A Distributed Swarm Optimizer With Adaptive Communication for Large-Scale Optimization

Qiang Yang, *Member, IEEE*, Wei-Neng Chen, *Senior Member, IEEE*, Tianlong Gu, Huaxiang Zhang, Huaqiang Yuan, Sam Kwong, *Fellow, IEEE*, and Jun Zhang, *Fellow, IEEE*

***Abstract*—Large-scale optimization with high dimensionality and high computational cost becomes ubiquitous nowadays. To tackle such challenging problems efficiently, devising distributed evolutionary computation algorithms is imperative. To this end, this paper proposes a distributed swarm optimizer based on a special master–slave model. Specifically, in this distributed optimizer, the master is mainly responsible for communication with slaves, while each slave iterates a swarm to traverse the solution space. An asynchronous and adaptive communication strategy based on the request–response mechanism is especially devised to let the slaves communicate with the master efficiently. Particularly, the communication between the master and each slave is adaptively triggered during the iteration. To aid the slaves to search the space efficiently, an elite-guided learning strategy is especially designed via utilizing elite particles in the current swarm and historically best solutions found by different slaves to guide the update of particles. Together, this distributed optimizer asynchronously iterates multiple swarms to collaboratively seek the optimum in parallel. Extensive experiments on a widely used large-scale benchmark set substantiate that the distributed optimizer could: 1) achieve competitive effectiveness in terms of solution quality as compared to the state-of-the-art large-scale methods; 2) accelerate the execution of the algorithm in comparison with the sequential one and obtain almost linear speedup as the number of cores increases; and 3) preserve a good scalability to solve higher dimensional problems.**

***Index Terms*—Distributed evolutionary algorithms, elite-guided learning (EGL), high-dimensional problems, large-scale optimization, particle swarm optimization (PSO).**

Manuscript received October 12, 2018; revised January 20, 2019; accepted March 1, 2019. Date of publication April 9, 2019; date of current version June 16, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61622206 and Grant 61876111, in part by the Natural Science Foundation of Guangdong under Grant 2015A030306024, in part by the Science and Technology Plan Project of Guangdong Province under Grant 2018B050502006, and in part by the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing. This paper was recommended by Associate Editor P. P. Angelov. *(Corresponding author: Wei-Neng Chen.)*

Q. Yang is with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China, also 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.

W. N. Chen and J. Zhang are 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, Guangzhou 510006, China (e-mail: cwnraul634@aliyun.com).

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

H. Zhang is with the School of Information Science and Engineering, Shandong Normal University, Jinan 250014, China.

H. Yuan is with the School of Computer Science and Network Security, Dongguan University of Technology, Dongguan 523808, China.

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

This paper 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 paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCYB.2019.2904543

I. Introduction

LARGE-SCALE optimization with high dimensionality and high computational cost has become more and more common in many research domains and engineering [1]–[4] in the era of big data [5]. Faced with such difficult problems, traditional population-based metaheuristic algorithms executed in serial would take hours or even days to find the optimum [6]. This mainly results from two aspects. On the one hand, due to the high time complexity of these problems, it takes a long time to evaluate the fitness of an individual. On the other hand, the solution space of these problems increases exponentially [7] and, thus, to traverse such vast space, population-based metaheuristics, such as particle swarm optimization (PSO) algorithms [8]–[10] and differential evolution (DE) algorithms [11]–[13], need to consume a considerably large number of fitness evaluations to achieve satisfactory performance. With these two challenges, the execution time of sequential metaheuristics is prolonged rapidly when dealing with such problems and such time may even become unbearable [14].

Fortunately, population-based metaheuristics generally preserve inherent parallelism and, thus, developing parallel and distributed metaheuristics is an efficient way to tackle optimization problems with high computational cost. Recently, this research direction has attracted increasing attention in evolutionary computation community, leading to the development of parallel metaheuristics [6]. However, most existing studies directly extended traditional sequential metaheuristics designed for low-dimensional problems in distributed environments [6]. Faced with high-dimensional problems, traditional metaheuristics dramatically lose their effectiveness and efficiency [15] and, thus, it is not effective to directly employ existing distributed metaheuristics to cope with high-dimensional problems. Consequently, to locate the optimum in acceptable time, it is significant to develop distributed metaheuristics with effective update mechanisms and efficient communication schemes, which are suitable to solve large-scale optimization problems.

In the literature, to cope with large-scale optimization effectively, many novel evolution mechanisms have been devised. Broadly, these evolution mechanisms can be classified into two categories [15]: 1) cooperative coevolution (CC) mechanisms [16]–[18] and 2) novel update mechanisms for traditional metaheuristics [8]–[10]. The former aim to decompose a high-dimensional problem into several smaller subproblems and then optimize each subproblem separately [16]. In contrast, the latter still optimize all dimensions together like traditional metaheuristics [19], [20], but incorporate new update strategies to preserve high diversity [9], [10].

Though most existing evolution mechanisms have shown promising performance in coping with large-scale optimization, they are especially designed for sequential environments but not for distributed environments. Despite that some evolution mechanisms can be adapted to distributed environments based on existing distributed models, they are still confronted with many limitations. For instance, a few studies have intuitively adapted some cooperative coevolutionary algorithms (CCEAs) to distributed computing [21]. However, they are not capable of solving problems with many interacting variables effectively (e.g., fully nonseparable problems). Besides, a few studies have extended some metaheuristics optimizing all variables together to distributed environments based on the master–slave distributed model [6]. Nevertheless, such an adaptation would cause a huge communication burden as will be demonstrated in the experiments in Section IV.

Consequently, developing distributed metaheuristics with effective evolution mechanisms and efficient communication for large-scale optimization still deserves further investigation. To this end, this paper proposes a distributed swarm optimizer with a new update scheme and an adaptive communication strategy to solve large-scale optimization problems efficiently.

Specifically, we adopt a special master–slave model, where the master is responsible for the communication with the slaves, while each slave iterates a swarm to traverse the high-dimensional space. To let the slaves communicate with the master efficiently, an adaptive communication strategy based on the request–response mechanism is specifically designed. In particular, the communication between the master and each slave is triggered adaptively according to the search state of the associated swarm during the iteration. By this means, each slave communicates with the master independently and asynchronously and, thus, little waiting time exists during the communication between the master and the slaves.

Besides, to aid the swarms in the slaves to traverse the high-dimensional space efficiently in the distributed environment, we especially design an elite-guided learning (EGL) strategy, which utilizes elite particles in the current swarm and historically best solutions found by different slaves to guide the update of particles. In this way, high swarm diversity could be preserved to let particles escape from local areas during the iteration.

Altogether, we name this distributed optimizer as distributed elite-guided learning swarm optimizer (DEGLSO). With the above two strategies, this distributed optimizer asynchronously iterates multiple swarms in parallel to seek the optimum of a high-dimensional problem. To verify its effectiveness and efficiency, experiments are conducted on the CEC'2013 large-scale benchmark set [22] to evaluate its performance in terms of solution quality, execution time, speedup, and scalability via comparing with state-of-the-art large-scale metaheuristics.

The rest of this paper is organized as follows. Section II reviews the related metaheuristics. Then, the devised distributed swarm optimizer is elucidated in Section III. In Section IV, a series of experiments is conducted to verify its effectiveness and efficiency. At last, Section V concludes this paper.

II. Related Work

A. Parallel and Distributed Metaheuristics

In the literature, many parallel and distributed metaheuristics have been developed to tackle optimization problems with high time complexity based on different distributed models [6]. Broadly, distributed models used in existing distributed metaheuristics can be classified into four main categories [6]: 1) master–slave models [23]; 2) island models [24], [25]; 3) cellular models [26], [27]; and 4) hierarchical models [28].

The master–slave model maintains only one master process but several slave processes [23]. Originally, the master is only responsible for the update of the population, while the slaves are to evaluate the fitness of individuals. After updating the population, the master sends several individuals to each slave, and the slaves compute the fitness of the allocated individuals and then send the calculated fitness back to the master. To improve the efficiency of this model, researchers extended the original model to a coarse-grained one [29], where each slave iterates a subpopulation and sends the global best position found so far to the master, while the master receives the global best positions from all slaves, determines the best one and then sends it to all slaves. As for the communication between the master and the slaves, most master–slave-based parallel metaheuristics adopt synchronous communication schemes [6]. A few of this kind of distributed metaheuristics [30] also adopt asynchronous communication strategies.

The island model [24], [25] is a spatially distributed model. In this model, each island maintains a subpopulation and communicates with each other using a certain migration mechanism. Particularly, this model seriously relies on the adopted migration mechanism, including the migration frequency, the selection of the immigrants, etc. [31]. Except for the migration mechanism, this distributed model is also very sensitive to the topologies that arrange the islands [6]. In the original island model, islands are arranged by a complete graph, which would cause a high communication burden. Recently, researchers have utilized network topologies, such as ring and torus, to arrange islands to improve the efficiency of this model [32], [33]. As for the communication between islands, both synchronous and asynchronous communication strategies have been utilized in the literature. For synchronous island models, the global best solution found by each island is exchanged periodically at a specific interval of

generations [33]. For asynchronous ones, each island could receive information sent from other islands as soon as it is ready [34].

The cellular model [26], [27] maintains only one population but arranges individuals onto grids with one grid occupying one or several individuals. In this model, individuals are updated by their corresponding neighbors determined by the topology that arranges individuals onto grids. In the literature, a lot of effort has been devoted to develop cellular-based distributed metaheuristics with different topologies, such as linear topology [35], toroid topology [36], and regular lattices [26]. Similar to island models, both synchronous and asynchronous communication strategies have been utilized in this model [37].

The hierarchical model [28] generally combines two or more distributed models stated above hierarchically, so that the advantages of different models can be inherited. Three kinds of hybrids exist. The first is the island–master–slave hybrid model [38], where the first layer adopts the island model to evolve multiple subpopulations and the second layer utilizes the master–slave model operated on each island in the first layer. Another is the island–cellular hybrid model [28], which is similar to the first one. The only difference is that the second layer takes advantage of the cellular model and thus individuals in each island in the first layer are arranged onto the grids of the cellular model. The other is the island–island hybrid model [39], where the second layer uses another island model and thus the population in each island in the first layer is further divided into subpopulations evolved by the islands in the second layer.

Based on these distributed models, many distributed metaheuristics have been designed via incorporating different traditional metaheuristics [6]. However, most of them only remain efficient and effective in low-dimensional space. When handling high-dimensional problems, their effectiveness and efficiency dramatically degrade [15].

B. Large-Scale Optimization

In the literature, the evolution mechanisms for large-scale optimization mainly lie in the two following aspects [15].

1) Cooperative Coevolution Mechanisms: CCEAs utilize the divide-and-conquer method to partition a high-dimensional problem into several smaller subproblems and then optimize each subproblem separately [16], [40]. Since interacted variables generally interfere with each other during the optimization, the decomposition strategy has been proven to play a crucial role in CCEAs [16], [41], [42].

Theoretically, the ideal decomposition strategy is to group interdependent variables into the same subproblem. However, in most cases, the prior knowledge about variable interdependency in an optimization problem is not available. As a result, current research on CCEAs mainly concentrates on devising effective decomposition strategies to divide high-dimensional problems as accurately as possible. Broadly, existing decomposition strategies can be classified into two categories [15]: 1) dynamic decomposition strategies [42]–[44] and 2) static decomposition strategies [16], [45].

Dynamic decomposition strategies are usually executed in each generation along with optimizers and, thus, the variable decomposition may be different in different generations [15]. In the literature, two kinds of dynamic decomposition strategies exist, namely random-based decomposition [17], [42], [46] and learning-based decomposition [44], [47], [48]. The former strategies randomly divide variables into groups without taking variable interaction into consideration [17], [42], [46] and thus they perform poorly on problems with more than two interdependent variables. The latter strategies make use of evolutionary information to learn variable interdependency and then divide variables into groups [44], [47]. For instance, in [47], an adaptive variable partition technique was developed based on the correlation coefficients of the top half individuals.

Unlike dynamic decomposition, static decomposition is executed before optimization based on variable interaction detection [16] and in the optimization stage, the variable decomposition is fixed. One typical static grouping approach is the differential grouping strategy (DG) [16], which utilizes partial difference between functional values to detect variable interdependency. However, this approach can only detect direct linkages between variables. To detect direct and indirect variable dependency simultaneously, extended DG (XDG) [49] and global DG (GDG) [45] were further developed based on DG. Though the above DG variants could partition variables into groups satisfactorily, they cost up to $O(D^2)$ (D is the dimension size) fitness evaluations in the decomposition stage. Thus, if only given limited total fitness evaluations, the number of fitness evaluations used for optimization is greatly reduced. To alleviate this predicament, many attempts have been made to reduce the cost of fitness evaluations in the decomposition stage. For instance, a recursive DG method (RDG) [50] was developed based on the idea of binary search. Particularly, it reduces the used fitness evaluations from $O(D^2)$ to $O(Dlog(D))$.

