Cloudde: A Heterogeneous Differential Evolution Algorithm and Its Distributed Cloud Version

Zhi-Hui Zhan, *Member, IEEE*, Xiao-Fang Liu, *Student Member, IEEE*, Huaxiang Zhang, Zhengtao Yu, Jian Weng, Yun Li, *Member, IEEE*, Tianlong Gu, and Jun Zhang, *Senior Member, IEEE*

Abstract—Existing differential evolution (DE) algorithms often face two challenges. The first is that the optimization performance is significantly affected by the ad hoc configurations of operators and parameters for different problems. The second is the long runtime for real-world problems whose fitness evaluations are often expensive. Aiming at solving these two problems, this paper develops a novel double-layered heterogeneous DE algorithm and realizes it in cloud computing distributed environment. In the first layer, different populations with various parameters and/or operators run concurrently and adaptively migrate to deliver robust solutions by making the best use of performance differences among multiple populations. In the second layer, a set of cloud virtual machines run in parallel to evaluate fitness of corresponding populations, reducing computational costs as offered by cloud. Experimental results on a set of benchmark problems with different search requirements and a case study with expensive design evaluations have shown that the proposed algorithm offers generally improved performance and reduced computational time, compared with not only conventional and a number of state-of-the-art DE variants, but also a number of other distributed DE and high-performing evolutionary algorithms. The speedup is significant especially on expensive problems, offering high potential in a broad range of real-world applications.

Index Terms—Differential evolution, evolutionary algorithm, adaptive migration strategy, cloud computing, heterogeneous parallelism

1 Introduction

A DISRUPTIVE technology fundamentally transforming the way that information and communication technology (ICT) services are delivered, cloud computing provides worldwide users a new dimension of computing resources conveniently, in the form of infrastructure, platform, and software as services via the Internet [1], [2]. While a trend is emerging to utilize evolutionary algorithms (EAs) for cloud resource scheduling and management [3], [4], [5], vice versa, this paper proposes to use cloud computing to enhance the power of EAs.

EAs offer a gradient-free global optimization tool for many real-world problems [6], [7], [8]. However, an EA often runs on a single computer or as a centralized algorithm [9]. With the increase in the complexity and scale of real-world problems, distributed EAs [10], [11], [12] have recently received much attention, as they utilize a set of computers/resources to enhance optimization capability, sophistication, and performance. Given the rapid development of the cloud computing paradigm and platforms, distributed computing has become readily available and affordable for realizing more powerful EAs.

This paper is focused on furthering these developments aiming at a sophisticated and cloud-ready EA. Differential evolution (DE) [13] is a widely-adopted type of EA whose performance is highly promising in many real-world applications. However, existing DE algorithms often face two challenges. The first is that the optimization performance is significantly affected by the ad hoc configurations of algorithm operators and parameters. In particular, the mutation operator and the crossover rate parameter need to vary with the application in hand [14], [15]. The second challenge is a relatively long runtime due to the iterative process, especially for real-world problems whose fitness evaluations (FEs) are often expensive.

This paper will address these issues using a heterogeneous parallelism and the cloud computing paradigm. In particular, different operators and/or parameters will be utilized in concurrent populations, so that various search requirements are met simultaneously in practice. Moreover, distributed and parallel computation will be used to speed up solutions. For this, a double-distributed cloud-ready DE algorithm, termed 'Cloudde', will be developed in this paper, both to robustness of global search for a range of problems and to reduce computational costs for complex or expensive problems.

With virtual machines (VMs) conveniently available on cloud, Cloudde offers three advantages when compared with physical resources: 1) The VMs are elastic, with fast

- *Z.-H. Zhan, X.-F. Liu, and J. Zhang are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China. E-mail: {cszhanzhh, csjun}@scut.edu.cn.*
- *H.-X. Zhang is with the School of Information Science and Engineering, Shandong Normal University, Jinan 250014, China.*
- *Z.-T. Yu is with the School of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650093, China.*
- *J. Weng is with the College of Information Science and Technology, Jinan University, Guangzhou 510632, China.*
- *Y. Li is with the College of Computer Science and Technology, Dongguan University of Technology, Dongguan 523808, China.*
- *T.-L. Gu is with the School of Computer Science and Engineering, Guilin University of Electronic Technology, Guilin 541004, China.*

Manuscript received 28 Dec. 2015; revised 12 July 2016; accepted 15 July 2016. Date of publication 3 Aug. 2016; date of current version 15 Feb. 2017. Recommended for acceptance by H. Jin.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2016.2597826

availability in time, cheapness in economic costs, and flexibility in billing; 2) The VMs are isolated, independent, and easy recoverable by fault tolerance techniques of the cloud platform; 3) The VMs are often ready for utilization once they are created and are easy for management.

The rest of the paper is organized as follows. In Section 2, DE and its centralized and distributed variants are reviewed. Section 3 develops the Cloudde algorithm. Experiments are conducted in Section 4 for in-depth comparisons. Section 5 presents a case study, an application to the design of a nonlinear circuit. Finally, conclusions are drawn and future work is highlighted in Section 6.

