the two individuals will be obviously in two different buckets. If $d$ is smaller than $r$, then the probability of the two individuals being in the same bucket will be $1 - d/r$. When all the individuals have been projected, the individuals in the same bucket are assigned to the same cluster.

*2) Adaptive Granularity Control Based on LR:* After clustering based on LSH, the subpopulation size (granularity) $M$ will be adaptively controlled based on the clustering analysis results. Consider two situations. When there are much more particles clustering nearby the globally worst particle ***gworst*** than the globally best particle ***gbest***, as shown in Fig. 3(a), it represents the exploration state. At this time, the ***gbest*** is far away from the current population, and the current population may obtain trapped in the local area. As a result, a smaller $M$ or decreasing $M$ is more likely to result in a larger number of subpopulations to maintain and improve the population diversity for the exploration state. Conversely, when there are much more particles clustering nearby the ***gbest*** than the ***gworst***, the algorithm is in the exploitation state like Fig. 3(b). At this time, a larger $M$ or increasing $M$ is more suitable to increase the learning neighborhood to accelerate the convergence speed for the exploitation state.

Since we have two evolutionary states (exploration and exploitation) and we wish to decrease or increase the subpopulation size $M$, respectively, a binary classifier is preferred here. In ML, LR is a famous binary classifier that can determine

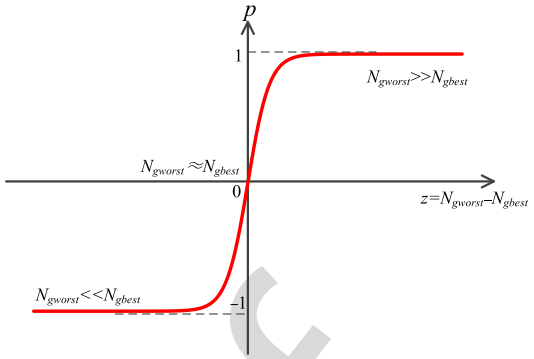


Fig. 4. Illustration of the *Tanh-sigmoid* function.

one or more independent variables to a dichotomous outcome (0 or 1). The core of LR is the *sigmoid* function, which maps the variables into a probability in (0, 1), shown as

$$p = \frac{1}{1 + e^{-z}} \tag{7}$$

where $z$ is the linear combination of the variables.

Therefore, we think that the LR can help us to determine/classify whether the algorithm is in exploration or exploitation evolutionary state. Herein, the variables for the input of LR are the number of particles in the same cluster with ***gworst*** (called $N_{\text{gworst}}$) and the number of particles in the same cluster with ***gbest*** (called $N_{\text{gbest}}$). Moreover, in our method, we used a variant of the LR function to estimate the evolutionary state and adaptively changed the $M$ according to the evolutionary state. Since we wish to adaptively increase or decrease the subpopulation size $M$, herein, we use a variant of the *sigmoid* function, called the *Tanh-sigmoid* function, to map the variables into the range in $(-1, 1)$, shown as

$$p = \frac{1 - e^{-2z}}{1 + e^{-2z}} \tag{8}$$

where the input $z$ is set as the difference between $N_{\text{gworst}}$ and $N_{\text{gbest}}$, as illustrated in Fig. 4. Therefore, the subpopulation size $M$ can be adaptively changed at the beginning of every generation as

$$M = M - \text{round}\left(\frac{1 - e^{-2\times(N_{\text{gworst}} - N_{\text{gbest}})}}{1 + e^{-2\times(N_{\text{gworst}} - N_{\text{gbest}})}}\right). \tag{9}$$

Using the above function, the following adaption rules are implemented. If $N_{\text{gworst}}$ is much larger than $N_{\text{gbest}}$, it means the current subpopulation may obtain trapped in the local optima. In this case, the LR will return 1 as shown in Fig. 4, where $z$ is very large, and therefore, $M$ will decrease by 1 and smaller $M$ will improve the population diversity. If $N_{\text{gbest}}$ is much larger than $N_{\text{gworst}}$, it indicates the global optima region may be found. Therefore, $M$ will increase by 1 (as shown in Fig. 4, where $z$ is very small and the LR will return $-1$) to obtain larger $M$ to speed up the convergence. However, when $N_{\text{gworst}}$ is closed to $N_{\text{gbest}}$, it means the current population may have a good balance between diversity and convergence. In this case, $z$ in Fig. 4 is near 0 and the LR will return 0, so that $M$ will remain unchanged.

Therefore, the adaptive granularity control based on LSH and LR can relieve the sensitivity of parameters and find

**Algorithm 1: AGLDPSO**

**Process of MASTER in AGLDPSO**
**Begin**
1. Randomly initialize the population; *FEs*=0;
2. Evaluate the population;
3. *FEs*=*FEs*+*N*;
4. **While** *FEs*≤MaxFEs
5. Determine the ***gbest***;
6. Adaptive *M* using (9); (Randomly select an integer as the subpopulation size *M* for initialization);
7. Randomly divide the population into *N*/*M* equal subpopulations;
8. **For** each subpopulation
9. Send the current subpopulation to the corresponding **SLAVE**;
10. Receive the updated subpopulation from the corresponding **SLAVE**;
11. **End For**
12. **End While**

**End**

**Process of SLAVE in ADGLPSO**
**Begin**
1. Receive a subpopulation from **MASTER**;
2. Determine the ***pbest***;
3. Update the worst particle $P_w$ using (3) and (4);
4. Evaluate $P_w$;
5. *FEs*=*FEs*+1;
6. Send the updated subpopulation to **MASTER**;

**End**

an appropriate subpopulation size for AGLDPSO, which can further find a potential balance between exploration and exploitation.

### D. Complete AGLDPSO Algorithm

Combining all the components mentioned above, the pseudocode of the complete AGLDPSO algorithm is outlined in Algorithm 1.

In each generation, we first adaptively set the size of subpopulation (learning granularity) *M* according to the LSH and LR using (9) in master. Then, the entire population will be randomly divided into *N*/*M* subpopulations of the same size. Next, the master will send each subpopulation to its corresponding slave, and each subpopulation is updated on its corresponding slave. After that, each updated subpopulation will be sent from its corresponding slave to master, and the master will collect all the updated subpopulations to form a new population. So far, we have finished a loop sequent. The procedures are repeated until the maximum number of FEs (MaxFEs) is met.

## IV. Experimental Results

### A. Experimental Setup

To test the performance of AGLDPSO, two widely used large-scale optimization benchmark functions test suites are used. The first one is the IEEE CEC2010 test suite [55], which contains 20 large-scale optimization benchmark functions. While the other one is the IEEE CEC2013 test suite [56], which contains 15 large-scale optimization benchmark functions.

We compare the results obtained by AGLDPSO with six PSO-based large-scale optimization algorithms, including CCPSO2 [30], SL-PSO [36], CSO [37], DSPLSO [48], DLLSO [49], and DMS-L-PSO [42]. Moreover, we also compare AGLDPSO with other three well-known large-scale optimization algorithms, including DECC with differential grouping (DECC-DG) [33], DECC with random grouping (DECC-G) [34], and multilevel CC (MLCC) [35]. The master–slave model of AGLDPSO is built in a multiprocessor distributed environment that consists of several distributed computing servers. The CPU of each server has eight processors configured with Intel Core i5-7400, 3.00 GHz. Therefore, we obtain the multiprocessor distributed environment and we can assign each subpopulation to one processor through MPI.

The MaxFEs is set as 3 000 000 for all competitors. The population size *N* is set as 500 in AGLDPSO and the interval for the *M* is $[10, \sqrt{N}]$. All the algorithms run 30 times independently for statistics and the mean results are reported. The parameters used in the compared algorithms are set the same in their original papers for a fair comparison. In addition, Wilcoxon's rank-sum test at $\alpha = 0.05$ is performed between AGLDPSO and other state-of-the-art large-scale optimization algorithms to evaluate the statistical significance of their performance [57]. The symbols "+," "≈," and "−" indicate AGLDPSO performs significantly better than (+), similar to (≈), or significantly worse than (−) the corresponding algorithm.

### B. Comparisons With State-of-the-Art Large-Scale Optimization Algorithms on the IEEE CEC2010 Test Suite

The functions in this test suite are with 1000 dimensions and can be classified into three groups. The first group consists of three separable functions $f_1$–$f_3$. The second group includes the following 15 functions $f_4$–$f_{18}$, which are partially separable functions. The last group consists of the last two functions $f_{19}$ and $f_{20}$ that are nonseparable functions. All these functions are shifted and rotated, which are more difficult to solve and make our test more comprehensive and convincing.

The detailed comparison results of AGLDPSO and other state-of-the-art large-scale optimization algorithms on the IEEE CEC2010 test suite are listed in Table I. For clarity, the best results are highlighted in boldface. From Table I, we can see the following.

For the first three separable functions $f_1$–$f_3$, AGLDPSO performs significantly better than most of other algorithms, especially on $f_3$. Although it performs slightly worse than DLLSO and MLCC on these three functions, they both lose their feasibilities when dealing with the partially separable or nonseparable functions, which will be discussed as follows.

For the next 15 partially separable functions $f_4$–$f_{18}$, AGLDPSO performs best on eight functions ($f_4$, $f_8$, $f_9$, $f_{12}$–$f_{14}$, $f_{17}$, and $f_{18}$). It dominates CCPSO2, DMS-L-PSO, DECC-G, and MLCC on all the 15 functions; dominates SL-PSO, CSO, DSPLSO, DLLSO, and DECC-DG on at least nine functions. Conversely, all the competitors cannot outperform AGLDPSO on more than five functions.

For the last two nonseparable functions $f_{19}$ and $f_{20}$, the performance of AGLDPSO is still better than or at least comparable to other algorithms, only worse than DECC-G on $f_{19}$ and CSO on $f_{20}$.