2) Novel Update Mechanisms for Traditional Metaheuristics: In this direction, plenty of novel update schemes have been developed to aid traditional metaheuristics to preserve high search diversity during the optimization, so that local traps can be avoided. In this section, we only review representative variants of PSO and DE in tackling large-scale optimization, because they are the most developed ones in this direction [15].

With respect to PSO, in the classical update scheme, each particle is guided by its own experience and the social experience of the swarm with the following update formula:

$$v_i^d \leftarrow wv_i^d + c_1 r_1 \left(pbest_i^d - x_i^d\right) + c_2 r_2 \left(gbest^d - x_i^d\right) \quad (1)$$

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

where $\boldsymbol{x}_i = [x_i^1, \ldots, x_i^d, \ldots, x_i^D]$ and $\boldsymbol{v}_i = [v_i^1, \ldots, v_i^d, \ldots, v_i^D]$ are the position and the velocity of the ith particle, respectively. D is the dimension size. $\boldsymbol{pbest}_i = [pbest_i^1, \ldots, pbest_i^d, \ldots, pbest_i^D]$ and $\boldsymbol{gbest} = [gbest^1, \ldots, gbest^d, \ldots, gbest^D]$ are the personal best position of the ith particle and the global best position found by the swarm,

respectively. As for the parameters, w is the inertia weight, c_1 and c_2 are two acceleration coefficients, and r_1 as well as r_2 is uniformly randomized within [0, 1]. When solving large-scale optimization problems, this classic update scheme usually loses its effectiveness. Oldewage [51] found that the velocity clamping and the parameter settings in (1) have great influence on PSO in solving high-dimensional problems.

However, compared with parameters, the update strategy generally plays a more important role in PSO [10]. Therefore, to adapt PSO to solve large-scale optimization problems, many researchers have developed a variety of novel update schemes. For instance, Zhao *et al.* [52] developed a dynamic multiswarm PSO along with the quasi-Newton method as the local search method, leading to DMS-L-PSO. Specifically, this optimizer randomly partitions the whole swarm into smaller subswarms in each generation. Hsieh *et al.* [53] proposed an efficient population utilization strategy for PSO, leading to EPUS-PSO. In particular, a population size managing approach and a solution sharing approach were devised to improve the search abilities of particles in EPUS-PSO. García-Nieto and Alba [54] devised a velocity modulation method and a restarting mechanism for PSO. With the aid of these two techniques, the developed PSO variant could effectively avoid premature convergence and redirect particles to promising areas in the search space. A multiswarm PSO based on a competition scheme was devised in [55]. In this PSO variant, two swarms are maintained and particles in the two swarms compete with each other. Then, each loser is updated via a convergence strategy, while each winner is updated by a mutation strategy.

The above studies mainly adopt multipopulation strategies or restarting mechanisms to promote the search diversity of PSO, so that falling into local traps could be avoided. In particular, they all utilize the historically best positions, such as *pbest*, *nbest*, and *gbest*, to update particles as the classical PSO [56], [57]. Nevertheless, these best positions may remain unchanged for many generations during the iteration and, thus, have great limitations in diversity maintenance [10].

To further promote the diversity of the swarm, some researchers introduced new exemplars to replace *pbest*, *nbest*, or *gbest* to guide the update of particles. For instance, Cheng and Jin [10] developed a competitive swarm optimizer (CSO). In this optimizer, particles are randomly arranged into pairs and the paired particles compete with each other. Then, the loser is guided by the winner, while the winner is not updated. In particular, the loser is updated as follows:

$$v_l^d \leftarrow r_1 v_l^d + r_2\left(x_w^d - x_l^d\right) + \phi r_3\left(\bar{x}^d - x_l^d\right) \tag{3}$$

$$x_l^d \leftarrow x_l^d + v_l^d \tag{4}$$

where $\boldsymbol{x}_l = [x_l^1, \ldots, x_l^d, \ldots, x_l^D]$ and $\boldsymbol{v}_l = [v_l^1, \ldots, v_l^d, \ldots, v_l^D]$ are the position and the velocity of the loser, respectively. $\boldsymbol{x}_w = [x_w^1, \ldots, x_w^d, \ldots, x_w^D]$ is the position of the corresponding winner of the loser and $\bar{\boldsymbol{x}} = [\bar{x}^1, \ldots, \bar{x}^d, \ldots, \bar{x}^D]$ is the mean position of the whole swarm. r_1, r_2, and r_3 are three random numbers uniformly generated within [0, 1]. ϕ is a control parameter within [0, 1], which is in charge of the influence of $\bar{\boldsymbol{x}}$.

In addition, in [58], a level-based learning swarm optimizer (LLSO) was proposed, which shares the similar update formula as CSO in (3). Different from CSO, particles in LLSO are divided into different levels according to their fitness values and then the ones in lower levels are guided by two superior particles randomly selected from two different higher levels. Since particles are updated generation by generation, the exemplars of updated particles in both CSO and LLSO are different in different generations. Therefore, both of them could preserve high diversity during the optimization, leading to their promising performance in handling large-scale optimization.

For DE, various novel mutation schemes have been designed to adapt DE to handle high-dimensional problems. For example, Wang *et al.* devised a generalized opposition-based learning strategy and hybridized it into the classical DE to update the population, leading to a generalized opposition DE (GODE) [13]. Weber *et al.* [12] developed a shuffle or update parallel DE (SOUPDE) via dividing individuals into subpopulations randomly with a probability during the optimization. Zhao *et al.* [11] hybridized the self-adaptive strategy in JADE [59] and a modified multitrajectory search algorithm to solve large-scale optimization problems, leading to a self-adaptive DE named SaDE-MMTS. In [60], an elite opposition-based DE (EOBDE) was devised by employing the opposite learning strategy to some selected elites with a certain probability. Ali *et al.* [61] first divided the population into independent subgroups, and then utilized different mutation strategies to update the subgroups. With this update scheme, the population diversity of DE could be largely boosted.

Though the above methods have shown good performance in solving large-scale optimization problems, they are designed for serial environments, but not for distributed environments. A few studies intuitively adapted some large-scale metaheuristics to distributed environments [21], using the aforementioned distributed models [6]. However, such adaption would cause a huge communication burden. To alleviate this predicament, this paper proposes a distributed swarm optimizer (named DEGLSO) with a new update scheme and an adaptive communication strategy to deal with large-scale optimization.

III. Distributed Swarm Optimizer

In this section, the proposed distributed swarm optimizer DEGLSO is elaborated in detail. Specifically, we first elucidate its framework and the EGL strategy in Section III-A. Section III-B states the adaptive communication strategy designed for efficient information exchange. In Section III-C, the implementation of DEGLSO is presented. At last, to make comparisons, the serial version of DEGLSO, named SEGLSO, is presented in Section III-D.

A. DEGLSO

1) Framework: The framework of DEGLSO is presented in Fig. 1. In this distributed optimizer, a special master–slave model is adopted. Particularly, in this model, only one master process exists, which is mainly in charge of information

exchange, and multiple slave processes, each of which iterates a small swarm, cooperate with each other to search the high-dimensional space.

Concretely, the master maintains an archive $\boldsymbol{A}$ to store the global best positions found by the slaves. It does nothing but takes charge of information exchange among slaves. In terms of the slaves, each of them iterates a swarm based on the proposed EGL to be described next to find the optimum. In this paper, the archive size in the master and the swarm size in the slaves are set the same and denoted as NP. The details of the master and the slaves are elaborated as follows.

1) As for the master, after it receives the *gbest*s sent from the slaves, it places them into the archive $\boldsymbol{A}$. Once $\boldsymbol{A}$ is full, a random individual is first selected from the archive and is then compared with the received one. If the received individual is better, it replaces the selected one; otherwise, it is discarded. Such a strategy not only facilitates high diversity maintenance, but also takes little time in updating $\boldsymbol{A}$, which is beneficial for reducing the computational cost of the master. It should be mentioned that instead of only using the best among these *gbest*s sent from the slaves like in [29], an archive of *gbest*s from the slaves are maintained for the sake of high diversity preservation. Specifically, by means of preserving multiple *gbest*s sent from slaves, the master could send different information to different slaves, and thus affords diverse information exchange between slaves. In this manner, the slaves could avoid receiving uniform information and thus they have great chance to search the high-dimensional space in different directions, avoiding converging to the same area. The usefulness of the archive $\boldsymbol{A}$ in the master will be verified in the experiments in Section III in the supplementary material.
2) As for the slaves, each of them also maintains an archive $\boldsymbol{P}$, whose size is set to M, to store the received individuals from the master. Once the archive of a slave is full, the received individual is compared with the worst one in $\boldsymbol{P}$. If the received individual is better, it replaces the worst one; otherwise, it is abandoned. Such a maintenance mechanism could ensure the archive of each slave always keeps the most promising individuals found historically by the slaves, which may facilitate fast convergence. It should be mentioned that instead of using the received individual from the master to replace the worst one in the swarm in the corresponding slave like in [29], an archive is utilized to store the received individuals and then is used in EGL to update the swarm. In this way, different exchanged information from other slaves could be preserved to aid the search of the swarm. Therefore, high diversity in exemplar selection could be maintained and high search diversity could be preserved during the optimization. Experiments in Section III in the supplementary material will verify the usefulness of the archives in the slaves.

Remark 1: It should be noticed that the archive technique has been widely employed to assist metaheuristics to tackle various optimization problems, like single-objective optimization problems [59], [62]; dynamic optimization problems [63], [64]; and multiobjective optimization problems [65], [66]. In single-objective optimization, an archive is generally utilized to store promising individuals during the optimization [59], [62]. By taking advantage of the information stored in the archive, either the diversity or the convergence of metaheuristics could be enhanced [59], [62]. In dynamic optimization, an archive is usually maintained to store the best solutions found when the environment change occurs [63], [64]. The archive is utilized mainly to assist metaheuristics to react to the change of the environment quickly [63], [64]. In multiobjective optimization, an archive is usually maintained to store the nondominated solutions [65], [66]. In some cases, these archives are also updated by a certain multiobjective metaheuristic [65], [66].

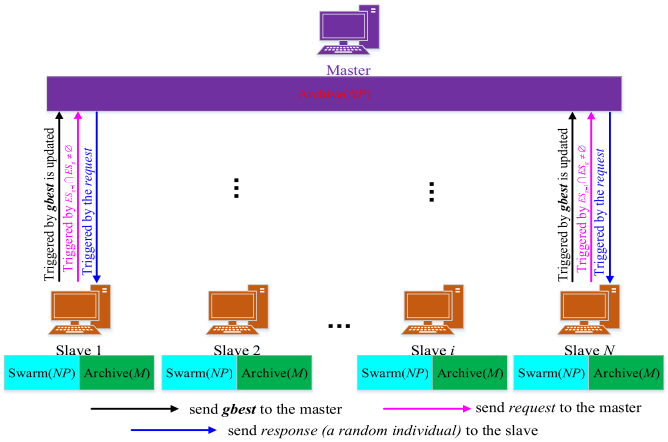

Fig. 1. Framework of DEGLSO.

Different from existing archive techniques, this paper maintains multiple archives in the distributed environment. Specifically, one archive is maintained in the master to store the *gbest*s found so far by all slaves, and one archive is maintained in each slave to store the introduced information from the master. The archive in the master is mainly utilized to realize the communication between slaves, and it provides diverse information exchange between slaves, which is beneficial for the slaves to avoid converging to the same area. The archive in each slave is mainly for preserving diverse promising information introduced from the master, and it provides extra diverse exemplar selection for the update of particles in EGL. Overall, we can see that these two kinds of archives are mainly utilized to promote the diversity of the distributed swarm optimizer. The effectiveness of these two kinds of archives will be verified in Section III in the supplementary material.

2) Elite-Guided Learning: To help the swarms in the slaves traverse the high-dimensional space efficiently in the distributed environment, we devise an EGL strategy to guide the update of particles in each slave.

According to Darwin's "survival-of-the-fittest" principle [67], [68], the top fittest individuals, namely elites, generally preserve greater chances to survive and more valuable evolutionary information than others. Inspired from this, we divide the swarm in each slave into two separate

sets: 1) the elite set ***ES*** containing M elites in the swarm and 2) the nonelite set ***NES*** consisting of the rest (NP-M) particles. Since the elites in ***ES*** are the best particles in the current swarm, they preserve the most useful information to guide the swarm. Thus, they directly enter the next generation. This matches Darwin's "survival-of-the-fittest" principle, and is helpful for protecting promising information from being lost or weakened during the optimization. Therefore, only the (NP-M) particles in ***NES*** are updated in each generation.