Overall, AGLDPSO performs better than CCPSO2, SL-PSO, CSO, DMS-L-PSO, DSPLSO, DLLSO, DECC-DG, DECC-G, and MLCC on 19, 17, 14, 20, 14, 11, 15,

TABLE I
EXPERIMENTAL RESULTS ON 1000-D IEEE CEC2010 FUNCTIONS

| fun | AGLDPSO | CCPSO2 | SL-PSO | CSO | DMS-L-PSO |
|---|---|---|---|---|---|
| | Mean±Std | Mean±Std | Mean±Std | Mean±Std | Mean±Std |
| $f_1$ | 1.22E-21±8.14E-22 | 1.40E+00±1.75E+00(+) | 8.47E-18±1.63E-18(+) | 4.58E-12±6.15E-13(+) | 4.88E+09±1.34E+08(+) |
| $f_2$ | 2.35E+03±9.05E+02 | 4.21E+00±1.22E+00(-) | 1.94E+03±1.07E+02(-) | 7.48E+03±1.82E+02(+) | 6.17E+03±1.87E+02(+) |
| $f_3$ | 1.32E-13±1.27E-15 | 4.61E-03±1.53E-03(+) | 1.76E+00±2.17E-01(+) | 2.58E-09±2.22E-10(+) | 1.74E+01±5.26E-02(+) |
| $f_4$ | **1.10E+11±9.13E+10** | 2.04E+12±1.19E+12(+) | 2.86E+11±5.93E+10(+) | 7.21E+11±1.75E+11(+) | 3.90E+13±4.40E+12(+) |
| $f_5$ | 1.12E+07±3.37E+06 | 4.45E+08±1.34E+08(+) | 2.61E+07±5.44E+06(+) | 1.13E+07±4.94E+07(≈) | 1.04E+08±7.83E+06(+) |
| $f_6$ | 1.99E+01±1.93E-02 | 1.80E+07±4.40E+06(+) | 2.00E+01±4.08E+00(+) | 8.53E-07±2.25E-08(-) | 1.66E+06±1.46E+05(+) |
| $f_7$ | 8.43E+04±6.32E+04 | 3.82E+08±6.95E+08(+) | 5.97E+04±3.49E+04(-) | 2.10E+04±4.87E+03(-) | 2.42E+10±1.63E+09(+) |
| $f_8$ | **2.96E+04±9.75E+03** | 2.74E+07±2.79E+07(+) | 7.71E+06±8.64E+05(+) | 3.86E+07±7.68E+04(+) | 1.43E+08±3.42E+07(+) |
| $f_9$ | **2.20E+07±2.36E+06** | 9.40E+07±3.41E+07(+) | 3.44E+07±3.01E+06(+) | 6.81E+07±5.58E+06(+) | 5.79E+09±2.02E+08(+) |
| $f_{10}$ | 2.38E+03±1.91E+02 | 4.87E+03±7.19E+02(+) | 3.08E+03±1.83E+03(+) | 9.57E+03±8.65E+01(+) | 5.88E+03±2.48E+02(+) |
| $f_{11}$ | 2.00E+01±2.78E+00 | 1.98E+02±3.73E-01(+) | 2.46E+01±4.47E+00(+) | 4.01E-08±4.54E-09(-) | 1.82E+02±1.38E+00(+) |
| $f_{12}$ | **1.33E+03±7.28E+02** | 3.72E+04±1.29E+04(+) | 1.72E+04±1.59E+04(+) | 4.38E+05±5.79E+04(+) | 2.84E+06±1.10E+05(+) |
| $f_{13}$ | **5.35E+02±1.42E+02** | 1.30E+03±1.53E+02(+) | 1.07E+03±5.21E+02(+) | 5.76E+02±1.30E+02(+) | 9.68E+07±2.62E+07(+) |
| $f_{14}$ | **6.30E+07±4.37E+06** | 2.37E+08±1.03E+08(+) | 8.51E+07±5.69E+06(+) | 2.47E+08±1.37E+07(+) | 5.02E+09±3.43E+08(+) |
| $f_{15}$ | 2.34E+03±1.95E+02 | 1.08E+04±1.34E+03(+) | 1.13E+04±1.08E+02(+) | 1.01E+04±4.12E+01(+) | 6.21E+03±2.76E+02(+) |
| $f_{16}$ | 6.21E+00±2.68E+00 | 3.96E+02±6.94E-01(+) | 2.27E+01±9.99E+00(-) | 5.76E-08±4.78E-09(-) | 3.39E+02±9.52E-01(+) |
| $f_{17}$ | **2.42E+04±2.54E+03** | 1.23E+05±5.22E+04(+) | 9.22E+04±2.21E+04(+) | 2.19E+06±1.37E+05(+) | 2.67E+06±1.54E+05(+) |
| $f_{18}$ | **1.40E+03±2.85E+02** | 3.06E+03±3.18E+02(+) | 2.56E+03±7.56E+02(+) | 1.66E+03±5.90E+02(+) | 2.82E+09±5.30E+08(+) |
| $f_{19}$ | 8.43E+05±4.61E+04 | 1.54E+06±1.08E+05(+) | 5.18E+06±7.00E+05(+) | 9.97E+06±5.93E+05(+) | 1.63E+07±6.70E+05(+) |
| $f_{20}$ | 1.44E+03±1.56E+02 | 2.14E+03±1.66E+02(+) | 1.77E+03±1.88E+02(+) | **1.11E+03±2.59E+02(-)** | 4.10E+09±7.56E+08(+) |
| +(AGLDPSO is significantly better) | | 19 | 17 | 14 | 20 |
| -(AGLDPSO is significantly worse) | | 1 | 3 | 6 | 0 |
| ≈ | | 0 | 0 | 0 | 0 |
| fun | DSPLSO | DLLSO | DECC-DG | DECC-G | MLCC |
| | Mean±Std | Mean±Std | Mean±Std | Mean±Std | Mean±Std |
| $f_1$ | 8.15E-20±8.38E-21(+) | 5.96E-22±6.18E-23(-) | 1.35E+01±6.50E+01(+) | 1.19E-14±1.10E-14(+) | **0.00E+00±0.00E+00(-)** |
| $f_2$ | 4.87E+02±1.63E+01(-) | 9.48E+02±4.86E+01(-) | 4.39E+03±2.02E+02(+) | 4.16E+01±2.22E+01(-) | **1.66E-01±3.77E-01(-)** |
| $f_3$ | 2.40E-13±1.71E-14(+) | **2.12E-14±2.66E-15(-)** | 1.66E+01±3.32E-01(+) | 1.77E+00±3.86E-01(+) | 8.17E-02±3.11E-01(+) |
| $f_4$ | 4.92E+11±8.54E+10(+) | 5.56E+11±1.00E+11(+) | 5.04E+12±1.44E+12(+) | 1.09E+13±3.52E+12(+) | 9.15E+12±2.93E+12(+) |
| $f_5$ | **7.16E+06±1.84E+06(-)** | 1.32E+07±4.41E+06(+) | 1.44E+08±1.85E+07(+) | 2.90E+08±1.05E+08(+) | 5.30E+08±1.14E+08(+) |
| $f_6$ | **9.05E-09±1.02E-09(-)** | 6.14E-01±8.45E-01(-) | 1.62E+01±4.07E-01(-) | 5.30E+06±1.42E+06(+) | 1.94E+07±1.21E+06(+) |
| $f_7$ | **4.72E+02±1.94E+02(-)** | 8.48E+02±1.23E+03(-) | 1.35E+04±1.38E+04(-) | 5.61E+06±1.13E+07(+) | 9.97E+05±7.57E+05(+) |
| $f_8$ | 3.33E+07±8.81E+04(+) | 3.23E+07±4.68E+05(+) | 1.61E+07±1.23E+07(+) | 6.50E+07±3.40E+07(+) | 5.74E+07±3.32E+07(+) |
| $f_9$ | 5.45E+07±5.09E+06(+) | 4.18E+07±4.55E+06(+) | 5.56E+07±6.06E+06(+) | 2.35E+08±2.40E+07(+) | 1.24E+08±1.44E+07(+) |
| $f_{10}$ | 6.32E+03±2.43E+02(+) | **8.26E+02±3.40E+01(-)** | 4.48E+03±1.22E+02(+) | 9.42E+03±5.22E+02(+) | 4.48E+03±1.51E+03(+) |
| $f_{11}$ | **4.83E-11±5.01E-12(-)** | 4.69E+00±5.64E+00(-) | 1.06E+01±1.05E+00(-) | 2.55E+01±1.38E+00(+) | 1.93E+02±2.66E+01(+) |
| $f_{12}$ | 9.33E+04±7.35E+03(+) | 1.58E+04±2.02E+03(+) | 2.70E+03±8.88E+02(+) | 4.08E+04±5.32E+03(+) | 3.56E+04±5.21E+03(+) |
| $f_{13}$ | 6.28E+02±2.78E+02(+) | 8.26E+02±2.40E+02(+) | 5.54E+03±3.56E+03(+) | 4.21E+03±3.17E+03(+) | 2.65E+03±1.91E+03(+) |
| $f_{14}$ | 1.25E+08±8.99E+06(+) | 1.67E+08±8.71E+06(+) | 3.37E+08±2.27E+07(+) | 5.85E+08±4.25E+07(+) | 3.16E+08±2.52E+07(+) |
| $f_{15}$ | 9.51E+03±7.98E+01(+) | **8.07E+02±3.68E+01(-)** | 5.85E+03±7.49E+01(+) | 8.23E+03±3.11E+03(+) | 9.44E+03±2.05E+03(+) |
| $f_{16}$ | 5.05E-12±6.58E-13(-) | 6.25E+00±2.87E+00(≈) | **7.31E-13±5.13E-14(-)** | 8.14E+01±1.15E+01(+) | 3.81E+02±5.80E+01(+) |
| $f_{17}$ | 7.43E+05±3.99E+04(+) | 9.28E+04±4.43E+03(+) | 4.06E+04±2.62E+03(+) | 1.63E+05±1.12E+04(+) | 1.61E+05±1.25E+04(+) |
| $f_{18}$ | 1.67E+03±3.88E+02(+) | 2.94E+03±7.96E+02(+) | 1.63E+10±2.52E+09(+) | 1.89E+04±1.05E+04(+) | 8.25E+03±6.34E+03(+) |
| $f_{19}$ | 8.77E+06±5.40E+05(+) | 1.63E+06±9.61E+04(+) | 1.74E+06±9.34E+04(+) | **7.36E+05±3.32E+04(-)** | 1.37E+06±8.20E+04(+) |
| $f_{20}$ | 1.75E+03±1.36E+02(+) | 1.74E+03±1.68E+02(+) | 6.42E+10±7.46E+09(+) | 3.35E+03±1.94E+02(+) | 2.05E+03±1.59E+02(+) |
| + | 14 | 11 | 16 | 18 | 18 |
| - | 6 | 8 | 4 | 2 | 2 |
| ≈ | 0 | 1 | 0 | 0 | 0 |

18, and 18 functions, respectively. Conversely, CCPSO2, SL-PSO, CSO, DSPLSO, DLLSO, DECC-DG, DECC-G, and MLCC can only surpass AGLDPSO on 1, 3, 6, 6, 8, 4, 2, and 2 functions, respectively. DMS-L-PSO cannot outperform AGLDPSO on any function. Therefore, AGLDPSO achieves the best performance on this test suite.

To further study the evolutionary behavior of different algorithms on the IEEE CEC2010 test functions, we draw their convergence curves to observe their evolutionary processes. Besides, in order to make our comparison more convincing, we choose several benchmark functions from all the three groups. Here, we select separable function $f_3$, partially separable functions $f_4$, $f_8$, $f_9$, and $f_{13}$, and nonseparable function $f_{20}$ as the representative instances. The convergence curves of AGLDPSO, CCPSO2, SL-PSO, CSO, DMS-L-PSO, DSPLSO, DLLSO, DECC-DG, DECC-G, and MLCC on these six selected benchmark functions are plotted in Fig. 5.

From Fig. 5(a), we can see that only AGLDPSO, DSPLSO, and DLLSO can converge to better solutions quickly while other algorithms evolve slower on separable function $f_3$. Moreover, AGLDPSO achieves faster convergence speed than DSPLSO and is a little slower than DLLSO on this function. While on partially separable functions $f_4$ and $f_9$ in Fig. 5(b) and (d), AGLDPSO and SL-PSO can achieve better and faster convergence than other algorithms, but AGLDPSO still has a faster convergence speed compared with SL-PSO. On

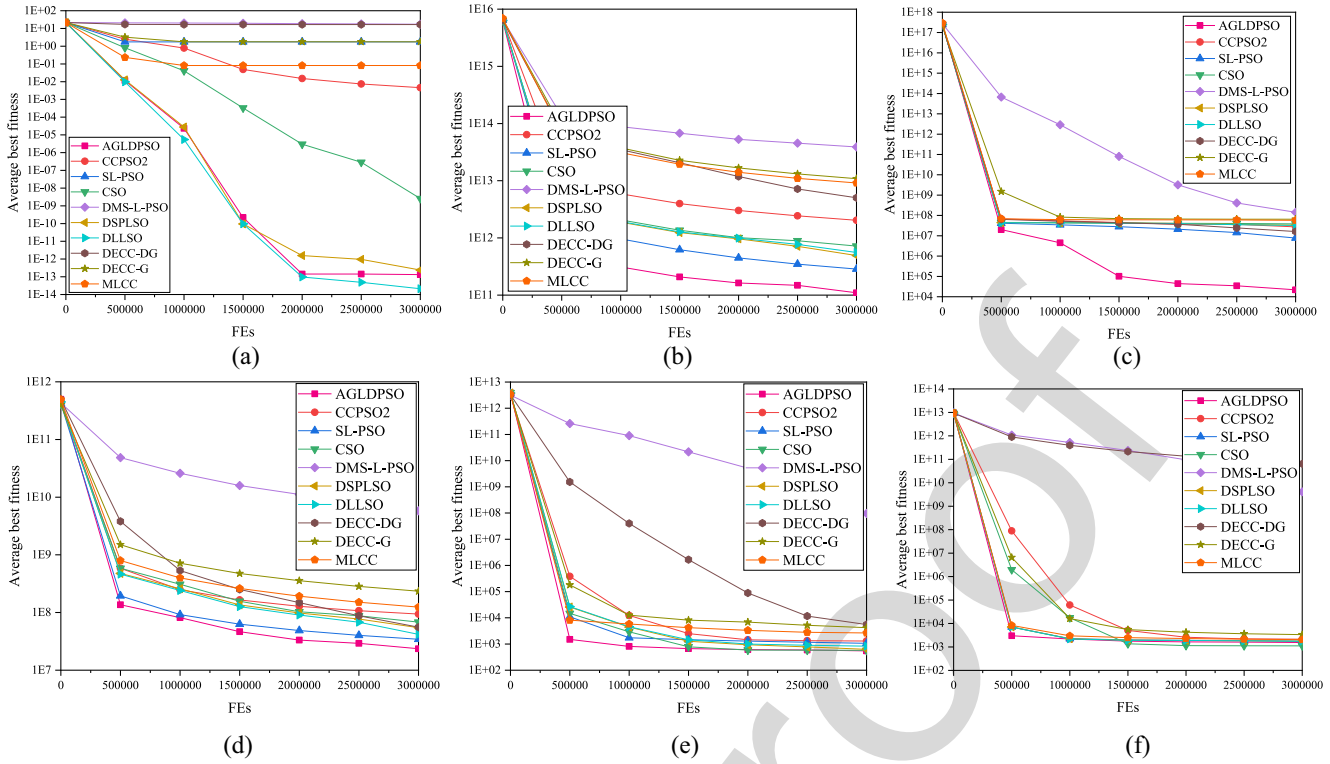


Fig. 5. Convergence curves of AGLDPSO and other state-of-the-art algorithms on six representative functions from IEEE CEC2010. (a) $f_3$. (b) $f_4$. (c) $f_8$. (d) $f_9$. (e) $f_{13}$. (f) $f_{20}$.

partially separable function $f_8$ shown in Fig. 5(c), most algorithms stagnate in the early stage, and only AGLDPSO can obtain better and more accurate solutions. While for the partially separable function $f_{13}$ and nonseparable function $f_{20}$ in Fig. 5(e) and (f), most algorithms can achieve the similar performance. However, AGLDPSO still obtains more accurate results than other algorithms, only performs a little worse than CSO on $f_{20}$.

Overall, AGLDPSO generally outperforms other large-scale optimization algorithms on these benchmark functions from IEEE CEC2010 test suite.

### C. Comparisons With State-of-the-Art Large-Scale Optimization Algorithms on the IEEE CEC2013 Test Suite

The functions in this test suite are with 1000 dimensions and can be classified into four groups. The first group includes three separable functions $f_1$–$f_3$. The second group consists of the following eight functions $f_4$–$f_{11}$, which are partially separable functions. The third group includes three overlapping functions $f_{12}$–$f_{14}$. (Note that $f_{13}$ and $f_{14}$ are with 905 dimensions.) The last group consists of the last function $f_{15}$ which is a nonseparable function.

The detailed comparison results of AGLDPSO and other state-of-the-art large-scale optimization algorithms on the IEEE CEC2013 test suite are listed in Table II. For clarity, the best results are highlighted in boldface. From Table II, we can see the following.

For the first three separable functions $f_1$–$f_3$, AGLDPSO dominates CSO, DMS-L-PSO, and DECC-DG on at least two functions. Moreover, AGLDPSO can achieve at least comparable performance among other algorithms, except DLLSO and MLCC. Although AGLDPSO performs a little worse than DLLSO and MLCC, they both lose their feasibilities when dealing with other functions, which will be discussed as follows.

For the next eight partially separable functions $f_4$–$f_{11}$, AGLDPSO performs the best on $f_4$ and $f_8$. It dominates all the other algorithms on at least four functions, while all the competitors cannot outperform AGLDPSO on more than three functions.

For the next three overlapping functions $f_{12}$–$f_{14}$, AGLDPSO performs significantly better than most of other algorithms, especially on $f_{14}$. It dominates other algorithms on at least two functions, except CSO and DSPLSO. Even if CSO and DSPLSO perform better than AGLDPSO on $f_{12}$ and $f_{13}$, they both lose their feasibilities when dealing with the nonseparable function, shown as follows.

For the last nonseparable function $f_{15}$, AGLDPSO achieves the best performance significantly and dominates all the competitors.

Overall, AGLDPSO performs better than CCPSO2, SL-PSO, CSO, DMS-L-PSO, DSPLSO, DLLSO, DECC-DG, DECC-G, and MLCC on 9, 8, 9, 11, 7, 7, 12, 10, and 8 functions, respectively. Conversely, CCPSO2, SL-PSO, CSO, DMS-L-PSO, DSPLSO, DLLSO, DECC-DG, DECC-G, and MLCC can only surpass AGLDPSO on 4, 5, 4, 4, 3, 5, 3, 5, and 6 functions, respectively. Therefore, AGLDPSO achieves the best performance on this test suite.