Since the elites in ***ES*** preserve the most useful information to update the swarm, they can be utilized to guide the update of the particles in ***NES***. Therefore, the update of these particles is

$$v_i^d \leftarrow r_1 v_i^d + r_2\left(x_{el_{\text{rand}}}^d - x_i^d\right) + \phi r_3\left(\hat{x}_{ES}^d - x_i^d\right) \tag{5}$$

$$x_i^d \leftarrow x_i^d + v_i^d \tag{6}$$

where $\boldsymbol{x}_i = [x_i^1, \ldots, x_i^d, \ldots, x_i^D]$ and $\boldsymbol{v}_i = [v_i^1, \ldots, v_i^d, \ldots, v_i^D]$ are the position and velocity of the ith particle in ***NES*** respectively; $\boldsymbol{x}_{el_{\text{rand}}} = [x_{el_{\text{rand}}}^1, \ldots, x_{el_{\text{rand}}}^d, \ldots, x_{el_{\text{rand}}}^D]$ is an elite randomly selected from $\boldsymbol{ES} \cup \boldsymbol{P}$, and is better than $\boldsymbol{x}_i$; and $\hat{\boldsymbol{x}}_{ES} = [\hat{x}_{ES}^1, \ldots, \hat{x}_{ES}^d, \ldots, \hat{x}_{ES}^D]$ is the means position of the elites and is computed as follows:

$$\hat{x}_{ES}^d = \frac{1}{M}\sum_{j=1}^{M} x_{ES_j}^d \tag{7}$$

where $\boldsymbol{x}_{ES_j} = [x_{ES_j}^1, \ldots, x_{ES_j}^d, \ldots, x_{ES_j}^D]$ is the jth elite in ***ES***, r_1, r_2, and r_3 are three uniformly generated numbers in [0, 1], ϕ in charge of the influence of $\hat{\boldsymbol{x}}_{ES}$ is a control parameter in [0, 1].

In (5), the following techniques should deserve attention.

1) The first exemplar $\boldsymbol{x}_{el_{\text{rand}}}$ in (5) is randomly selected from $\boldsymbol{ES} \cup \boldsymbol{P}$ not just from ***ES***. For one thing, with this technique, the useful information from other slaves, which is stored in ***P***, could be made full use of to update the swarm in the slave. For another, $2M$ candidate exemplars could be potentially utilized to guide the update of these particles, and thus high diversity may be maintained.
2) The selected exemplar $\boldsymbol{x}_{el_{\text{rand}}}$ should be better than $\boldsymbol{x}_i$. By this means, each $\boldsymbol{x}_i$ in ***NES*** is always guided by better particles, and thus could approach to promising areas fast, which facilitates fast convergence. Therefore, if the selected $\boldsymbol{x}_{el_{\text{rand}}}$ is worse than $\boldsymbol{x}_i$ (which only may occur when the selected exemplar comes from ***P***), a new $\boldsymbol{x}_{el_{\text{rand}}}$ is randomly selected from $\boldsymbol{ES} \cup \boldsymbol{P}$ until it is better than $\boldsymbol{x}_i$.
3) As for another exemplar in (5), the particles in ***NES*** share the social knowledge via the mean position of the elites in ***ES***. Since the elites in ***ES*** in two successive generations may be different, the mean position $\hat{\boldsymbol{x}}_{ES}$ may be different in different generations. This is beneficial for diversity preservation. Besides, the mean position $\hat{\boldsymbol{x}}_{ES}$ of these elites can be a good distribution estimation of the swarm in the current generation. Utilizing it as the social exemplar is beneficial for finding promising areas fast.

In summary, the devised EGL can potentially let DEGLSO compromise diversity maintenance and fast convergence well to search the high-dimensional space. Such a good compromise will be verified in Section III in the supplementary material.

As for the parameters, only two extra parameters (the number of elites M and the control parameter ϕ) are introduced in DEGLSO. Taking deep insight into the influence of ϕ on DEGLSO, we find that a large value of ϕ could promote the influence of $\hat{\boldsymbol{x}}_{ES}$ on the update of particles, which is beneficial for preventing the updated particle from being greedily attracted by the selected elite $\boldsymbol{x}_{el_{\text{rand}}}$. Such prevention is helpful when $\boldsymbol{x}_{el_{\text{rand}}}$ falls into local areas. Nevertheless, a small value of ϕ could weaken the aforementioned prevention, which is profitable when $\boldsymbol{x}_{el_{\text{rand}}}$ is close to the globally optimal areas and thus exploitation is strongly needed. Together, we can see that a large ϕ is preferred at the early optimization stages when the swarm explores the search space and a small ϕ is needed at the late stages when the swarm exploits the searching areas.

Bearing the above consideration in mind and inspired from the linear adaption of the inertia weight in PSO [56], we design a linear adjustment strategy for ϕ, which is defined as follows:

$$\phi_g = 0.5\left(1 - \frac{g}{G_{\max}}\right) \tag{8}$$

where g is the current generation index and $G_{\max}$ is the maximal number of generations.

Overall, we can see that since the elites in ***ES*** directly enter the next generation, the most promising information is preserved. After the particles in ***NES*** are updated, new better particles could become elites and enter ***ES***, while those obsolete "elites" in the last generation have to walk out of ***ES*** and go into ***NES*** and then will be updated. As a result, elites in ***ES*** become better and better during the iteration, and thus they may converge to the optimum of the optimized problem.

Remark 2: Compared with the classical PSO (1), the recently proposed CSO (3) and LLSO, the proposed EGL (5) differs from them in the following four aspects.

First, from the viewpoint of the frameworks of these PSO variants, EGL, CSO, and LLSO are totally different from the classical PSO. Instead of using historically best positions to guide the update of the swarm in PSO, the three PSO variants directly utilize particles in the current swarm to guide the update of the swarm. Specifically, in CSO, particles are randomly paired together and then the loser in the paired particles is guided by the winner, while the winner is not updated and directly enters the next generation. In LLSO, particles are divided into multiple levels according to their fitness. Then, particles in lower levels are guided by those from higher levels. In DEGLSO, particles in each slave are divided into two separate sets (namely the elite set ***ES*** and the nonelite set ***NES***) according to their fitness. Besides, only particles in ***NES*** are updated by learning from those in ***ES*** and the solutions stored in the archive ***P***, which are introduced from the master.

Second, as for the first exemplar, DEGLSO, CSO, and LLSO utilize superior particles in the current swarm to update inferior ones, while the classical PSO utilizes the personal best position (*pbest*) to guide the update of one particle. On the one

side, superior particles may be different in different generations; on the other side, the first exemplar guiding the update of one inferior particle is randomly selected from superior ones. Thus, the selection diversity of the first exemplar in DEGLSO, CSO, and LLSO is much higher than that in PSO, because *pbest* may remain unchanged for many generations especially in the later stage of the optimization. Compared with CSO and LLSO, DEGLSO further promotes the selection diversity of the first exemplar via selecting it from the archive $\boldsymbol{P}$ in each slave, which preserves useful information collected from other slaves.

Third, with respect to the second exemplar, both DEGLSO and CSO share the social knowledge via mean positions of either the whole population (CSO) or the elites (DEGLSO), while LLSO still adopts superior particles in the current swarm to update particles. Nevertheless, the classical PSO shares the social knowledge via the global best position (*gbest*) found so far by the swarm. Unlike *gbest*, which may remain unchanged in many generations, the mean positions in both DEGLSO and CSO and superior particles in LLSO may be different in different generations due to the update of particles. Therefore, the second exemplar in DEGLSO, CSO, and LLSO could assist them to maintain higher diversity than the classical PSO. Unlike CSO and LLSO, DEGLSO utilizes the mean position of the elites in the current swarm as the second exemplar. Particularly, this mean position can be taken as a good distribution estimation of the current swarm. Utilizing it as the second exemplar may be beneficial for the optimizer to find promising areas fast.

Fourth, the proposed EGL in (5) is specifically designed for distributed environments. In particular, the division of the swarm into $\boldsymbol{ES}$ and $\boldsymbol{NES}$ makes it possible to devise an adaptive communication strategy for efficient information exchange between slaves, which will be introduced in the next section.

Remark 3: It deserves attention that the elite-based update mechanisms have been widely utilized in the literature. Broadly speaking, two kinds of elite-based update mechanisms exist in optimizing single-objective problems. One is the explicit elite-guided update, like in JADE [59] and the other is the implicit elite-guided update, like in estimation of distribution algorithms (EDAs) [69]. In the former, the top p best individuals (namely elites) in the population are utilized to generate offspring, while in the latter, the elites are generally utilized to estimate the distribution of the population and then offspring are generated based on the estimated distribution.

The proposed EGL belongs to the former kind of elite-based update. However, the differences between JADE [59] and the proposed EGL are twofold. First, in EGL, the elites are not updated, but directly enter the next generation to protect useful information from being lost or weakened. Nevertheless, in JADE, the elites are also updated. Second, in DEGLSO, the elite set $\boldsymbol{ES}$ is further utilized to devise an adaptive communication strategy for efficient information exchange between slaves, which will be introduced in the next section.

B. Adaptive Communication

In distributed models, one of the most important components is the communication strategy, which has a significant effect on the performance of distributed metaheuristics with respect to both the solution quality and the execution time [6]. In particular, for the master–slave distributed model, a good communication strategy should appropriately provide solutions to two issues: 1) when the slaves make communication with the master and 2) what the slaves exchange with the master.

To make the slaves communicate with the master efficiently, we devise an adaptive communication strategy, which could afford proper solutions to the above two issues. As shown in Fig. 1, there are two kinds of information exchange between the master and the slaves in DEGLSO: 1) each slave sends *gbest* found by the associated swarm to the master, so that different *gbest*s found by different slaves can be preserved in the archive ($\boldsymbol{A}$) in the master and then can be introduced to the slaves in the next kind of information exchange and 2) each slave introduces an individual from the master to aid the related swarm to update.

In terms of the former exchange, to save communication time, first, we let each slave communicate with the master independently. Then, the delivery of *gbest* to the master for each slave is triggered when *gbest* found by the related swarm is updated. In this way, this kind of communication is asynchronous and triggered adaptively during the optimization. Therefore, much waiting time could be saved.

With respect to the latter, to save communication time, we also design an asynchronous communication strategy based on the request–response mechanism. First, for each slave, when overlap exists in the elites of the associated swarm between two consecutive generations, namely, $\boldsymbol{ES}_{g-1} \cap \boldsymbol{ES}_g \neq \emptyset$ (where $\boldsymbol{ES}_{g-1}$ and $\boldsymbol{ES}_g$ are the elite sets of the last generation and the current generation, respectively), this slave sends a request to the master to introduce one individual to aid the swarm to update. Then, after receiving the request from the slave, the master randomly selects an individual from archive $\boldsymbol{A}$ and sends this individual along with its fitness to the corresponding slave. In this manner, each slave introduces one individual from the master independently as well. It should be mentioned that such communication is adaptively triggered by $\boldsymbol{ES}_{g-1} \cap \boldsymbol{ES}_g \neq \emptyset$ based on the following consideration. When $\boldsymbol{ES}_{g-1} \cap \boldsymbol{ES}_g = \emptyset$, all elites found in the last generation are out-of-date and replaced by the new ones found in this generation. Therefore, the ability of the swarm in finding better solutions using current information is strong and thus no introduction of individuals from the master is needed. However, when $\boldsymbol{ES}_{g-1} \cap \boldsymbol{ES}_g \neq \emptyset$, the ability of the swarm to find more promising areas is limited. Thus, before it is too late to help the swarm enhance its ability, the slave immediately sends a request to the master to introduce individuals once $\boldsymbol{ES}_{g-1} \cap \boldsymbol{ES}_g \neq \emptyset$. We can take $\boldsymbol{ES}_{g-1} \cap \boldsymbol{ES}_g$ as a measure to percept the search ability of the swarm.

Remark 4: By means of the above two schemes, the slaves could communicate with the master efficiently. Particularly, the developed communication strategy has the following features.

1) The communication is asynchronously and independently conducted between the slaves and the master, leading to great reduction in the waiting time. In particular, all communication is adaptively triggered according to the requirement of the search process.