To further study the evolutionary behavior of different algorithms on the IEEE CEC2013 test functions, we draw their

TABLE II
EXPERIMENTAL RESULTS ON1000-D IEEE CEC2013 FUNCTIONS

| fun | AGLDPSO | CCPSO2 | SL-PSO | CSO | DMS-L-PSO |
|---|---|---|---|---|---|
| | Mean±Std | Mean±Std | Mean±Std | Mean±Std | Mean±Std |
| $f_1$ | 1.46E-21±1.19E-22 | 3.53E+01±2.33E+01(+) | 1.64E-17±3.30E-17(+) | 7.88E-12±1.21E-12(+) | 5.70E+09±1.44E+08(+) |
| $f_2$ | 2.56E+03±1.29E+02 | 3.57E+01±5.22E+00(-) | 2.09E+03±1.33E+02(-) | 8.58E+03±1.79E+02(+) | 1.24E+04±2.69E+02(+) |
| $f_3$ | 2.16E+01±4.69E-03 | **2.00E+01±9.00E-05(-)** | 2.16E+01±1.85E-03(≈) | 2.16E+01±5.99E-03(≈) | 2.14E+01±1.92E-02(-) |
| $f_4$ | **1.13E+09±7.72E+08** | 3.41E+10±1.97E+10(+) | 4.33E+09±1.01E+09(+) | 1.35E+10±3.17E+09(+) | 9.00E+11±3.78E+10(+) |
| $f_5$ | 8.57E+05±2.30E+05 | 1.64E+07±4.22E+06(+) | 8.88E+05±1.32E+05(+) | **5.97E+05±1.04E+05(-)** | 5.49E+06±4.19E+05(+) |
| $f_6$ | 1.05E+06±1.58E+03 | 1.05E+06±7.68E+03(≈) | 1.06E+06±1.32E+03(+) | 1.06E+06±1.20E+03(+) | **1.03E+06±3.99E+03(-)** |
| $f_7$ | 3.71E+06±2.38E+06 | 3.04E+08±4.42E+08(+) | 1.58E+06±7.65E+05(-) | 5.81E+06±3.09E+06(+) | 3.55E+09±2.61E+08(+) |
| $f_8$ | **1.36E+13±4.73E+12** | 1.02E+15±5.48E+14(+) | 1.09E+14±5.54E+13(+) | 2.46E+14±8.86E+13(+) | 6.79E+15±1.38E+15(+) |
| $f_9$ | 1.59E+08±2.51E+07 | 3.72E+09±1.01E+09(+) | 7.98E+07±1.18E+07(-) | **6.08E+07±1.31E+07(-)** | 5.05E+08±2.66E+07(+) |
| $f_{10}$ | 9.39E+07±3.50E+05 | 9.32E+07±4.92E+05(-) | 9.26E+07±1.66E+06(-) | 9.40E+07±2.24E+05(+) | 9.31E+07±3.11E+05(-) |
| $f_{11}$ | 9.35E+11±4.59E+10 | 9.32E+11±1.08E+10(≈) | 9.44E+11±9.10E+09(≈) | 9.29E+11±9.80E+09(≈) | 4.96E+11±4.01E+10(-) |
| $f_{12}$ | 1.53E+03±2.79E+02 | 2.15E+03±2.12E+02(+) | 1.77E+03±1.71E+02(+) | **1.08E+03±7.51E+01(-)** | 4.42E+09±8.34E+08(+) |
| $f_{13}$ | 2.12E+10±3.49E+09 | 4.13E+09±1.70E+09(-) | 5.19E+08±4.93E+08(-) | 7.48E+08±2.89E+08(-) | 1.16E+11±1.05E+10(+) |
| $f_{14}$ | **9.46E+07±5.68E+06** | 9.37E+10±1.02E+11(+) | 2.51E+08±2.29E+08(+) | 3.67E+09±3.38E+09(+) | 1.25E+12±1.59E+11(+) |
| $f_{15}$ | **2.73E+06±1.19E+05** | 6.95E+06±5.80E+06(+) | 6.03E+07±6.64E+06(+) | 7.61E+07±6.24E+06(+) | 1.60E+09±6.18E+08(+) |
| +(AGLDPSO is significantly better) | | 9 | 8 | 9 | 11 |
| -(AGLDPSO is significantly worse) | | 4 | 5 | 4 | 4 |
| ≈ | | 2 | 2 | 2 | 0 |
| fun | DSPLSO | DLLSO | DECC-DG | DECC-G | MLCC |
| | Mean±Std | Mean±Std | Mean±Std | Mean±Std | Mean±Std |
| $f_1$ | 1.47E-18±9.05E-19(+) | 4.58E-22±1.22E-23(-) | 9.65E+00±2.70E+01(+) | 4.22E-12±4.03E-12(+) | **2.81E-30±1.17E-29(-)** |
| $f_2$ | 1.07E+03±4.54E+02(-) | 1.26E+03±5.86E+01(-) | 1.28E+04±6.17E+02(+) | 3.65E+01±1.57E+01(-) | **2.69E+00±3.88E+00(-)** |
| $f_3$ | 2.16E+01±7.48E-03(≈) | 2.16E+01±4.31E-03(≈) | 2.14E+01±1.58E-02(-) | 2.01E+01±3.14E-03(-) | **2.00E+01±6.97E-05(-)** |
| $f_4$ | 9.40E+09±1.81E+09(+) | 6.92E+09±1.11E+09(+) | 7.87E+10±3.79E+10(+) | 9.34E+10±3.22E+10(+) | 1.10E+11±6.25E+10(+) |
| $f_5$ | 6.64E+05±1.56E+05(-) | 6.93E+05±1.98E+05(-) | 5.61E+06±3.82E+05(+) | 8.19E+06±1.67E+06(+) | 1.42E+07±4.31E+06(+) |
| $f_6$ | 1.06E+06±8.47E+02(+) | 1.06E+06±8.95E+02(+) | 1.06E+06±1.68E+03(+) | 1.06E+06±2.29E+03(+) | 1.05E+06±4.86E+03(≈) |
| $f_7$ | 5.63E+06±2.93E+06(+) | **1.54E+06±8.47E+05(-)** | 4.72E+08±2.41E+08(+) | 3.95E+08±3.26E+08(+) | 6.18E+08±5.39E+08(+) |
| $f_8$ | 1.67E+14±2.73E+13(+) | 1.54E+14±3.39E+13(+) | 4.05E+15±2.11E+15(+) | 3.04E+15±1.54E+15(+) | 4.34E+15±2.26E+15(+) |
| $f_9$ | 8.72E+07±2.10E+07(-) | 1.52E+08±3.25E+07(≈) | 4.82E+08±2.38E+07(+) | 5.98E+08±1.27E+08(+) | 1.16E+09±3.83E+08(+) |
| $f_{10}$ | 9.39E+07±2.99E+05(≈) | 9.40E+07±2.37E+05(+) | 9.45E+07±2.74E+05(+) | 9.29E+07±5.52E+05(-) | **9.25E+07±6.24E+05(-)** |
| $f_{11}$ | 9.27E+11±9.21E+09(≈) | 9.30E+11±9.87E+09(≈) | **4.09E+10±5.06E+10(-)** | 6.53E+10±4.12E+10(-) | 8.97E+10±7.11E+10(-) |
| $f_{12}$ | 1.10E+03±5.58E+01(-) | 1.97E+03±1.10E+02(+) | 1.68E+11±1.61E+10(+) | 3.70E+03±1.21E+03(+) | 2.26E+03±7.14E+02(+) |
| $f_{13}$ | 1.06E+09±4.36E+08(-) | **3.40E+08±1.46E+08(-)** | 1.98E+10±4.84E+09(-) | 5.44E+09±1.55E+09(-) | 6.42E+09±2.05E+09(-) |
| $f_{14}$ | 8.44E+09±6.84E+09(+) | 1.58E+08±1.36E+08(+) | 2.02E+10±1.20E+10(+) | 7.86E+10±3.33E+10(+) | 1.25E+11±8.27E+10(+) |
| $f_{15}$ | 4.76E+07±3.35E+06(+) | 4.62E+06±3.79E+05(+) | 9.44E+06±1.62E+06(+) | 5.81E+06±4.94E+06(+) | 6.87E+06±8.83E+05(+) |
| + | 7 | 7 | 12 | 10 | 8 |
| - | 5 | 5 | 3 | 5 | 6 |
| ≈ | 3 | 3 | 0 | 0 | 1 |

convergence curves to observe their evolutionary processes. Besides, in order to make our comparison more convincing, we choose several benchmark functions from all the four groups. Here, we select separable function $f_1$, partially separable functions $f_4, f_5$, and $f_8$, overlapping function $f_{14}$, and nonseparable function $f_{15}$ as the representative instances. The convergence curves of AGLDPSO, CCPSO2, SL-PSO, CSO, DMS-L-PSO, DSPLSO, DLLSO, DECC-DG, DECC-G, and MLCC on these six selected benchmark functions are plotted in Fig. 6.

From Fig. 6(a), we can see that only AGLDPSO and MLCC can converge to better solutions quickly while other algorithms evolve slower on separable function $f_1$. However, AGLDPSO converges a little slower than MLCC. While on partially separable function $f_4$ in Fig. 6(b), only AGLDPSO and SL-PSO can obtain more accurate results. Similar phenomenon can be observed on partially separable function $f_8$ and overlapping function $f_{14}$ in Fig. 6(d) and (e). On partially separable function $f_5$ in Fig. 6(c), AGLDPSO performs much better and obtains more promising results than most algorithms, only worse than CSO, DSPLSO, and DLLSO. While on the nonseparable function $f_{15}$ in Fig. 6(e), most algorithms can achieve the similar performance. However, AGLDPSO can obtain much better and more accurate results than other competitors.

Overall, AGLDPSO generally outperforms other large-scale optimization algorithms on these benchmark functions from IEEE CEC2013 test suite.

### *D. Comparison With Winner of IEEE CEC2010 Competition*

To further demonstrate the superiority of AGLDPSO, in this section, we compare AGLDPSO with the winner of the IEEE CEC2010 competition on large-scale optimization, the memetic algorithm based on local search chains (MA-SW-Chains) [58]. For a fair comparison, we directly cite the mean results of MA-SW-Chains from the original paper [58].

The detailed comparison results of AGLDPSO and MA-SW-Chains are listed in Table III. The best results are highlighted in boldface. Due to the lack of the detailed results of MA-SW-Chains in each run, whether AGLDPSO is better than (+), worse than (−), or similar to (≈) MA-SW-Chains is just measured by the mean results.

From Table III, we find that AGLDPSO still keeps its promising performance when compared with MA-SW-Chains.

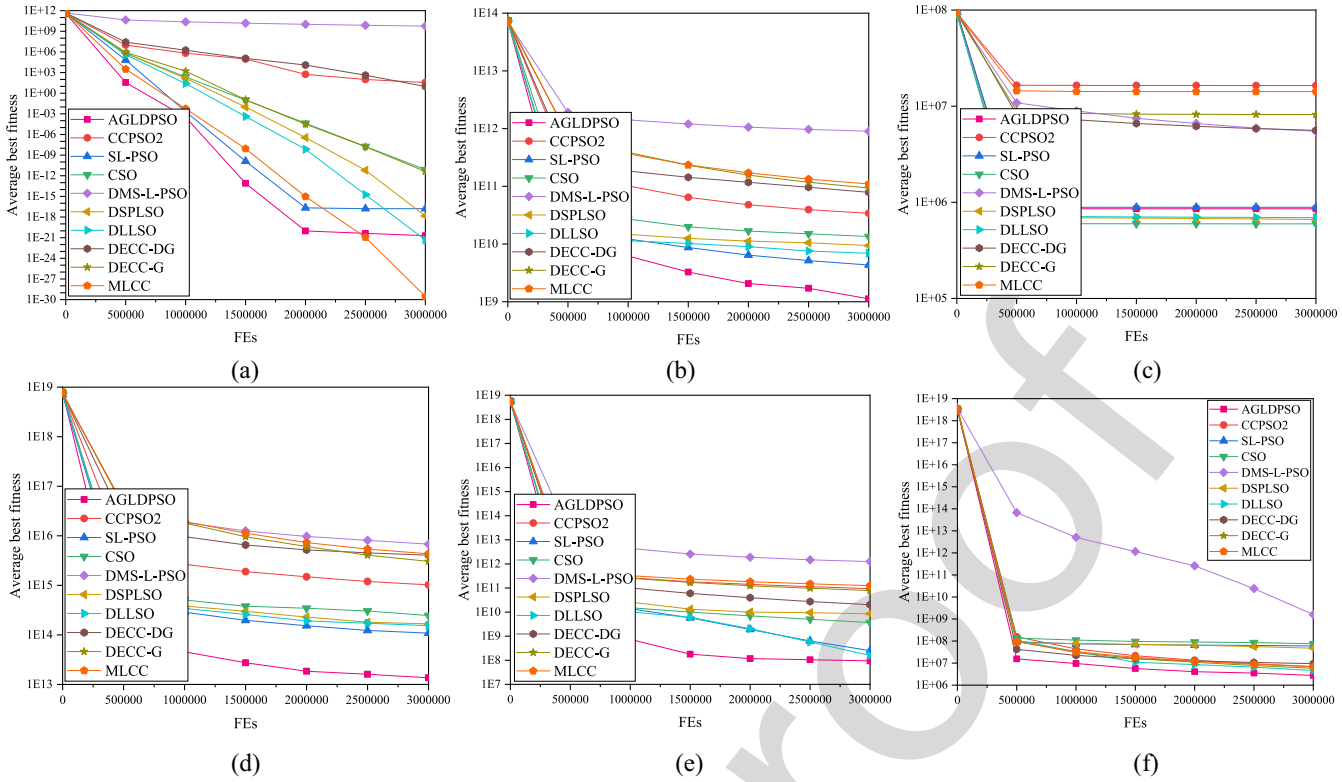


Fig. 6. Convergence curves of AGLDPSO and other state-of-the-art algorithms on six representative functions from IEEE CEC2013. (a) $f_1$. (b) $f_4$. (c) $f_5$. (d) $f_8$. (e) $f_{14}$. (f) $f_{15}$.

TABLE III
EXPERIMENTAL RESULTS OF AGLDPSO AND MA-SW-CHAINS ON 1000-D IEEE CEC2010 FUNCTIONS

| fun | AGLDPSO | MA-SW-Chains |
|---|---|---|
| | Mean±Std | Mean±Std |
| $f_1$ | **1.22E-21±8.14E-22** | 2.10E-14±1.99E-14(+) |
| $f_2$ | 2.35E+03±9.05E+02 | **8.10E+02±5.88E+01(-)** |
| $f_3$ | **1.32E-13±1.27E-15** | 7.28E-13±3.40E-13(+) |
| $f_4$ | **1.10E+11±9.13E+10** | 3.53E+11±3.12E+10(+) |
| $f_5$ | **1.12E+07±3.37E+06** | 1.68E+08±1.04E+08(+) |
| $f_6$ | **1.99E+01±1.93E-02** | 8.14E+04±2.84E+05(+) |
| $f_7$ | 8.43E+04±6.32E+04 | **1.03E+02±8.70E+01(-)** |
| $f_8$ | **2.96E+04±9.75E+03** | 1.41E+07±3.68E+07(+) |
| $f_9$ | 2.20E+07±2.36E+06 | **1.41E+07±1.15E+06(-)** |
| $f_{10}$ | 2.38E+03±1.91E+02 | **2.07E+03±1.44E+02(-)** |
| $f_{11}$ | **2.00E+01±2.78E+00** | 3.80E+01±7.35E+00(+) |
| $f_{12}$ | 1.33E+03±7.28E+02 | **3.62E-06±5.92E-07(-)** |
| $f_{13}$ | **5.35E+02±1.42E+02** | 1.25E+03±5.72E+02(+) |
| $f_{14}$ | 6.30E+07±4.37E+06 | **3.11E+07±1.93E+06(-)** |
| $f_{15}$ | **2.34E+03±1.95E+02** | 2.74E+03±1.22E+02(+) |
| $f_{16}$ | **6.21E+00±2.68E+00** | 9.98E+01±1.40E+01(+) |
| $f_{17}$ | 2.42E+04±2.54E+03 | **1.24E+00±1.25E-01(-)** |
| $f_{18}$ | 1.40E+03±2.85E+02 | **1.30E+03±4.36E+02(-)** |
| $f_{19}$ | 8.43E+05±4.61E+04 | **2.85E+05±1.78E+04(-)** |
| $f_{20}$ | 1.44E+03±1.56E+02 | **1.07E+03±7.29E+01(-)** |
| +(AGLDPSO is better) | | 10 |
| -(AGLDPSO is worse) | | 10 |
| ≈ | | 0 |

First, on the separable functions $f_1$–$f_3$ and the partially separable functions $f_4$–$f_{18}$, AGLDPSO dominates MA-SW-Chains on two functions and eight functions, respectively, while only dominated by MA-SW-Chains on one function and seven functions, respectively. Second, in total, the number of functions that AGLDPSO dominates MA-SW-Chains is the same to the number of functions that AGLDPSO is dominated by MA-SW-Chains, which means AGLDPSO can achieve the equivalent performance compared to MA-SW-Chains. However, we can see that AGLDPSO achieves the mean results with 1.99E+01 and 2.96E+04 on $F_6$ and $F_8$, respectively, which is much better than the mean results in MA-SW-Chains with 8.14E+04 and 1.41E+07. Even on the functions where AGLDPSO is dominated by MA-SW-Chains, AGLDPSO still achieves the comparable performance to MA-SW-Chains. For example, on $f_{10}$ and $F_{18}$, AGLDPSO achieves the mean results with 2.38E+03 and 1.40E+03, which is very close to the mean results in MA-SW-Chains with 2.07E+03 and 1.30E+03. Third, compared with MA-SW-Chains, AGLDPSO is much easier and simpler to understand and implement, which fully facilitates its practical application.

Overall, we can see that AGLDPSO is better than or at least competitive to the winner of the IEEE CEC2010 competition.

### *E. Scalability of AGLDPSO on 2000-D Problems*

In order to investigate the scalability of AGLDPSO, we further compare the performance of AGLDPSO with other large-scale optimization algorithms on the IEEE CEC2010 test functions with dimensionality increasing to 2000.

When dealing with the 2000-D problems, the MaxFEs is set as 6 000 000 for all competitors, while the population size $N$ is set as 1000 in AGLDPSO. Moreover, the parameter $c_2$ which controls the convergence speed to the ***gbest*** is set as 0.2. The detailed experimental results can be seen in Table S.II in the supplementary file.

TABLE IV
EXPERIMENTAL RESULTS OF AGLDPSO AND ITS VARIANTS WITH DIFFERENT SUBPOPULATION SIZES ON 1000-D IEEE CEC2010 FUNCTIONS