2) Only one individual is sent and received in each time of communication, which could save a lot of time in communication and is also beneficial for DEGLSO to adapt to a large number of cores.
3) Each slave introduces one candidate from the master adaptively according to its own search state measured by $\boldsymbol{ES}_{g-1} \cap \boldsymbol{ES}_g \neq \emptyset$. This adaptiveness could avoid too frequent or too little information exchange, both of which are not beneficial for the update of particles.
4) Since each slave communicates with the master independently and asynchronously, it is possible to make DEGLSO adapt to clusters with heterogeneous computers, which is considerably common in the real-world applications.

C. Implementation

We utilize message passing interface (MPI) as the tool to implement the devised DEGLSO. Specifically, the procedures of the master and the slaves are presented in Algorithms S1 and S2 in the supplementary material, respectively.

From Algorithm S1 in the supplementary material, we can see that the master does nothing but receives and sends information from and to the slaves. During the communication between the master and the slaves, three cases occur. In case 1 (lines 8–12), the master receives *gbest* found by the associated swarm when the slave terminates, which is corresponding to line 23 in Algorithm S2 in the supplementary material. Then, it compares these *gbest*s and obtains the final *gbest* of DEGLSO. In case 2 (lines 13–19), the master receives *gbest* from a slave during the iteration, which is associated with line 6 in Algorithm S2 in the supplementary material. After receiving this data, the master updates the archive ($\boldsymbol{A}$) either using a random replacement strategy along with an elite mechanism once $\boldsymbol{A}$ is full or directly putting it into $\boldsymbol{A}$ when $\boldsymbol{A}$ is not full. In case 3 (lines 20–22), the master receives the request from one slave requiring to introduce candidates, which corresponds to line 9 in Algorithm S2 in the supplementary material. Then, the master randomly selects an individual from $\boldsymbol{A}$ and sends this individual along with its fitness to the corresponding slave.

From Algorithm S2 in the supplementary material, we can see that each slave iterates a swarm using the proposed EGL to locate promising solutions. Besides, an archive $\boldsymbol{P}$ of size M is also maintained by each slave to store the individuals introduced from the master. To exchange information, each slave sends *gbest* found by the swarm to the master (line 6, triggered when *gbest* is updated) and then requests to introduce candidates from the master (lines 9 and 10, triggered when $\boldsymbol{ES}_{g-1} \cap \boldsymbol{ES}_g \neq \emptyset$). After receiving the data from the master, the slave will either put the received individual into $\boldsymbol{P}$ using the principle of replacing the worst when $\boldsymbol{P}$ is full (lines 11–13) or directly place the received individual into $\boldsymbol{P}$ when $\boldsymbol{P}$ is not full (line 15). Subsequently, particles in $\boldsymbol{NES}$ are updated using EGL (lines 18–20).

Compared with existing master–slave-based distributed metaheuristics, three differences can be noticed in DEGLSO.

1) With the adaptive communication strategy, the distributed model in DEGLSO is asynchronous. However, in most existing master–slave-based parallel metaheuristics [6], [23], the distributed models are synchronous, where the master stops and waits to receive information from all slaves before proceeding to the next generation, and each slave must wait until the master finishes sending information to those slaves ahead of it. However, in DEGLSO, the master waits only when there is no slave sending information to it. Once there is such a slave, it will receive the information and conduct operations corresponding to one of the three cases (cases 1–3) in Algorithm S1 in the supplementary material. Besides, one slave waits only when it sends a request to the master to introduce candidate individuals. Therefore, DEGLSO takes little waiting time.
2) In DEGLSO, the master takes charge of information exchange, while the slaves asynchronously iterate the swarms to find the optimum. However, in most existing master–slave-based distributed metaheuristics [6], [23], the master iterates the swarm and the slaves are to evaluate the fitness of the allocated individuals.
3) DEGLSO could adapt to clusters composed of heterogeneous computers. This advantage benefits from the devised adaptive and asynchronous communication strategy, which could reduce waiting time largely.

D. SEGLSO

To compare with DEGLSO, we also develop the serial version of DEGLSO, named SEGLSO, whose pseudocode is presented in Algorithm S3 in the supplementary material. To make fair comparisons, the number of subswarms in SEGLSO is the same as that of the slaves in DEGLSO. Besides, the main components of SEGLSO are the same as DEGLSO.

Comparing Algorithms S1 and S2 with Algorithm S3 in the supplementary material, we can find two main differences between DEGLSO and SEGLSO.

1) The iteration of the swarms is executed in parallel in DEGLSO, while that in SEGLSO is carried out sequentially. Theoretically, the execution time of DEGLSO should be much less than that of SEGLSO, which is empirically demonstrated in Section IV.
2) The communication among the swarms is asynchronous in DEGLSO, while it is synchronous in SEGLSO. Specifically, in SEGLSO, since the swarms are iterated sequentially, the new *gbest*s produced in the $(k-1)$ swarms may be immediately used to update the kth and the rest swarms if they are chosen as the candidates to be introduced into these swarms and then are utilized to update particles in these swarms. Nevertheless, in DEGLSO, due to the asynchronization, when a slave introduces candidates from the master, the new *gbest* may be not produced in other swarms or the

new *gbest*s generated by other swarms have not been delivered to the master to update $\boldsymbol{A}$.

In Section IV, experiments are conducted to compare DEGLSO with SEGLSO, and the experimental results substantiate that DEGLSO could achieve as good performance in terms of solution quality as SEGLSO, but takes much less execution time than SEGLSO.

IV. Experiments

To verify the effectiveness and efficiency of DEGLSO, we conduct extensive experiments on the latest and challenging CEC'2013 large-scale benchmark set [22]. The main properties of this set are listed in Table SI in the supplementary material.

In the following, we first investigate the parameter settings of DEGLSO and SEGLSO in Section IV-A. Then, extensive comparisons with respect to solution quality between DEGLSO and several state-of-the-art large-scale metaheuristics are conducted in Section IV-B. Particularly, in this section, we also investigate the comparison in regard to execution time and speedup between DEGLSO and one state-of-the-art large-scale optimizer implemented with the traditional master–slave model. In Section IV-C, we investigate the scalability of DEGLSO from the perspective of more cores and higher dimensionality. At last, in Section IV-D, we provide an in-depth investigation on DEGLSO via analyzing the influence of its components. However, due to the page limit, we attach the details of the experiments to Section III in the supplementary material.

In the experiments, unless otherwise stated, the number of cores is experimentally set to 21 for DEGLSO (indicating that except for one master, 20 slaves exist) and the maximum number of fitness evaluations is set to $5000 \times D$ (where D is the dimension size) for all compared algorithms. For fairness, median, mean, and standard deviation (Std) values over 30 independent runs are used to evaluate different algorithms. When two algorithms are compared, two-tailed Wilcoxon rank sum test is performed at the significance level of 0.05.

In addition, all algorithms are run on a homogeneous cluster with PCs composed of 4 Intel Xeon E3-1225 3.30-GHz CPUs, 8-Gb memory and 64-bit Ubuntu 16.04 LTS system.

A. Parameter Investigation

In DEGLSO, only the number of elites M needs fine-tuning. Since it is related to swarm size NP, which is a common parameter for all population-based metaheuristics, we set $M = \lfloor ER \times NP \rfloor$ for the convenience of fine-tuning, where $ER \in [0, 1]$ is the ratio of the elites out of the swarm, and $\lfloor x \rfloor$ is the floor function, which returns the largest integer smaller than or equal to x. For both DEGLSO and SEGLSO, we conduct experiments on six 1000-D functions (F_1, F_7, F_8, $F_{13} \sim F_{15}$) with ER varying from 0.1 to 0.5 and NP ranging from 20 to 50. It should be mentioned that these six functions are selected because we want to investigate the parameter settings on almost all kinds of functions, like fully separable, partially separable, overlapping, nonseparable, unimodal, and multimodal functions.

Table SII, in the supplementary material, shows the experimental results with the left of the bolded line representing the results of DEGLSO and the right denoting the results of SEGLSO. The best results of both DEGLSO and SEGLSO are bolded in the left and right of the bolded line, respectively.

From this table, we can obtain the following findings.

1) On most functions, when NP varies from 20 to 50, the most proper ER is 0.2 for both algorithms.
2) When ER is fixed as 0.2, it seems that NP makes little difference on the performance of both algorithms on most functions, except for F_1.
3) Particularly, we find that $NP = 30$ with $ER = 0.2$ is the most proper setting for both algorithms.

In conclusion, $NP = 30$ with $ER = 0.2$ is adopted for both DEGLSO and SEGLSO in the following experiments conducted on 1000-D problems.

B. Comparison With State-of-the-Art Large-Scale EAs

To comprehensively verify the effectiveness and efficiency of DEGLSO along with SEGLSO, we compare them with several state-of-the-art large-scale methods. Particularly, five large-scale optimizers focusing on the second aspect and five CCEAs concentrating on the first aspect on large-scale optimization as described in Section II are, respectively, selected. The former five are CSO [10], DMS-L-PSO [52], EOBDE [60], GODE [13], and SOUPDE [12], while the latter five are CCPSO2 [17], DECC-G [42], MLCC [46], DECC-DG [16], and DECC-XDG [49]. For fairness, the parameters of the compared algorithms are set as recommended in the related papers.

1) Solution Quality Comparison: Table I displays the comparison results among different algorithms on the 15 1000-D CEC'2013 benchmark functions. In this table, two rows of p-values exist with the first row representing the results when DEGLSO is compared with the associated algorithms and the other denoting the results when SEGLSO is compared with the corresponding algorithms. Besides, the highlighted p-values mean that DEGLSO or SEGLSO is significantly better than the corresponding compared algorithms. In addition, the symbols, "+," "−," and "=," above the p-values represent that DEGLSO or SEGLSO is significantly better than, significantly worse than, and equivalent to the compared algorithms on the associated functions. Accordingly, there are two "*w/t/l*" in the last row with the first representing that DEGLSO wins on w functions, ties on t functions, and loses on l functions in total in the competitions with the counterpart methods and the second denoting those numbers of SEGLSO.

The experimental results in this table can be summarized as follows.

1) In terms of solution quality, DEGLSO and SEGLSO achieve very similar performance on most (13 out of 15) functions.
2) Both DEGLSO and SEGLSO are better than the ten compared algorithms on at least 8 functions.
3) Concretely, both DEGLSO and SEGLSO are better than CSO on at least 8 functions and significantly superior to DMS-L-PSO on at least 11 functions. Particularly,

DEGLSO and SEGLSO beat GODE and EOBDE down both on 14 functions, and are significantly better than SOUPDE on 10 functions. In comparison with CCPSO2, DECC-G, MLCC, DECC-DG, and DECC-XDG, DEGLSO significantly dominates them on at least 8 functions, while SEGLSO performs quite better on at least 9 functions.

In short, DEGLSO achieves very similar performance with SEGLSO and both of them achieve very competitive or even better performance in comparison with the ten state-of-the-art large-scale algorithms. The superior performance of DEGLSO and SEGLSO in regard to the solution quality benefits from the proposed EGL strategy and the devised adaptive communication mechanism, which bring many benefits to them in diversity maintenance as elucidated in Section III.

2) Execution Time and Speedup Comparison: To investigate the superiority of DEGLSO in execution time, we record the execution time of both DEGLSO and SEGLSO on the CEC'2013 set. In addition, to make comparison, we also develop the distributed version of CSO, namely DCSO (the serial version of CSO is denoted as SCSO), using the traditional master–slave distributed model, where the master is responsible for the update of particles, while the slaves are to compute the fitness of particles. In particular, in DCSO, the updated particles are equally distributed to the slaves to compute their fitness values. In this experiment, CSO is selected because: 1) CSO is one popular and state-of-the-art PSO variant in handling large-scale problems, and also focuses on the same aspect (the second aspect as stated in Section II) as DEGLSO and SEGLSO and 2) compared with DMS-L-PSO, EOBDE, GODE, and SOUPDE, the superiority of DEGLSO and SEGLSO to CSO is the smallest in solution quality as shown in Table I.

The efficiency of a distributed metaheuristic can be reflected by the speedup of the execution time defined

$$\text{Speedup} = \frac{T_{\text{serial}}}{T_{\text{parallel}}} \tag{9}$$

where T_{parallel} and T_{serial} are the averaged execution time of the distributed metaheuristic and that of its serial version over multiple independent runs, respectively.

For fairness, the source code of SCSO is directly downloaded from the associated authors' websites and DCSO is implemented based on SCSO. The maximum number of fitness evaluations is set to $5000 \times D$ for all algorithms. Besides, DEGLSO and DCSO utilize 21 cores to optimize each problem in the CEC'2013 set. Table II presents the execution time (in second) of the four algorithms and the speedup of the two distributed metaheuristics. The function evaluation time of each function is also provided in this table.

From this table, we can observe the following.