| fun | AGLDPSO | | AGLDPSO(10) | | AGLDPSO(15) | | AGLDPSO(20) | | AGLDPSO(rand) | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Mean±Std | Time | Mean±Std | Time | Mean±Std | Time | Mean±Std | Time | Mean±Std | Time |
| $f_1$ | 1.22E-21±8.14E-22 | 452.1 | **1.03E-21±1.82E-22(-)** | 246.2 | 1.28E-21±2.55E-22(≈) | 322.9 | 1.86E-21±6.13E-22(+) | 409.9 | 1.94E-21±9.94E-22(+) | 360.4 |
| $f_2$ | 2.35E+03±9.05E+02 | 386.0 | **1.66E+03±7.81E+01(-)** | 224.0 | 2.30E+03±1.13E+02(≈) | 271.5 | 2.84E+03±1.37E+02(+) | 330.1 | 2.65E+03±1.39E+02(+) | 290.9 |
| $f_3$ | **1.32E-13±1.27E-15** | 313.5 | 1.38E-13±7.45E-15(≈) | 145.7 | 1.03E+00±3.13E-01(+) | 210.2 | 1.67E+00±1.76E-01(+) | 284.1 | 1.51E+00±2.19E-01(+) | 214.4 |
| $f_4$ | **1.10E+11±9.13E+10** | 432.0 | 1.23E+11±2.65E+10(+) | 257.7 | 1.14E+11±2.65E+10(≈) | 335.1 | 1.11E+11±5.27E+10(≈) | 425.2 | 1.50E+11±4.27E+10(+) | 374.4 |
| $f_5$ | **1.12E+07±3.37E+06** | 396.6 | 1.15E+08±1.21E+08(+) | 237.8 | 3.52E+07±4.83E+07(+) | 301.6 | 2.99E+07±7.04E+06(+) | 307.4 | 2.63E+07±5.79E+06(+) | 302.7 |
| $f_6$ | **1.99E+01±1.93E-02** | 374.9 | 2.14E+01±1.27E-01(+) | 240.4 | **1.99E+01±5.89E-02(≈)** | 265.2 | **1.99E+01±1.17E-02(≈)** | **317.5** | **1.99E+01±1.46E-02(≈)** | **315.0** |
| $f_7$ | 8.43E+04±6.32E+04 | 433.6 | **6.82E-04±3.74E-04(-)** | 226.0 | 8.22E+04±8.15E+04(-) | 310.9 | 5.29E+05±1.99E+05(+) | 384.7 | 3.72E+05±1.95E+05(+) | 360.1 |
| $f_8$ | 2.96E+04±9.75E+03 | 436.8 | **3.81E+03±1.53E+03(-)** | 227.2 | 3.98E+04±1.51E+04(+) | 311.8 | 3.41E+05±9.83E+04(+) | 388.6 | 1.94E+05±5.27E+04(+) | 361.1 |
| $f_9$ | 2.20E+07±2.36E+06 | 440.5 | 2.71E+07±2.85E+06(+) | 289.1 | 2.34E+07±2.43E+06(+) | 350.3 | **2.06E+07±2.19E+06(-)** | 451.5 | 2.22E+07±2.29E+06(≈) | 396.4 |
| $f_{10}$ | 2.38E+03±1.91E+02 | 355.1 | **1.77E+03±8.48E+01(-)** | 247.2 | 2.46E+03±1.23E+02(≈) | 317.5 | 3.05E+03±1.95E+02(+) | 387.7 | 2.79E+03±1.07E+02(+) | 295.9 |
| $f_{11}$ | **2.00E+01±2.78E+00** | 345.3 | 2.04E+01±3.83E-01(≈) | 230.7 | 2.24E+01±4.92E+00(+) | 241.1 | 3.70E+01±1.43E+01(+) | 303.1 | 3.45E+01±1.50E+01(+) | 281.5 |
| $f_{12}$ | 1.33E+03±7.28E+02 | 345.0 | 5.18E+03±5.33E+02(+) | 190.6 | 1.40E+03±3.01E+02(≈) | 253.8 | **1.29E+03±3.98E+02(≈)** | 345.2 | 2.77E+03±6.89E+03(+) | 302.7 |
| $f_{13}$ | **5.35E+02±1.42E+02** | 395.6 | **5.35E+02±1.10E+02(≈)** | 200.4 | 5.41E+02±1.12E+02(≈) | 276.7 | 5.47E+02±1.50E+02(+) | 360.5 | 5.64E+02±1.34E+02(+) | 313.2 |
| $f_{14}$ | 6.30E+07±4.37E+06 | 597.7 | 8.28E+07±5.49E+06(+) | 404.8 | 6.75E+07±4.03E+06(+) | 482.4 | **6.02E+07±5.20E+06(-)** | 566.6 | 6.55E+07±5.20E+06(+) | 514.9 |
| $f_{15}$ | **2.34E+03±1.95E+02** | 465.0 | 5.97E+03±4.52E+03(+) | 346.9 | 2.55E+03±1.22E+02(+) | 354.8 | 3.15E+03±1.54E+02(+) | 418.6 | 2.93E+03±1.56E+02(+) | 378.5 |
| $f_{16}$ | **6.21E+00±2.68E+00** | 406.1 | 7.30E+00±8.20E+00(≈) | 269.7 | 4.63E+01±2.80E+01(+) | 330.1 | 9.27E+01±2.66E+01(+) | 383.5 | 8.28E+01±2.66E+01(+) | 352.2 |
| $f_{17}$ | 2.42E+04±2.54E+03 | 497.8 | 7.69E+04±4.82E+03(+) | 261.7 | 2.99E+04±2.78E+03(+) | 362.5 | **2.04E+04±4.32E+03(-)** | 450.0 | 2.35E+04±2.69E+03(≈) | 308.7 |
| $f_{18}$ | **1.40E+03±2.85E+02** | 524.2 | 1.57E+03±3.10E+02(+) | 272.8 | 1.57E+03±3.55E+03(+) | 374.3 | 1.52E+03±2.67E+02(+) | 459.4 | 1.63E+03±3.22E+02(+) | 321.8 |
| $f_{19}$ | 8.43E+05±4.61E+04 | 493.9 | 3.32E+06±5.38E+05(+) | 261.2 | 9.25E+05±4.64E+04(+) | 361.9 | **7.97E+05±5.21E+04(-)** | 448.6 | 8.13E+05±4.67E+04(-) | 306.5 |
| $f_{20}$ | **1.44E+03±1.56E+02** | 534.4 | 1.55E+03±1.23E+02(+) | 279.0 | 1.51E+03±1.20E+02(+) | 380.6 | 1.53E+03±1.45E+02(+) | 466.6 | 1.56E+03±1.35E+02(+) | 321.7 |
| +(AGLDPSO is significantly better) | | | 11 | | 12 | | 13 | | 16 | |
| -(AGLDPSO is significantly worse) | | | 5 | | 1 | | 4 | | 1 | |
| ≈ | | | 4 | | 7 | | 3 | | 3 | |

It can be observed that as the dimension increases, the performance of many algorithms is greatly deteriorated, except AGLDPSO. Besides, MLCC may be more suitable for solving separable functions, while CSO performs relatively better on some partially separable functions. Even so, AGLDPSO still keeps it tremendous advantage and superiority on all functions. It performs better than CCPSO2, SL-PSO, CSO, DMS-L-PSO, DSPLSO, DLLSO, DECC-DG, DECC-G, and MLCC on 19, 15, 13, 20, 11, 11, 17, 16, and 18 functions, respectively. Conversely, CCPSO2, SL-PSO, CSO, DSPLSO, DLLSO, DECC-DG, DECC-G, and MLCC can only surpass AGLDPSO on 1, 2, 7, 8, 8, 2, 4, and 2 functions, respectively. DMS-L-PSO cannot outperform AGLDPSO on any functions. These results fully demonstrate that AGLDPSO can also achieve good performance even the dimension increases to 2000.

### *F. Effects of AGLS*

In this section, we will discuss the property and influence of AGLS, which can achieve the adaptive subpopulation size and control the learning granularity. Herein, to investigate the effectiveness of the new proposed AGLS, we compare AGLDPSO with three AGLDPSO variants with fixed subpopulation size and one AGLDPSO variant with random subpopulation size. We denote the AGLDPSO variant with fixed subpopulation size $M = a$ as AGLDPSO($a$) and denote the AGLDPSO variant with random subpopulation size as AGLDPSO(rand). The comparison results of AGLDPSO and its variants on the IEEE CEC2010 test functions are listed in Table IV.

As we can see, different subpopulation sizes are suitable for different problems. For instance, a large subpopulation size (coarse granularity) is appropriate for exploitation, performing well on unimodal functions $f_4$, $f_{12}$, and $f_{19}$. While a small subpopulation size (fine-granularity) is suitable for diversity maintaining, performing well on multimodal functions $f_2$, $f_{11}$, and $f_{16}$. However, the adaptive granularity control achieved by AGLS, without any prior information, still outperforms AGLDPSO(10), AGLDPSO(15), and AGLDPSO(20) on 11, 12, and 13 functions, respectively, while is worse than these three variants on only 5, 1, and 4 functions, respectively. Moreover, it performs better than AGLDPSO(rand) on 16 functions and only worse than AGLDPSO(rand) on 1 function. This may be due to that the AGLS can estimate the evolutionary state based on LSH and LR, which will further adaptively change the subpopulation size to meet the search requirement of the current evolutionary state. Therefore, benefited from the adaptive granularity control, we not only eliminate the sensitivity of this parameter but also find a potential balance between exploration and exploitation to obtain better performance on a broad range of functions.

Moreover, from Table IV, we can see that AGLDPSO generally consumes more CPU time (measured in second) than its variants with fixed subpopulation size or random subpopulation size. This may be due to that the adaptive subpopulation size $M$ in AGLS involves hashing-based clustering and the LR, which are relatively time consuming. Although the AGLS induces some extra CPU time to AGLDPSO, it also helps AGLDPSO find the suitable subpopulation size to effectively control the learning granularity, which will further balance the exploration and exploitation, obtaining better results.

Next, we further count the number of different subpopulation size $M$ during the evolutionary process in AGLDPSO. From the above experiments, we find that the large subpopulation size (coarse granularity) performs well on unimodal functions $f_4$, $f_{12}$, and $f_{19}$, while the small subpopulation size (fine-granularity) is suitable for multimodal functions $f_2$, $f_{11}$, and $f_{16}$. Thus, we take these six functions as the representative instances to further verify the effectiveness of adaptive granularity control. The statistical results of $M$ in the evolutionary process on these six functions are shown in Fig. 7.