1) DEGLSO not only takes much less time than SEGLSO but also takes significantly less time than DCSO and SCSO.
2) The speedup of DEGLSO is significantly higher than that of DCSO.
3) Concretely, on almost all functions, except for F_{12}, the speedup of DEGLSO is at least 14, while that of DCSO is smaller than 4.
4) Particularly, on F_{12}, due to the short function evaluation time, the efficiency of both DEGLSO and DCSO is not as good as that on functions with longer function evaluation time. Due to the large communication between the slaves and the master in DCSO, which leads to more time in communication than the function evaluation, DCSO even takes more time than SCSO. However, DEGLSO still takes much less time than SEGLSO. The only exception is that the speedup is not as high as that on other functions with longer function evaluation time, but is still much higher than DCSO on all functions.
5) With the function evaluation time increasing, in most cases, the speedup of both DEGLSO and DCSO increases as well. This matches the expectation that distributed metaheuristics are more suitable to tackle problems with high computational cost.
6) DEGLSO can not only handle problems with high computational cost better but also retain greater efficiency on problems with low computational cost as compared to DCSO.

The superiority of DEGLSO in the execution time and speedup benefits from the devised adaptive and asynchronous communication strategy. On the one hand, with this adaptive communication strategy, the slaves in DEGLSO communicate with the master independently. Thus, little waiting time exists in the communication between the slaves and the master. However, in the traditional master–slave model, the master needs to wait before at least one slave finishes the fitness evaluation and sends the fitness values to the master. Besides, the slaves also need to wait when the master updates particles and one slave must wait until the master has finished sending particles to those slaves ahead of it. Therefore, much waiting time exists in the traditional master–slave model, leading to the deficiency of DCSO. On the other hand, the slaves in DEGLSO communicate with the master adaptively. This indicates that the slaves in DEGLSO do not communicate with the master every generation like in the traditional master–slave model. Instead, for each slave, the communication is adaptively trigged based on the search state, like *gbest* is updated or $\boldsymbol{ES}_{g-1} \cap \boldsymbol{ES}_g \neq \emptyset$.

C. Scalability Investigation

In this section, we conduct experiments to investigate the scalability of DEGLSO from three perspectives: 1) scalability to more cores with fixed total fitness evaluations; 2) scalability to more cores with fixed iterations for each slave; and 3) scalability to higher dimensionality.

1) Scalability to More Cores With Fixed Total Fitness Evaluations: First, we investigate the scalability of DEGLSO to more cores when given fixed 5×10^6 total fitness evaluations. In this case, the total fitness evaluations are equally allocated to all slaves. We execute DEGLSO on the 1000-D CEC'2013 set with the number of cores varying from 6 to 46.

TABLE I
FITNESS COMPARISON RESULTS AMONG DIFFERENT ALGORITHMS ON 1000-D CEC'2013 PROBLEMS

F	Quality	DEGLSO	SEGLSO	CSO	DMS-L-PSO	EOBDE	GODE	SOUPDE	CCPSO2	DECC-G	MLCC	DECC-DG	DECC-XDG
F_1	Median	1.06E-13	2.14E-14	2.01E-17	1.89E+09	3.68E+11	3.36E+11	1.85E-29	4.09E+00	1.46E-25	0.00E+00	5.34E-04	1.52E-04
	Mean	2.94E-13	1.39E-13	2.00E-17	1.89E+09	3.62E+11	3.23E+11	4.51E-28	4.53E+00	9.59E-25	0.00E+00	3.96E-03	8.94E-02
	Std	4.45E-13	4.22E-13	9.82E-19	1.52E+08	2.99E+10	4.64E+10	1.37E-27	1.27E+00	1.65E-24	0.00E+00	1.16E-02	4.44E-01
	p-value	-	2.27E-03$^-$	3.02E-11$^-$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	2.52E-11$^-$	**3.02E-11**$^+$	3.02E-11$^-$	1.21E-12$^-$	**3.02E-11**$^+$	**3.02E-11**$^+$
		-	-	3.02E-11$^-$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	2.52E-11$^-$	**3.02E-11**$^+$	3.02E-11$^-$	1.21E-12$^-$	**3.02E-11**$^+$	**3.02E-11**$^+$
F_2	Median	6.29E+03	6.21E+03	7.43E+03	1.02E+04	1.04E+05	7.02E+04	1.96E+00	1.11E+01	2.69E+01	9.95E-01	1.28E+04	1.28E+04
	Mean	6.32E+03	6.27E+03	7.42E+03	1.02E+04	1.01E+05	6.92E+04	2.53E+00	1.14E+01	3.22E+01	2.69E+00	1.28E+04	1.29E+04
	Std	2.31E+02	2.89E+02	3.94E+02	2.67E+02	2.32E+04	2.10E+04	2.48E+00	1.52E+00	1.67E+01	3.88E+00	6.17E+02	6.47E+02
	p-value	-	4.29E-01$^=$	**4.98E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	2.97E-11$^-$	3.02E-11$^-$	3.00E-11$^-$	2.96E-11$^-$	**3.02E-11**$^+$	**3.02E-11**$^+$
		-	-	**3.34E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	2.97E-11$^-$	3.02E-11$^-$	3.00E-11$^-$	2.96E-11$^-$	**3.02E-11**$^+$	**3.02E-11**$^+$
F_3	Median	2.00E+01	2.00E+01	2.16E+01	2.03E+01	2.16E+01	2.16E+01	2.01E+01	2.00E+01	2.01E+01	2.00E+01	2.13E+01	2.13E+01
	Mean	2.02E+01	2.01E+01	2.16E+01	2.04E+01	2.16E+01	2.16E+01	2.01E+01	2.00E+01	2.01E+01	2.00E+01	2.13E+01	2.13E+01
	Std	5.02E-01	2.98E-01	4.68E-03	2.70E-01	1.03E-02	1.07E-02	2.25E-03	6.07E-05	1.67E-03	2.35E-05	1.56E-02	1.40E-02
	p-value	-	1.54E-01$^=$	**1.95E-10**$^+$	**2.32E-06**$^+$	**4.06E-11**$^+$	**9.74E-10**$^+$	7.95E-03$^-$	1.38E-11$^-$	1.91E-01$^=$	2.83E-11$^-$	**1.07E-07**$^+$	**1.07E-07**$^+$
		-	-	**7.37E-11**$^+$	**7.12E-09**$^+$	**3.68E-11**$^+$	**1.33E-10**$^+$	1.17E-05$^-$	2.81E-10$^-$	1.69E-02$^-$	3.26E-10$^-$	**5.57E-10**$^+$	**5.57E-10**$^+$
F_4	Median	3.11E+09	3.19E+09	1.33E+10	2.08E+11	3.59E+13	2.85E+13	3.56E+11	2.00E+10	5.34E+10	5.55E+10	4.17E+10	4.28E+09
	Mean	3.11E+09	3.05E+09	1.34E+10	2.23E+11	3.71E+13	2.96E+13	3.55E+11	2.34E+10	5.29E+10	6.20E+10	4.68E+10	4.50E+09
	Std	6.90E+08	6.71E+08	2.30E+09	9.64E+10	1.18E+13	9.18E+12	5.32E+10	1.52E+10	2.09E+10	3.83E+10	2.61E+10	1.33E+09
	p-value	-	9.59E-01$^=$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**1.34E-05**$^+$
		-	-	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**8.29E-06**$^+$
F_5	Median	2.10E+06	2.02E+06	6.08E+05	3.78E+06	7.14E+07	6.46E+07	9.84E+06	1.59E+07	7.67E+06	1.30E+07	4.91E+06	4.47E+06
	Mean	2.11E+06	2.04E+06	5.92E+05	3.98E+06	7.04E+07	6.44E+07	9.88E+06	1.64E+07	8.19E+06	1.42E+07	4.94E+06	4.51E+06
	Std	3.32E+05	3.55E+05	7.86E+04	6.62E+05	6.82E+06	7.63E+06	7.89E+05	4.23E+06	1.68E+06	4.31E+06	4.40E+05	4.41E+05
	p-value	-	2.64E-01$^=$	3.02E-11$^-$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$
		-	-	3.02E-11$^-$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$
F_6	Median	1.06E+06	1.06E+06	1.06E+06	1.03E+06	1.06E+06	1.06E+06	1.05E+06	1.04E+06	1.06E+06	1.04E+06	1.06E+06	1.06E+06
	Mean	1.06E+06	1.06E+06	1.06E+06	1.03E+06	1.06E+06	1.06E+06	1.05E+06	1.04E+06	1.06E+06	1.04E+06	1.06E+06	1.06E+06
	Std	3.65E+03	4.05E+03	1.08E+03	3.47E+03	1.42E+03	2.30E+03	1.65E+03	9.37E+03	2.39E+03	6.10E+03	1.36E+03	1.14E+03
	p-value	-	1.65E-01$^=$	3.71E-02$^-$	3.02E-11$^-$	3.02E-11$^-$	9.18E-06$^-$	5.56E-10$^-$	1.96E-10$^-$	9.47E-03$^-$	1.78E-10$^-$	2.64E-01$^=$	9.18E-01$^=$
		-	-	6.52E-01$^=$	3.02E-11$^-$	3.34E-11$^-$	1.78E-04$^-$	5.56E-10$^-$	2.61E-10$^-$	9.03E-04$^-$	2.15E-10$^-$	6.63E-01$^=$	1.26E-01$^=$
F_7	Median	1.11E+06	8.85E+05	1.01E+06	2.45E+09	6.94E+15	3.37E+15	2.49E+09	3.17E+07	5.51E+07	1.28E+08	2.06E+08	2.55E+06
	Mean	1.15E+06	9.91E+05	1.22E+06	2.45E+09	8.58E+15	4.37E+15	2.45E+09	8.42E+07	1.04E+08	2.49E+08	2.22E+08	3.33E+06
	Std	3.93E+05	3.51E+05	4.65E+05	1.54E+08	5.36E+15	3.48E+15	5.37E+08	1.30E+08	2.05E+08	3.35E+08	1.06E+08	2.84E+06
	p-value	-	6.79E-02$^=$	9.00E-01$^=$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**1.31E-08**$^+$
		-	-	**3.39E-02**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**1.29E-09**$^+$
F_8	Median	6.45E+13	5.44E+13	1.40E+14	8.65E+13	1.71E+18	1.43E+18	1.29E+16	5.84E+14	1.33E+15	1.90E+15	2.57E+15	1.61E+14
	Mean	6.47E+13	6.15E+13	1.43E+14	2.50E+14	1.70E+18	1.47E+18	1.28E+16	6.44E+14	1.53E+15	2.45E+15	2.92E+15	1.70E+14
	Std	1.70E+13	3.09E+13	2.72E+13	4.82E+14	4.17E+17	5.47E+17	2.92E+15	3.48E+14	7.86E+14	1.59E+15	1.88E+15	9.19E+13
	p-value	-	4.55E-01$^=$	**3.02E-11**$^+$	**3.67E-03**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**2.20E-07**$^+$
		-	-	**1.96E-10**$^+$	**2.05E-03**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**1.87E-07**$^+$
F_9	Median	6.11E+08	4.83E+08	5.88E+07	3.67E+08	3.49E+16	1.53E+15	1.36E+09	3.35E+09	5.72E+08	1.13E+09	4.20E+08	4.67E+08
	Mean	6.04E+08	5.51E+08	5.88E+07	3.67E+08	4.53E+16	1.11E+16	1.39E+09	3.71E+09	5.93E+08	1.16E+09	4.25E+08	4.63E+08
	Std	1.35E+08	2.05E+08	1.52E+07	3.90E+07	4.18E+16	2.09E+16	1.31E+08	1.00E+09	1.25E+08	3.83E+08	3.43E+07	3.23E+07
	p-value	-	1.33E-02$^-$	3.02E-11$^-$	8.15E-11$^-$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	8.42E-01$^=$	**4.57E-09**$^+$	1.56E-08$^-$	1.64E-05$^-$
		-	-	3.02E-11$^-$	1.41E-09$^-$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.69E-11**$^+$	**3.02E-11**$^+$	**1.33E-02**$^+$	**8.48E-09**$^+$	1.89E-04$^-$	2.12E-01$^=$
F_{10}	Median	9.11E+07	9.11E+07	9.40E+07	9.29E+07	9.50E+07	9.44E+07	9.29E+07	9.32E+07	9.28E+07	9.22E+07	9.44E+07	9.44E+07
	Mean	9.13E+07	9.14E+07	9.39E+07	9.29E+07	9.50E+07	9.44E+07	9.29E+07	9.32E+07	9.28E+07	9.24E+07	9.43E+07	9.44E+07
	Std	7.93E+05	8.63E+05	2.32E+05	3.17E+05	3.16E+05	3.01E+05	3.22E+05	5.20E+05	5.53E+05	6.19E+05	2.60E+05	2.66E+05
	p-value	-	6.31E-01$^=$	**3.47E-10**$^+$	**1.10E-08**$^+$	**3.02E-11**$^+$	**4.50E-11**$^+$	**1.01E-08**$^+$	**3.82E-09**$^+$	**1.31E-08**$^+$	**2.78E-07**$^+$	**4.98E-11**$^+$	**4.08E-11**$^+$
		-	-	**9.76E-10**$^+$	**2.02E-08**$^+$	**3.02E-11**$^+$	**5.49E-11**$^+$	**1.31E-08**$^+$	**1.10E-08**$^+$	**6.01E-08**$^+$	**1.39E-06**$^+$	**8.15E-11**$^+$	**4.98E-11**$^+$
F_{11}	Median	9.22E+11	9.22E+11	9.23E+11	2.69E+11	1.35E+16	1.13E+16	9.32E+11	9.32E+11	1.82E+10	1.88E+10	8.66E+09	2.67E+08
	Mean	9.26E+11	9.28E+11	9.27E+11	2.41E+11	1.35E+16	1.08E+16	9.34E+11	9.31E+11	2.16E+10	3.42E+10	2.23E+10	2.70E+08
	Std	8.34E+09	1.03E+10	1.01E+10	7.88E+10	4.62E+15	4.36E+15	7.49E+09	1.08E+10	1.92E+10	3.71E+10	3.28E+10	6.88E+07
	p-value	-	5.89E-01$^=$	6.20E-01$^=$	3.02E-11$^-$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.99E-04**$^+$	**4.68E-02**$^+$	3.02E-11$^-$	3.02E-11$^-$	3.02E-11$^-$	3.02E-11$^-$
		-	-	9.23E-01$^=$	3.02E-11$^-$	**3.02E-11**$^+$	**3.02E-11**$^+$	**2.32E-02**$^+$	2.01E-01$^=$	3.02E-11$^-$	3.02E-11$^-$	3.02E-11$^-$	3.02E-11$^-$
F_{12}	Median	1.59E+08	3.68E+03	1.04E+03	1.33E+04	8.10E+12	1.71E+12	1.02E+03	1.99E+03	2.72E+03	1.87E+03	7.48E+10	2.93E+03
	Mean	2.15E+08	3.78E+03	1.07E+03	3.20E+04	6.86E+12	1.81E+12	1.02E+03	1.95E+03	2.77E+03	1.94E+03	7.40E+10	2.94E+03
	Std	2.04E+08	6.59E+02	4.53E+01	4.57E+04	2.45E+12	5.70E+11	8.35E+00	2.02E+02	2.73E+02	4.05E+02	9.10E+09	3.12E+02
	p-value	-	3.02E-11$^-$	3.02E-11$^-$	3.02E-11$^-$	**3.02E-11**$^+$	**3.02E-11**$^+$	3.02E-11$^-$	3.02E-11$^-$	3.02E-11$^-$	3.02E-11$^-$	**3.02E-11**$^+$	3.02E-11$^-$
		-	-	3.02E-11$^-$	**3.69E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	3.02E-11$^-$	3.02E-11$^-$	7.39E-11$^-$	3.16E-10$^-$	**3.02E-11**$^+$	3.16E-10$^-$
F_{13}	Median	9.18E+07	1.11E+08	3.06E+08	1.13E+10	5.64E+17	3.18E+17	3.40E+10	1.91E+09	2.29E+09	2.71E+09	1.35E+10	4.35E+08
	Mean	1.17E+08	1.31E+08	3.26E+08	1.21E+10	8.16E+17	4.06E+17	3.38E+10	2.03E+09	2.44E+09	3.18E+09	1.34E+10	4.77E+08
	Std	6.55E+07	7.95E+07	1.08E+08	7.02E+09	6.27E+17	3.14E+17	4.12E+09	8.79E+08	9.46E+08	1.51E+09	3.99E+09	1.59E+08
	p-value	-	6.95E-01$^=$	**4.20E-10**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**6.70E-11**$^+$
		-	-	**2.67E-09**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**1.21E-10**$^+$
F_{14}	Median	4.44E+07	4.63E+07	9.12E+08	2.41E+11	9.96E+17	7.07E+17	4.49E+11	2.80E+10	3.31E+10	4.46E+10	4.46E+09	1.58E+08
	Mean	4.82E+07	5.37E+07	1.04E+09	2.48E+11	1.29E+18	7.38E+17	4.34E+11	4.96E+10	3.50E+10	5.72E+10	6.50E+09	2.18E+08
	Std	1.37E+07	2.48E+07	6.71E+08	1.10E+11	9.13E+17	4.49E+17	7.35E+10	7.68E+10	1.89E+10	4.57E+10	6.88E+09	1.42E+08
	p-value	-	6.20E-01$^=$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**4.98E-11**$^+$	**5.49E-11**$^+$
		-	-	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**5.49E-11**$^+$	**3.82E-10**$^+$
F_{15}	Median	2.66E+06	2.67E+06	6.63E+07	1.06E+07	2.20E+17	2.42E+16	3.47E+06	2.19E+06	2.06E+06	3.46E+06	6.41E+06	6.69E+06
	Mean	2.74E+06	2.71E+06	6.65E+07	1.15E+07	3.07E+17	3.21E+16	3.46E+06	3.04E+06	2.78E+06	3.39E+06	6.62E+06	7.13E+06
	Std	4.55E+05	1.97E+05	4.03E+06	4.11E+06	3.14E+17	3.41E+16	2.84E+05	2.00E+06	3.60E+06	4.08E+05	1.16E+06	1.95E+06
	p-value	-	4.92E-01$^=$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**2.44E-09**$^+$	3.04E-01$^=$	**5.53E-08**$^+$	**2.19E-08**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$
		-	-	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$	**9.92E-11**$^+$	2.97E-01$^=$	**2.39E-08**$^+$	**7.12E-09**$^+$	**3.02E-11**$^+$	**3.02E-11**$^+$
w/t/l			0/13/2	8/2/5	11/0/4	14/0/1	14/0/1	10/0/5	10/1/4	8/2/5	9/0/6	12/1/2	11/1/3
			-	9/2/4	12/0/3	14/0/1	14/0/1	10/0/5	9/2/4	9/0/6	9/0/6	12/1/2	11/2/2

Accordingly, the number of subswarms in SEGLSO ranges from 5 to 45.

Fig. S1, in the supplementary material, displays the change of the averaged fitness values of both DEGLSO and SEGLSO on different functions with the number of cores (subswarms) increasing from 6 to 46 (5 to 45). Fig. S2, in the supplementary material, presents the change of the execution time of DEGLSO and SEGLSO and that of the speedup as the number of cores changes.

From Fig. S1 in the supplementary material, the following observations can be obtained with respect to solution quality.

1) On most functions, except for F_{12}, on which DEGLSO is inferior to SEGLSO, DEGLSO achieves similar performance with SEGLSO.
2) On most functions (8 out of 15), the performance of both DEGLSO and SEGLSO first becomes better and better and then becomes worse and worse with the number of cores increasing. This phenomenon can be explained as

TABLE II
TIME (IN SECOND) AND SPEEDUP COMPARISON AMONG DEGLSO, SEGLSO, DCSO, AND SCSO ON 1000-D CEC'2013 PROBLEMS

F	FT	Quality	DEGLSO	SEGLSO	DCSO	SCSO
F_1	2.91E-04	Time	69.54	1166.29	489.72	1177.50
		Speedup	16.77		2.40	
F_2	3.74E-04	Time	83.11	1417.98	505.58	1480.96
		Speedup	17.06		2.93	
F_3	3.78E-04	Time	85.03	1485.07	508.11	1512.88
		Speedup	17.47		2.98	
F_4	3.86E-04	Time	72.29	1239.83	497.27	1255.91
		Speedup	17.15		2.53	
F_5	4.70E-04	Time	87.16	1541.16	511.37	1571.07
		Speedup	17.68		3.07	
F_6	4.72E-04	Time	90.37	1588.33	513.36	1604.18
		Speedup	17.58		3.12	
F_7	1.81E-04	Time	30.30	448.09	459.27	468.72
		Speedup	14.79		1.02	
F_8	4.52E-04	Time	85.92	1480.26	489.32	1500.05
		Speedup	17.23		3.07	
F_9	5.30E-04	Time	101.16	1791.88	520.79	1784.77
		Speedup	17.71		3.43	
F_{10}	5.42E-04	Time	104.10	1886.41	508.20	1859.99
		Speedup	18.12		3.66	
F_{11}	4.14E-04	Time	78.65	1414.98	502.09	1394.68
		Speedup	17.99		2.78	
F_{12}	1.48E-05	Time	12.14	77.25	433.97	140.26
		Speedup	6.36		0.32	
F_{13}	4.09E-04	Time	79.56	1369.86	501.24	1414.66
		Speedup	17.22		2.82	
F_{14}	4.13E-04	Time	79.41	1360.66	480.55	1405.22
		Speedup	17.13		2.92	
F_{15}	2.62E-04	Time	61.84	1040.52	483.11	1040.79
		Speedup	16.83		2.15	

follows. On the one hand, when the number of cores is too small, only a few swarms are maintained, leading to that no enough diversity is afforded. In this situation, both DEGLSO and SEGLSO may easily get trapped into local optima; on the other hand, when the number of cores is too large, given fixed total fitness evaluations, the number of fitness evaluations allocated to each slave is small, leading to that the swarm in each slave cannot be iterated well enough to approach to the global optimum.

3) On F_6 and F_{10}, the performance of DEGLSO and SEGLSO tends to be worse and worse when the number of cores increases.
4) Surprisingly, on F_2, F_5, and F_9, the performance of DEGLSO and SEGLSO becomes better and better as the number of cores increases. This may benefit from the increasing diversity brought by the increasing swarms, which is precious for multimodal functions.

From Fig. S2 in the supplementary material, the experimental results in terms of execution time and speedup can be summarized as follows.

1) The execution time of DEGLSO is much smaller than that of SEGLSO.
2) With the number of subswarms increasing, the execution time of SEGLSO almost remains unchanged. This is not strange because the total number of fitness evaluations is fixed and during the execution, the fitness evaluation takes the most computational cost.
3) As the number of cores increases, the execution time of DEGLSO becomes smaller and smaller almost on all functions, except for F_{12}. This is not surprising because more cores partition the iteration of the optimizer and then more fitness evaluations are performed in parallel.
4) The only exception for DEGLSO is F_{12}, on which its execution time first decreases when the number of cores increases from 6 to 26 and then increases as the number of cores increases from 26 to 46. This is because the function evaluation time of F_{12} is too small. When the number of cores is small, the communication time in DEGLSO is negligible compared with the function evaluation time. However, when the number of cores is too large, the communication time is not negligible, but becomes comparable or even larger than the function evaluation time.
5) On almost all functions, except for F_{12}, DEGLSO achieves linear speedup as the number of cores increases.

Overall, taking both the solution quality and the execution time into consideration, we find that given fixed 5×10^6 total fitness evaluations, DEGLSO with 21 cores could make a good compromise over all functions.

2) Scalability to More Cores With Fixed Iterations for Each Slave: Subsequently, we conduct experiments on the 1000-D CEC'2013 set to investigate the scalability of DEGLSO to more cores when given a fixed number of iterations for each slave. Particularly, in this experiment, the maximum number of generations for each slave (each subswarm) is fixed to 10 500. This number is selected to keep consistency with the previous experiments, so that when the number of cores is 21, the maximum number of function evaluations is close to $5000 \times D$ ($D = 1000$). We execute DEGLSO with the number of cores varying from 6 to 46. Accordingly, for SEGLSO, the number of subswarms ranges from 5 to 45.

Fig. S3, in the supplementary material, displays the change of the averaged fitness values of both DEGLSO and SEGLSO on different functions with the number of cores (subswarms) increasing from 6 to 46 (5 to 45). Fig. S4, in the supplementary material, presents the change of the execution time of DEGLSO and SEGLSO and that of the speedup as the number of cores changes.

From Fig. S3 in the supplementary material, we can obtain the following observations.

1) On most functions, with the number of cores (subswarms) increasing, the performance of both DEGLSO and SEGLSO becomes better and better. This is not surprising because with the number of generations fixed in each slave (subswarms), as the number of cores (subswarms) grows, not only more particles participate in locating the optima but also more fitness evaluations are allocated.
2) DEGLSO can achieve very similar performance with SEGLSO on almost all functions, except for F_{12}, where DEGLSO performs worse than SEGLSO.
3) On F_3, F_{10}, and F_{11}, the performance of both DEGLSO and SEGLSO vibrates, but the tendency of the quality of the final solutions obtained by both algorithms is becoming better and better as the number of cores (subswarms) increases.