From Fig. 7, we can see that the small subpopulation size appears more frequently than the large subpopulation size in all the listed functions. This may be due to that when solving the large-scale optimization problems, more diversity is needed. Small subpopulation size $M$ will result in large number of subpopulations, and many ***sbest*** can be learned to increase

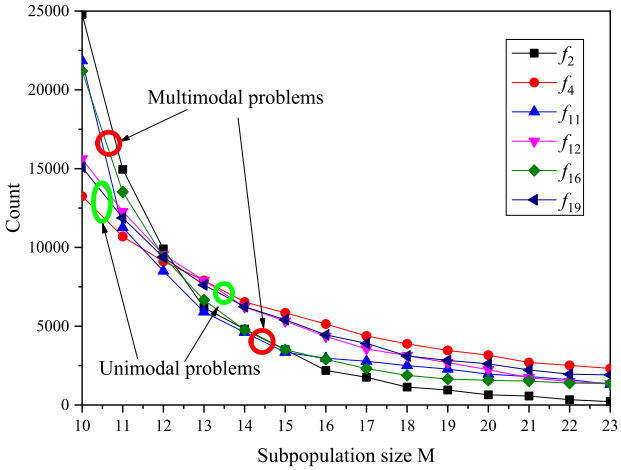


Fig. 7. Statistical results of subpopulation size $M$ on different functions.

the diversity. As a result, small subpopulation size is more needed to keep population diversity when solving the large-scale optimization problems. Moreover, Fig. 7 also shows that in multimodal functions $f_2$, $f_{11}$, and $f_{16}$, the small subpopulation size $M$ appears more frequently than that in unimodal functions $f_4$, $f_{12}$, and $f_{19}$. This is due to that small $M$ can improve the population diversity to avoid local optima and therefore, is more needed in the multimodal environment. On the contrary, the large subpopulation size $M$ appears more frequently in unimodal functions $f_4$, $f_{12}$, and $f_{19}$ than that in multimodal functions $f_2$, $f_{11}$, and $f_{16}$, because large $M$ can further accelerate the convergence speed and improve the accuracy of solution in the unimodal environment. Therefore, we can say that our adaptive granularity control method in the AGLDPSO algorithm is effective and adaptive to the evolutionary states of different problems.

### G. Influences of Parameters

The bucket number $nb$ has an important influence on the effectiveness of AGLS. If $nb$ is small, the bucket size will be large. Thus, ***gbest*** and ***gworst*** will be in the same bucket with a higher probability, and there is no difference between $N_{\text{gworst}}$ and $N_{\text{gbest}}$. While if $nb$ is large, the bucket size will be small and the population will be divided more dispersedly. Therefore, the number of particles in each bucket is relatively small and there is still no significant difference between $N_{\text{gworst}}$ and $N_{\text{gbest}}$. Since the subpopulation size $M$ is adaptively changed according to the difference between $N_{\text{gworst}}$ and $N_{\text{gbest}}$, neither larger nor smaller $nb$ is suitable for AGLS.

The $nb$ is tested with four values, $0.05N$, $0.1N$ (the one used in AGLDPSO), $0.15N$, and $0.2N$. The AGLDPSO variant with $nb = \lambda \times N$ is called AGLDPSO ($\lambda$). For example, the AGLDPSO variant with $nb = 0.05N$ is called AGLDPSO(0.05). The comparison results of AGLDPSO and its variants with different $nb$ values on the IEEE CEC2010 test suite are listed in Table S.III in the supplementary file.

As we can see, different $nb$ values make nearly no difference in AGLDPSO on $f_1, f_4, f_5, f_{15}$, and $f_{20}$. Larger $nb$ value with $0.2N$ performs the best on $f_7$, while smaller $nb$ value with $0.05N$ performs the best on $f_2$, $f_8$, and $f_{10}$. However, on the other functions, AGLDPSO with $nb = 0.1N$ performs better than AGLDPSO(0.05), AGLDPSO(0.15), and AGLDPSO(0.2) on 10, 11, and 12 functions, respectively. Therefore, $nb = 0.1N$ is the most suitable parameter for AGLDPSO to achieve a good performance in our testing.

Moreover, we also investigate the influence of population size $N$, which is a hyperparameter in AGLDPSO. Generally speaking, larger population size $N$ can maintain population diversity and enhance the exploration ability of algorithm, while smaller population size $N$ can save more FEs in every generation, so as to result in longer evolutionary generations to further improve the accuracy of solutions.

$N$ is tested with five values, 100, 300, 500 (the one used in AGLDPSO), 600, and 1000. The AGLDPSO variant with $N = a$ is called AGLDPSO($a$). For example, the AGLDPSO variant with $N = 100$ is called AGLDPSO(100). The comparison results of AGLDPSO and its variants with different $N$ values on the IEEE CEC2010 test suite are listed in Table S.IV in the supplementary file.

As we can see from Table S.IV in the supplementary file, AGLDPSO variants with larger population size perform generally better than the AGLDPSO variants with smaller population size. Especially, the AGLDPSO variant with the smallest population size, that is, $N = 100$, performs the worst in our test. This may be due to that when solving the large-scale optimization problems, more diversity is needed. Larger population size $N$ can increase the diversity, which is more suitable for large-scale optimization problems. Nevertheless, AGLDPSO variants with too large population size $N$ may lead to large FEs consuming in every generation and therefore, makes the algorithm terminate early with fewer generations. This is not good for the algorithm to search for the global optimum sufficiently. For example, both AGLDPSO(600) and AGLDPSO(1000) perform worse than AGLDPSO on $f_5$, $f_9$, $f_{11}$, $f_{12}$, $f_{14}$, $f_{15}$, $f_{17}$, and $f_{19}$. All in all, AGLDPSO with $N = 500$ performs significantly better than AGLDPSO(100), AGLDPSO(300), AGLDPSO(600), and AGLDPSO(1000) on 20, 15, 13, and 11 functions, respectively, while is only significantly beaten by them on 0, 4, 2, and 7 functions, respectively. Therefore, $N = 500$ is the most suitable parameter for AGLDPSO to achieve a good performance in our test.

## V. Conclusion

This article develops an algorithm called AGLDPSO with the help of ML techniques for large-scale optimization. Two major novel techniques are designed: 1) master–slave multisubpopulation distributed model and 2) AGLS.

Several subpopulations are co-evolved by using the master–slave multisubpopulation distributed model. Compared with other large-scale optimization algorithms with the single population evolution or centralized mechanism, the distributed multisubpopulation co-evolution mechanism will fully exchange the evolutionary information among different populations to further enhance the population diversity. Furthermore, the AGLS based on LSH and LR in AGLDPSO can find an appropriate subpopulation size on different evolutionary states, and further to relieve the sensitivity of parameters. Moreover,

the adaptive subpopulation size can further control the learning granularity effectively and can find a potential balance between diversity and convergence.

Equipped with these two novel techniques, AGLDPSO achieves a promising and satisfying performance when solving the large-scale optimization problems. The comparison results between AGLDPSO and other large-scale optimization algorithms, even the winner of the competition on large-scale optimization, fully show the efficiency and feasibility of AGLDPSO.

## REFERENCES


[1] H. Shayanfar and F. S. Gharehchopogh, "Farmland fertility: A new metaheuristic algorithm for solving continuous optimization problems," *Appl. Soft Comput.*, vol. 71, pp. 728–746, Oct. 2018.

[2] R.-D. Liu, Z.-G. Chen, Z.-J. Wang, and Z.-H. Zhan, "Intelligent path planning for AUVs in dynamic environments: An EDA-based learning fixed height histogram approach," *IEEE Access*, vol. 7, pp. 185433–185446, 2019.

[3] E. H. V. Segundo, V. C. Mariani, and L. S. Coelho, "Design of heat exchangers using Falcon optimization algorithm," *Appl. Thermal Eng.*, vol. 156, pp. 119–144, Jun. 2019.

[4] S. Shadravan, H. R. Naji, and V. K. Bardsiri, "The sailfish optimizer: A novel nature-inspired metaheuristic algorithm for solving constrained engineering optimization problems," *Eng. Appl. Artif. Intell.*, vol. 80, pp. 20–34, Apr. 2019.

[5] C. E. Klein and L. D. S. Coelho, "Meerkats-inspired algorithm for global optimization problems," in *Proc. Eur. Symp. Artif. Neural Netw.*, 2018, pp. 579–584.

[6] E. H. D. V. Segundo, V. C. Mariani, and L. D. S. Coelho, "Metaheuristic inspired on owls behavior applied to heat exchangers design," *Thermal Sci. Eng. Progr.*, vol. 14, Dec. 2019, Art. no. 100431, doi: 10.1016/j.tsep.2019.100431.

[7] A. Mortazavi, V. Togan, and A. Nuhoglu, "Interactive search algorithm: A new hybrid metaheuristic optimization algorithm," *Eng. Appl. Artif. Intell.*, vol. 71, pp. 275–292, May 2018.

[8] J. Pierezan and L. D. S. Coelho, "Coyote optimization algorithm: A new metaheuristic for global optimization problems," in *Proc. IEEE Congr. Evol. Comput.*, 2018, pp. 1–8.

[9] C. E. Klein, V. C. Mariani, and L. D. S. Coelho, "Cheetah based optimization algorithm: A novel swarm intelligence paradigm," in *Proc. Eur. Symp. Artif. Neural Netw.*, 2018, pp. 685–690.

[10] X.-Y. Zhang, J. Zhang, Y.-J. Gong, Z.-H. Zhan, W.-N. Chen, and Y. Li, "Kuhn–Munkres parallel genetic algorithm for the set cover problem and its application to large-scale wireless sensor networks," *IEEE Trans. Evol. Comput.*, vol. 20, no. 5, pp. 695–710, Oct. 2016.

[11] 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.

[12] X.-F. Liu *et al.*, "Historical and heuristic-based adaptive differential evolution," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 49, no. 12, pp. 2623–2635, Dec. 2019.

[13] 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.

[14] 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.*, early access, doi: 10.1109/TEVC.2019.2944180.

[15] H. Zhao *et al.*, "Local binary pattern based adaptive differential evolution for multimodal optimization problems," *IEEE Trans. Cybern.*, early access, doi: 10.1109/TCYB.2019.2927780.

[16] 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.*, early access, doi: 10.1109/TCYB.2019.2937565.

[17] Y. Lin, Y.-S. Jiang, Y.-J. Gong, Z.-H. Zhan, and J. Zhang, "A discrete multiobjective particle swarm optimizer for automated assembly of parallel cognitive diagnosis tests," *IEEE Trans. Cybern.*, vol. 49, no. 7, pp. 2792–2805, Jul. 2019.

[18] X.-W. Luo, Z.-J. Wang, R.-C. Guan, Z.-H. Zhan, and Y. Gao, "A distributed multiple populations framework for evolutionary algorithm in solving dynamic optimization problems," *IEEE Access*, vol. 7, pp. 44372–44390, 2019.

[19] 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.

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

[21] D. Liang, Z.-H. Zhan, Y. Zhang, and J. Zhang, "An efficient ant colony system approach for new energy vehicle dispatch problem," *IEEE Trans. Intell. Transp. Syst.*, early access, doi: 10.1109/TITS.2019.2946711.

[22] S. M. Meerkov and M. T. Ravichandran, "Combating curse of dimensionality in resilient monitoring systems: Conditions for lossless decomposition," *IEEE Trans. Cybern.*, vol. 47, no. 5, pp. 1263–1272, May 2017.

[23] 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.

[24] Y.-F. Zhang and H.-D. Chiang, "A novel consensus-based particle swarm optimization-assisted trust-tech methodology for large-scale global optimization," *IEEE Trans. Cybern.*, vol. 47, no. 9, pp. 2717–2729, Sep. 2017.

[25] Z.-J. Wang, Z.-H. Zhan, and J. Zhang, "Distributed minimum spanning tree differential evolution for multimodal optimization problems," *Soft Comput.*, vol. 23, no. 24, pp. 13339–13349, Mar. 2019.

[26] R. Cheng, Y. Jin, M. Olhofer, and B. Sendhoff, "Test problems for large-scale multiobjective and many-objective optimization," *IEEE Trans. Cybern.*, vol. 47, no. 12, pp. 4108–4121, Dec. 2017.

[27] M. A. Potter and K. A. D. Jong, "A cooperative coevolutionary approach to function optimization," in *Proc. Int. Conf. Parallel Problem Solving Nat.*, 1994, pp. 249–257.

[28] Z. Cai and Z. Peng, "Cooperative coevolutionary adaptive genetic algorithm in path planning of cooperative multi-mobile robot systems," *J. Intell. Robot Syst.*, vol. 33, no. 1, pp. 61–71, Jan. 2002.

[29] F. Van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 225–239, Jun. 2004.

[30] X. Li and X. Yao, "Cooperatively coevolving particle swarms for large scale optimization," *IEEE Trans. Evol. Comput.*, vol. 16, no. 2, pp. 210–224, Apr. 2012.

[31] Y.-J. Shi, H.-F. Teng, and Z.-Q. Li, "Cooperative co-evolutionary differential evolution for function optimization," in *Proc. Int. Conf. Adv. Nat. Comput.*, 2005, pp. 1080–1088.

[32] Z. Yang, K. Tang, and X. Yao, "Differential evolution for high-dimensional function optimization," in *Proc. IEEE Congr. Evol. Comput.*, 2007, pp. 3523–3530.

[33] M. Omidvar, X. Li, Y. Mei, and X. Yao, "Cooperative co-evolution with differential grouping for large scale optimization," *IEEE Trans. Evol. Comput.*, vol. 18, no. 3, pp. 378–393, Jun. 2014.

[34] Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Inf. Sci.*, vol. 178, no. 15, pp. 2985–2999, Aug. 2008.

[35] Z. Yang, K. Tang, and X. Yao, "Multilevel cooperative coevolution for large scale optimization," in *Proc. IEEE Congr. Evol. Comput.*, 2008, pp. 1663–1670.

[36] R. Cheng and Y. Jin, "Social learning particle swarm optimization algorithm for scalable optimization," *Inf. Sci.*, vol. 291, pp. 43–60, Jan. 2015.

[37] R. Cheng and Y. Jin, "A competitive swarm optimizer for large scale optimization," *IEEE Trans. Cybern.*, vol. 45, no. 2, pp. 191–204, Feb. 2015.

[38] K.-C. Tan, Y.-J. Yang, and C.-K. Goh, "A distributed cooperative coevolutionary algorithm for multiobjective optimization," *IEEE Trans. Evol. Comput.*, vol. 10, no. 5, pp. 527–549, Oct. 2006.

[39] 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.

[40] 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.

[41] 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.*, early access, doi: 10.1109/TCYB.2019.2933499.
[42] S.-Z. Zhao, J. J. Liang, P. N. Suganthan, and M. F. Tasgetiren, "Dynamic multi-swarm particle swarm optimizer with local search for large scale global optimization," in *Proc. IEEE Congr. Evol. Comput.*, 2008, pp. 3845–3852.
[43] E. Mjolsness and D. DeCoste, "Machine learning for science: State of the art and future prospects," *Science*, vol. 293, no. 5537, pp. 2051–2055, Sep. 2001.
[44] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, 1995, pp. 1942–1948.
[45] 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.
[46] X. Xia *et al.*, "Triple archives particle swarm optimization," *IEEE Trans. Cybern.*, early access, doi: 10.1109/TCYB.2019.2943928.
[47] Y. H. Li, Z.-H. Zhan, S. J. Lin, J. Zhang, and X. N. Luo, "Competitive and cooperative particle swarm optimization with information sharing mechanism for global optimization problems," *Inf. Sci.*, vol. 293, no. 1, pp. 370–382, Feb. 2015.
[48] Q. Yang *et al.*, "Segment-based predominant learning swarm optimizer for large-scale optimization," *IEEE Trans. Cybern.*, vol. 47, no. 9, pp. 2896–2910, Sep. 2017.
[49] Q. Yang, W.-N. Chen, J. D. Deng, Y. Li, T. L. Gu, and J. Zhang, "A level-based learning swarm optimizer for large-scale optimization," *IEEE Trans. Evol. Comput.*, vol. 22, no. 4, pp. 578–594, Aug. 2018.
[50] Z.-H. Zhan, J. Zhang, Y. Li, and H. S.-H. Chung, "Adaptive particle swarm optimization," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 39, no. 6, pp. 1362–1381, Dec. 2009.
[51] W. J. Yu *et al.*, "Differential evolution with two-level parameter adaptation," *IEEE Trans. Cybern.*, vol. 44, no. 7, pp. 1080–1099, Jul. 2014.
[52] Z. J. Wang, H. Jin, and J. Zhang, "Adaptive distributed differential evolution," *IEEE Trans. Cybern.*, early access, doi: 10.1109/TCYB.2019.2944873.
[53] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimension via hashing," in *Proc. Int Conf. Very Large Data Bases*, 1999, pp. 518–529.
[54] Y.-H. Zhang, Y.-J. Gong, H.-X. Zhang, T.-L. Gu, and J. Zhang, "Toward fast Niching evolutionary algorithms: A locality sensitive hashing-based approach," *IEEE Trans. Evol. Comput.*, vol. 21, no. 3, pp. 347–362, Jun. 2017.
[55] K. Tang, X. Li, P. Suganthan, Z. Yang, and T. Weise, "Benchmark functions for the CEC 2010 special session and competition on large scale global optimization," Nat. Inspired Comput. Appl. Lab., Univ. Sci. Technol. China, Hefei, China, Rep., 2009.
[56] X. Li, K. Tang, M. N. Omidvar, Z. Yang, and K. Qin, "Benchmark functions for the CEC 2013 special session and competition on large scale global optimization," Evol. Comput. Mach. Learn. Subpopulation, RMIT Univ., Melbourne, VIC, Australia, Rep., 2013.
[57] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 3–18, Mar. 2011.
[58] D. Molina, M. Lozano, and F. Herrera, "MA-SW-chains: Memetic algorithm based on local search chains for large scale continuous global optimization," in *Proc. IEEE Congr. Evol. Comput.*, 2010, pp. 1–8.

AQ4

**Zi-Jia Wang** (Student Member, IEEE) received the B.S. degree in automation from Sun Yat-sen University, Guangzhou, China, in 2015, where he is currently pursuing the Ph.D. degree.

His current research interests include evolutionary computation algorithms like differential evolution, particle swarm optimization, and their applications in design and optimization such as cloud computing resources scheduling.

**Zhi-Hui Zhan** (Senior Member, IEEE) received the bachelor's and Ph.D. degrees from the Department of Computer Science, Sun Yat-sen University, Guangzhou, China, in 2007 and 2013, respectively.

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

Dr. Zhan's doctoral dissertation was awarded the IEEE Computational Intelligence Society Outstanding Ph.D. Dissertation and the China Computer Federation Outstanding Ph.D. Dissertation. He was a 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 of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and *Neurocomputing*.

**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, and 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, Bloomington, MN, USA, where he designed the diagnostic software to detect the manufacture faults of the VLSI chips in the Cyber 430 machine. He later joined the Bell Northern Research Canada, Ottawa, ON, Canada, as a Member of the Scientific Staff. In 1990, he joined the Department of Electronic Engineering, City University of Hong Kong, Hong Kong, as a Lecturer. He is currently a Professor with the Department of Computer Science. His research areas are in pattern recognition, evolutionary computations, and video analytics.

Prof. Kwong has been the Vice President of the IEEE Systems, Man and Cybernetics for conferences and meetings since 2014. He was elevated to IEEE fellow for his contributions on optimization techniques for cybernetics and video coding in 2014. He has been an IEEE Distinguished Lecturer for the IEEE SMC Society since March 2017.

**Hu Jin** (Senior Member, IEEE) received the B.E. degree in electronic engineering and information science from the University of Science and Technology of China, Hefei, China, in 2004, and the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2006 and 2011, respectively.

From 2011 to 2013, he was a Postdoctoral Fellow with the University of British Columbia, Vancouver, BC, Canada. From 2013 to 2014, he was a Research Professor with Gyeongsang National University, Tongyeong, South Korea. Since 2014, he has been with the Division of Electrical Engineering, Hanyang University, Ansan, South Korea, where he is currently an Associate Professor. His research interests include wireless communications, Internet of Things, and machine learning.

**Jun Zhang** (Fellow, IEEE) received the Ph.D. degree from the City University of Hong Kong, Hong Kong, in 2002.

He is currently a Visiting Professor with Hanyang University, Ansan, South Korea. His current research interests include computational intelligence, cloud computing, high-performance computing, operations research, and power electronic circuits.

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