From Fig. S4 in the supplementary material, we can find the following phenomena.

1) As the number of cores (subswarms) increases, the execution time of SEGLSO increases fast and linearly on all functions, while that of DEGLSO grows very mildly on almost all functions, except for F_{12}, on which the

execution time of DEGLSO also increases linearly but much slower than that of SEGLSO. This is not strange because in DEGLSO, all swarms are iterated in parallel, while in SEGLSO, all subswarms are iterated in serial. Besides, due to the proposed adaptive and asynchronous communication strategy, little communication time is occupied, which could be ignored compared with the long function evaluation time. However, on F_{12}, due to its too short function evaluation time, the communication time may be comparable to the function evaluation time, leading to the increased execution time of DEGLSO as the number of cores increases.

2) Except for F_{12}, the speedup of DEGLSO increases linearly as the number of cores grows on almost all functions due to the efficient communication strategy.

Overall, we can conclude that DEGLSO could preserve a good scalability to more cores when given a fixed number of iterations for each slave. In particular, DEGLSO scales well to a large number of cores with linear speedup. Such superiority benefits from the devised communication strategy and the developed EGL. The former provides efficient and effective information exchange, leading to nearly unchanged execution time as the number of cores increases. The latter helps the swarms maintain high search diversity, giving rise to the good performance of DEGLSO in terms of the solution quality.

3) Scalability to Higher Dimensionality: At last, we investigate the scalability of DEGLSO to solve higher dimensional problems. First, we construct several 2000-D and 3000-D composite problems by concatenating different 1000-D functions in the CEC'2013 set. Without loss of generality, we utilize F_1, F_4, F_7, F_8, F_{11}, F_{13}, F_{14}, and F_{15} in the CEC'2013 set to construct six 2000-D problems and six 3000-D problems, respectively. These functions are selected because on the one hand, they could cover all the main properties of the CEC'2013 set, like fully separable, partially separable, overlapping, etc.; on the other hand, each variable of these functions has the same lower and upper bounds, which makes it easy to program. The six constructed 2000-D problems denoted as "$CF_{2,i}$" (the ith 2000-D composite function) and the six constructed 3000-D problems denoted as "$CF_{3,i}$" (the ith 3000-D composite function) are presented in Tables SIII and SV, in the supplementary material, respectively.

Second, we also adopt both DCSO and SCSO to make comparisons with DEGLSO and SEGLSO. In addition, *NP* and *ER* are set to 80 and 0.1 for DEGLSO and SEGLSO based on preliminary experiments. As for the parameters in CSO, they are set according to the guideline in [10]. In this experiment, 21 cores are adopted to execute both DEGLSO and DCSO and the maximum number of fitness evaluations is set to $5000 \times D$.

Tables SIII and SIV, in the supplementary material, present the experimental results on the six 2000-D problems with the former displaying the fitness comparison results and the latter showing the execution time and speedup comparison results. Tables SV and SVI, in the supplementary material, respectively, display the fitness comparison results and the execution time and speedup comparison results on the six 3000-D problems.

From Table SIII in the supplementary material, in terms of solution quality on the 2000-D problems, we find that: 1) DEGLSO still achieves very similar performance with SEGLSO on the functions and 2) DEGLSO and SEGLSO are better than DCSO and SCSO on the six functions.

From Table SIV in the supplementary material, with respect to the execution time and the speedup, we can obtain three observations.

1) DEGLSO takes significantly less time not only than SEGLSO, but also than both DCSO and SCSO.
2) The speedup of DEGLSO is considerably larger than that of DCSO. Specifically, on these 2000-D problems, the speedup of DEGLSO is at least 16, while that of DCSO is less than 3.
3) Compared with the speedup of DEGLSO on 1000-D problems in Table II, the speedup of DEGLSO on 2000-D problems is a little larger, due to the increase of the function evaluation time. However, that of DCSO does not increase so obviously, because although the function evaluation time increases, the communication content also increases due to the higher dimensionality.

From Table SV in the supplementary material, in terms of solution quality on the 3000-D problems, we can get similar observations: 1) DEGLSO achieves very similar performance with SEGLSO on these functions as well and 2) DEGLSO and SEGLSO are better than DCSO and SCSO on five functions. Unfortunately, on $CF_{3,2}$, DEGLSO and SEGLSO perform worse than DCSO and SCSO.

From Table SVI in the supplementary material, as for the execution time and speedup, we can find the following.

1) DEGLSO still takes much less time not only than SEGLSO, but also than DCSO and SCSO.
2) The speedup of DEGLSO is still much larger than that of DCSO. Specifically, on these 3000-D problems, the speedup of DEGLSO is at least 18, while that of DCSO is less than 3.
3) Compared with the speedup of DEGLSO and DCSO on the 2000-D problems, the speedup of DEGLSO and DCSO increases not so obviously. This is because the communication content greatly increases due to the higher dimensionality.

Overall, DEGLSO retains great efficiency and effectiveness on higher dimensional problems in terms of both the solution quality and the execution time. In particular, DEGLSO still maintains high speedup to optimize higher dimensional problems. The good scalability of DEGLSO to solve higher dimensional problems mainly benefits from the proposed EGL and the devised adaptive communication strategy. The former mainly makes DEGLSO obtain good performance in terms of the solution quality, while the latter mainly lets DEGLSO achieve good performance in terms of the execution time.

D. Investigation About DEGLSO

In this section, we analyze the influence of each component on DEGLSO, so that it is clear to know what contributes to the good performance of DEGLSO. However, due to the page limit, we have to attach the details of the analysis to Section III

in the supplementary material. As a consequence, we only list brief introductions and conclusions of the analysis here.

First, we take investigation about the swarm diversity and the fast convergence of DEGLSO via comparing it with GPSO and CSO (related to Section III-A in the supplementary material). The experimental results verify that DEGLSO compromises swarm diversity and fast convergence better than CSO and GPSO on both unimodal and multimodal problems.

Second, we investigate the influence of the archives in the master and the slaves on DEGLSO. We compare DEGLSO with its versions without archives in the master and the slaves (associated with Section III-B in the supplementary material). The experimental results substantiate that the archives in both the master and the slaves are vital and the archive in the master is more crucial than the archives in the slaves for DEGLSO.

Third, we investigate the influence of the proposed adaptive communication strategy from three perspectives (associated with Section III-C in the supplementary material).

1) We record the accumulated communication times between the slaves and the master triggered by $\boldsymbol{ES}_{g-1} \cap \boldsymbol{ES}_g \neq \emptyset$. The experimental results show that compared with letting the slaves communicate with the master every generation, the adaptive communication strategy could help DEGLSO save many communication times during the optimization.
2) We compare the asynchronous communication strategy with the one widely used synchronous communication strategy that each slave communicates with the master every T generations. The experimental results demonstrate that the proposed asynchronous communication strategy is much more effective than the synchronous one.
3) We compare the adaptive communication scheme with two asynchronous communication strategies triggered adaptively by two measures that the global best fitness remains unchanged in T consecutive generations and the relative improvement of the global best fitness between two successive generations is less than a threshold R, respectively. The experimental results verify that compared with the above two measures, the proposed measure $\boldsymbol{ES}_{g-1} \cap \boldsymbol{ES}_g \neq \emptyset$ could bring more sufficient and effective information exchange among slaves, contributing to the good performance of DEGLSO in terms of the solution quality.

Overall, the proposed EGL and the devised communication strategy cooperate with each other cohesively to aid the slaves to search the high-dimensional space effectively and efficiently, leading to the good performance of DEGLSO in terms of both the solution quality and the execution time and speedup.

V. Conclusion

This paper has proposed a DEGLSO to tackle large-scale optimization problems with high computational cost. Particularly, this distributed optimizer utilizes the special master–slave distributed model, where the master is mainly responsible for information exchange, while each slave iterates a swarm using the devised EGL to find the optimum of the problem. An adaptive and asynchronous communication strategy based on the request–response mechanism is especially designed for this distributed model, which adaptively triggers the communication between the master and the slaves. In this manner, each slave iterates the swarm asynchronously and communicates with the master independently. The proposed EGL and the devised communication strategy cooperate with each other cohesively to aid the slaves to search the high-dimensional space effectively and efficiently.

Extensive experiments conducted on the CEC'2013 benchmark set have demonstrated the competitive effectiveness and efficiency of DEGLSO with respect to the solution quality and the execution time, as compared to state-of-the-art large-scale algorithms. Particularly, DEGLSO achieves nearly linear speedup as the number of cores increases and preserves a good scalability to solve higher dimensional problems.

References

[1] X. Yu *et al.*, "ACO-A*: Ant colony optimization plus A* for 3D traveling in environments with dense obstacles," *IEEE Trans. Evol. Comput.*, to be published.

[2] W. Shi *et al.*, "An adaptive estimation of distribution algorithm for multipolicy insurance investment planning," *IEEE Trans. Evol. Comput.*, vol. 23, no. 1, pp. 1–14, Feb. 2019.

[3] W.-N. Chen *et al.*, "A cooperative co-evolutionary approach to large-scale multisource water distribution network optimization," *IEEE Trans. Evol. Comput.*, to be published.

[4] X. Wen *et al.*, "A maximal clique based multiobjective evolutionary algorithm for overlapping community detection," *IEEE Trans. Evol. Comput.*, vol. 21, no. 3, pp. 363–377, Jun. 2017.

[5] S. Cheng, B. Liu, Y. Shi, Y. Jin, and B. Li, "Evolutionary computation and big data: Key challenges and future directions," in *Proc. Int. Conf. Data Min. Big Data*, 2016, pp. 3–14.

[6] Y.-J. Gong *et al.*, "Distributed evolutionary algorithms and their models: A survey of the state-of-the-art," *Appl. Soft Comput.*, vol. 34, pp. 286–300, Sep. 2015.

[7] M. N. Omidvar, X. Li, and K. Tang, "Designing benchmark problems for large-scale continuous optimization," *Inf. Sci.*, vol. 316, pp. 419–436, Sep. 2015.

[8] 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.

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

[10] 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.

[11] S.-Z. Zhao, P. N. Suganthan, and S. Das, "Self-adaptive differential evolution with multi-trajectory search for large-scale optimization," *Soft Comput.*, vol. 15, no. 11, pp. 2175–2185, 2011.

[12] M. Weber, F. Neri, and V. Tirronen, "Shuffle or update parallel differential evolution for large-scale optimization," *Soft Comput.*, vol. 15, no. 11, pp. 2089–2107, 2011.

[13] H. Wang, Z. Wu, and S. Rahnamayan, "Enhanced opposition-based differential evolution for solving high-dimensional continuous optimization problems," *Soft Comput.*, vol. 15, no. 11, pp. 2127–2140, 2011.

[14] W. Dong, T. Chen, P. Tiòo, and X. Yao, "Scaling up estimation of distribution algorithms for continuous optimization," *IEEE Trans. Evol. Comput.*, vol. 17, no. 6, pp. 797–822, Dec. 2013.

[15] S. Mahdavi, M. E. Shiri, and S. Rahnamayan, "Metaheuristics in large-scale global continues optimization: A survey," *Inf. Sci.*, vol. 295, pp. 407–428, Feb. 2015.

[16] M. N. 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.

[17] 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.

[18] 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.

[19] Q. Yang *et al.*, "Adaptive multimodal continuous ant colony optimization," *IEEE Trans. Evol. Comput.*, vol. 21, no. 2, pp. 191–205, Apr. 2017.

[20] Q. Yang *et al.*, "Multimodal estimation of distribution algorithms," *IEEE Trans. Cybern.*, vol. 47, no. 3, pp. 636–650, Mar. 2017.

[21] Y.-H. Jia *et al.*, "Distributed cooperative co-evolution with adaptive computing resource allocation for large scale optimization," *IEEE Trans. Evol. Comput.*, to be published.

[22] X. Li *et al.*, "Benchmark functions for the CEC 2013 special session and competition on large-scale global optimization," Evol. Comput. Mach. Learn. Group, RMIT Univ., Melbourne, VIC, Australia, 2013.

[23] M. Dubreuil, C. Gagné, and M. Parizeau, "Analysis of a master–slave architecture for distributed evolutionary computations," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 36, no. 1, pp. 229–235, Feb. 2006.

[24] F. Herrera and M. Lozano, "Gradual distributed real-coded genetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 4, no. 1, pp. 43–63, Apr. 2000.

[25] H. Pierreval and J.-L. Paris, "Distributed evolutionary algorithms for simulation optimization," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 30, no. 1, pp. 15–24, Jan. 2000.

[26] M. Giacobini, M. Tomassini, A. G. B. Tettamanzi, and E. Alba, "Selection intensity in cellular evolutionary algorithms for regular lattices," *IEEE Trans. Evol. Comput.*, vol. 9, no. 5, pp. 489–505, Oct. 2005.

[27] E. Alba and B. Dorronsoro, "The exploration/exploitation tradeoff in dynamic cellular genetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 9, no. 2, pp. 126–142, Apr. 2005.

[28] G. Folino, C. Pizzuti, and G. Spezzano, "Training distributed GP ensemble with a selective algorithm based on clustering and pruning for pattern classification," *IEEE Trans. Evol. Comput.*, vol. 12, no. 4, pp. 458–468, Aug. 2008.

[29] L. Xu and F. Zhang, "Parallel particle swarm optimization for attribute reduction," in *Proc. Int. Conf. Softw. Eng. Artif. Intell. Netw. Parallel Distrib. Comput.*, 2007, pp. 770–775.

[30] S. M. Said and M. Nakamura, "Parallel enhanced hybrid evolutionary algorithm for continuous function optimization," in *Proc. Int. Conf. P2P Parallel Grid Cloud Internet Comput.*, 2012, pp. 125–131.

[31] R. Michel and M. Middendorf, "An island model based ant system with lookahead for the shortest supersequence problem," in *Proc. Parallel Probl. Solving Nat.*, 1998, pp. 692–701.

[32] T. Ishimizu and K. Tagawa, "A structured differential evolution for various network topologies," *Int. J. Comput. Commun.*, vol. 4, no. 1, pp. 2–8, 2010.

[33] C. Zhang, J. Chen, and B. Xin, "Distributed memetic differential evolution with the synergy of Lamarckian and Baldwinian learning," *Appl. Soft Comput.*, vol. 13, no. 5, pp. 2947–2959, 2013.

[34] X. Du, L. Ding, and L. Jia, "Asynchronous distributed parallel gene expression programming based on estimation of distribution algorithm," in *Proc. Int. Conf. Nat. Comput.*, 2008, pp. 433–437.

[35] M. Giacobini, M. Tomassini, and A. Tettamanzi, "Modeling selection intensity for linear cellular evolutionary algorithms," in *Proc. Artif. Evol.*, 2004, pp. 345–356.

[36] M. Giacobini *et al.*, "Modeling selection intensity for Toroidal cellular evolutionary algorithms," in *Proc. Genet. Evol. Comput. Conf.*, Seattle, WA, USA, 2004, pp. 1138–1149.

[37] B. Schönfisch and A. de Roos, "Synchronous and asynchronous updating in cellular automata," *Biosystems*, vol. 51, no. 3, pp. 123–143, 1999.

[38] D. Lim, Y.-S. Ong, Y. Jin, B. Sendhoff, and B.-S. Lee, "Efficient hierarchical parallel genetic algorithms using grid computing," *Future Gener. Comput. Syst.*, vol. 23, no. 4, pp. 658–670, 2007.

[39] F. Herrera, M. Lozano, and C. Moraga, "Hierarchical distributed genetic algorithms," *Int. J. Intell. Syst.*, vol. 14, no. 11, pp. 1099–1121, 1999.

[40] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Proc. Parallel Probl. Solving Nat.*, 1994, pp. 249–257.

[41] M. N. Omidvar, X. Li, Z. Yang, and X. Yao, "Cooperative co-evolution for large scale optimization through more frequent random grouping," in *Proc. IEEE Congr. Evol. Comput.*, 2010, pp. 1–8.

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

[43] J. Sun and H. Dong, "Cooperative co-evolution with correlation identification grouping for large scale function optimization," in *Proc. Int. Conf. Inf. Sci. Technol.*, 2013, pp. 889–893.

[44] M. N. Omidvar, X. Li, and X. Yao, "Cooperative co-evolution with delta grouping for large scale non-separable function optimization," in *Proc. IEEE Congr. Evol. Comput.*, 2010, pp. 1–8.

[45] Y. Mei, M. N. Omidvar, X. Li, and X. Yao, "A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization," *ACM Trans. Math. Softw.*, vol. 42, no. 2, pp. 1–24, 2016.

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

[47] T. Ray and X. Yao, "A cooperative coevolutionary algorithm with correlation based adaptive variable partitioning," in *Proc. IEEE Congr. Evol. Comput.*, 2009, pp. 983–989.

[48] Q. Yang, W.-N. Chen, and J. Zhang, "Evolution consistency based decomposition for cooperative coevolution," *IEEE Access*, vol. 6, pp. 51084–51097, 2018.

[49] Y. Sun, M. Kirley, and S. K. Halgamuge, "Extended differential grouping for large scale global optimization with direct and indirect variable interactions," in *Proc. Conf. Genet. Evol. Comput.*, 2015, pp. 313–320.

[50] Y. Sun, M. Kirley, and S. K. Halgamuge, "A recursive decomposition method for large scale continuous optimization," *IEEE Trans. Evol. Comput.*, vol. 22, no. 5, pp. 647–661, Oct. 2018.

[51] E. T. Oldewage, "The perils of particle swarm optimization in high dimensional problem spaces," Ph.D. dissertation, Dept. Comput. Sci., Univ. Pretoria, Pretoria, South Africa, 2018.

[52] 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.

[53] S.-T. Hsieh, T.-Y. Sun, C.-C. Liu, and S.-J. Tsai, "Solving large scale global optimization using improved particle swarm optimizer," in *Proc. IEEE Congr. Evol. Comput.*, 2008, pp. 1777–1784.

[54] J. García-Nieto and E. Alba, "Restart particle swarm optimization with velocity modulation: A scalability test," *Soft Comput.*, vol. 15, no. 11, pp. 2221–2232, 2011.

[55] R. Cheng, C. Sun, and Y. Jin, "A multi-swarm evolutionary framework based on a feedback mechanism," in *Proc. IEEE Congr. Evol. Comput.*, 2013, pp. 718–724.

[56] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proc. IEEE Congr. Evol. Comput.*, 1998, pp. 69–73.

[57] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 4, 1995, pp. 1942–1948.

[58] Q. Yang *et al.*, "A level-based learning swarm optimizer for large scale optimization," *IEEE Trans. Evol. Comput.*, vol. 22, no. 4, pp. 578–594, Aug. 2018.

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

[60] X. Zhou, Z. Wu, and H. Wang, "Elite opposition-based differential evolution for solving large-scale optimization problems and its implementation on GPU," in *Proc. Int. Conf. Parallel Distrib. Comput. Appl. Techn.*, 2012, pp. 727–732.

[61] M. Z. Ali, N. H. Awad, and P. N. Suganthan, "Multi-population differential evolution with balanced ensemble of mutation strategies for large-scale global optimization," *Appl. Soft Comput.*, vol. 33, pp. 304–327, Aug. 2015.

[62] S.-M. Guo, C.-C. Yang, P.-H. Hsu, and J. S.-H. Tsai, "Improving differential evolution with a successful-parent-selecting framework," *IEEE Trans. Evol. Comput.*, vol. 19, no. 5, pp. 717–730, Oct. 2015.

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

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

[65] H. Wang, L. Jiao, and X. Yao, "Two_Arch2: An improved two-archive algorithm for many-objective optimization," *IEEE Trans. Evol. Comput.*, vol. 19, no. 4, pp. 524–541, Aug. 2015.

[66] N. Chen *et al.*, "An evolutionary algorithm with double-level archives for multiobjective optimization," *IEEE Trans. Cybern.*, vol. 45, no. 9, pp. 1851–1863, Sep. 2015.

[67] C. Darwin. (1872). *The Origin of Species*. [Online]. Available: https://www.lulu.com/

[68] C. Darwin, "The origin of species by means of natural selection, or, the preservation of favored races in the struggle for life," in *International Science Library*. New York, NY, USA: D. Appleton & Company, 1897.
[69] P. Larrañaga and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. New York, NY, USA: Springer, 2002.

Qiang Yang (S'14–M'18) received the M.S. degree in computer science from Sun Yat-sen University, Guangzhou, China, in 2014, where he is currently pursuing the Ph.D. degree in computer science.

He is currently a Research Assistant with the School of Computer Science and Engineering, South China University of Technology, Guangzhou. His current research interests include evolutionary computation algorithms and their applications on real-world problems. So far, he specifically works on large scale optimization algorithms, multimodal optimization algorithms, distributed evolutionary algorithms and their applications on real-world problems, like intelligent transportation.

Wei-Neng Chen (S'07–M'12–SM'17) received the bachelor's and Ph.D. degrees in computer science from Sun Yat-sen University, Guangzhou, China, in 2006 and 2012, respectively.

Since 2016, he has been a Full Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou. He has co-authored over 100 international journal and conference papers, including over 30 papers published in the IEEE TRANSACTIONS journals. His current research interests include computational intelligence, swarm intelligence, and network science and their applications.

Dr. Chen was a recipient of the IEEE Computational Intelligence Society (CIS) Outstanding Dissertation Award in 2016, and the National Science Fund for Excellent Young Scholars in 2016. He is currently the Vice-Chair of the IEEE Guangzhou Section. He is also a Committee Member of the IEEE CIS Emerging Topics Task Force. He serves as an Associate Editor for the IEEE TRANSACTIONS ON NETWORKS AND LEARNING SYSTEMS and *Complex and Intelligent Systems*.

Tianlong Gu received the M.Eng. degree in computer science from Xidian University, Xi'an, China, in 1987, and the Ph.D. degree in computer science from Zhejiang University, Hangzhou, China, in 1996.

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

Huaxiang Zhang received the Ph.D. degree in computer software from Shanghai Jiaotong University, Shanghai, China, in 2004.

He was an Associated Professor with the Department of Computer Science, Shandong Normal University, Jinan, China, from 2004 to 2005, where he is currently a Professor with the School of Information Science and Engineering. He has authored over 100 journal and conference papers and has been granted eight invention patents. His current research interests include machine learning, pattern recognition, evolutionary computation, and Web information processing.

Huaqiang Yuan received the Ph.D. degree in computer software from Shanghai Jiao Tong University, Shanghai, China in 1996.

He is currently a Professor with the School of Computer Science and Network Security, Dongguan University of Technology, Dongguan, China. His current research interests include computational intelligence and cyberspace security.

Sam Kwong (M'93–SM'04–F'14) received the B.Sc. degree in electrical engineering from the State University of New York at Buffalo, Buffalo, NY, USA, in 1983, the M.A.Sc. degree in electrical engineering from the University of Waterloo, Waterloo, ON, Canada, in 1985, and the Ph.D. degree in electrical engineering from the University of Hagen, Hagen, Germany, in 1996.

From 1985 to 1987, he was a Diagnostic Engineer with Control Data Canada, Mississauga, ON, Canada, where he designed the diagnostic software to detect the manufacture faults of the VLSI chips in the Cyber 430 machine. He is currently the Head and a Professor of the Department of Computer Science, City University of Hong Kong, Hong Kong. His current research interests include pattern recognition, evolutionary computations, and video analytics.

Prof. Kwong was elevated to IEEE Fellow for his contributions on Optimization Techniques for Cybernetics and Video Coding in 2014. He is also appointed as the IEEE Distinguished Lecturer for IEEE SMC Society in 2017. He is currently the Vice President of Conferences and Meetings with IEEE Systems, Man and Cybernetics. He is an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, and the *Journal of Information Science*.

Jun Zhang (M'02–SM'08–F'17) received the Ph.D. degree in electrical engineering from the City University of Hong Kong, Hong Kong, in 2002.

From 2004 to 2016, he was a Professor with Sun Yat-sen University, Guangzhou, China. Since 2016, he has been with the South China University of Technology, Guangzhou, China, where he is currently a Cheung Kong Chair Professor. His current research interests include computational intelligence, cloud computing, big data, high performance computing, data mining, wireless sensor networks, operations research, and power electronic circuits. He has authored seven research books and book chapters, and over 100 technical papers in the above areas.

Prof. Zhang was a recipient of the China National Funds for Distinguished Young Scientists from the National Natural Science Foundation of China in 2011 and the First-Grade Award in Natural Science Research from the Ministry of Education, China, in 2009. He is currently an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, and the IEEE TRANSACTIONS ON CYBERNETICS. He is the Founding and Current Chair of the IEEE Guangzhou Subsection and IEEE Beijing (Guangzhou) Section Computational Intelligence Society Chapters. He is the Founding and Current Chair of the ACM Guangzhou Chapter.