2 DE and Distributed DE

2.1 DE Framework

Like other EAs, DE is also a population-based iterative process to search for the global optimum. In initialization, N individuals are randomly created, each as $\boldsymbol{X}_i = [x_{i1}, x_{i2}, \ldots, x_{iD}]$, where i is the individual index, D is the problem dimension, and the coordinates of each dimension x_{id} is clamped in a certain search range $[x_{\min,d}, x_{\max,d}]$.

During the evolutionary process, each individual i first performs a mutation operation to generate a new individual vector $\boldsymbol{V}_i$. In the literature, various mutation schemes have been proposed [14]. For example, Eq. (1) shows the 'DE/best' mutation scheme that produces a new solution $\boldsymbol{V}_i$ based on the best individual $\boldsymbol{X}_{best}$, while Eq. (2) shows the 'DE/rand' mutation scheme that produces a new solution $\boldsymbol{V}_i$ based on a randomly selected individual $\boldsymbol{X}_{r1}$,

$$V_i = X_{best} + F(X_{r\mathbf{1}} - X_{r\mathbf{2}}) \tag{1}$$

$$V_i = X_{r\mathbf{1}} + F(X_{r\mathbf{2}} - X_{r\mathbf{3}}), \tag{2}$$

where parameter F is an '*amplification factor*' taking into account differential or directional information between two random individuals such as $r1$,$r2$, and $r3$. It should be noted that if the range of $\boldsymbol{V}_i$ excesses the search range, it will be kept in the corresponding boundary.

Following mutation, DE performs a crossover operation on vectors $\boldsymbol{X}_i$ and $\boldsymbol{V}_i$ to form a candidate solution $\boldsymbol{U}_i = [u_{i1}, u_{i2}, \ldots, u_{iD}]$, as:

$$u_{id} = \begin{cases} v_{id}, & \text{if } rand(0,1) \le CR \text{ or } d = r_i \\ x_{id}, & \text{otherwise} \end{cases}, \tag{3}$$

where CR is a '*crossover rate*' parameter and $r_i \in 1, 2, \ldots, D$ is a randomly selected dimension used to make sure that at least one dimension of $\boldsymbol{U}_i$ comes from $\boldsymbol{V}_i$.

Finally, a selection operation is applied on $\boldsymbol{X}_i$ and $\boldsymbol{U}_i$ for the better one to survive into the next generation, as per Eq. (4) for a minimization problem.

$$X_i = \begin{cases} U_i, & \text{if } fit(U_i) \le fit(X_i) \\ X_i, & \text{otherwise} \end{cases}. \tag{4}$$

All the individuals go through these three operations (mutation, crossover, and selection) generation by generation until a termination criterion is satisfied.

2.2 Current Art of DE

Here the current state of DE is reviewed in four main categories. The first category consists of adaptive DEs. Salman et al. [16] have developed a self-adaptive DE that tunes the F value automatically. Brest et al. [17] have developed an adaptive DE (jDE) to control both parameters F and CR. Two other popular adaptive DEs are SaDE and JADE, proposed by Qin et al. [18] and Zhang and Sanderson [19], respectively. Zhan and Zhang proposed a novel and efficient parameter learning adaptive DE (PLADE) [20], using the learning mechanism found in particle swarm optimization (PSO) [21], [22] to adjust F and CR adaptively. More adaptive DE variants are also proposed by Yu et al. [23], Wang et al. [24], and Liu et al. [25].

The second category is concerned with research on the mutation scheme. Islam et al. proposed a DE/current-to-gr_best mutation scheme [26]. Elsayed et al. [27] improved DE with a mix of different mutation schemes. Dorronsoro and Bouvry [28] proposed Gaussian mutation while Epitropakis et al. [29] proposed proximity-based mutation to improve DE.

The third category covers hybrid DE algorithms, such as those enhanced by opposition-based learning strategy [30] and local search strategy [31], [32]. Li et al. [33] enhanced DE with evolution path (EP) strategy to design a DEEP algorithm that can learn the historical information to help the algorithm search. In addition, random walk [34], restart procedure [35], mean distance-based selection strategy [36], cultural algorithm [37], and bacterial foraging algorithm [38] are also combined with DE.

The fourth category covers DE applied to real-world problems. In 1999, Storn proposed the use of DE for analog filter design [39]. Since then, DE has been applied to a diverse range of problems. To name but a few, timetable scheduling [40], control systems and robotics [41], bioinformatics [42], chemical engineering [43], network systems [44], topological active net optimization [45], and power electronic circuit (PEC) optimization [46].

2.3 Distributed DE

In the literature, Tasoulis et al. [47] have shown a parallel DE (PDE) algorithm using a ring topology structure. Kwedlo and Bandurski [48] and Kozlov and Samsonov [49] have designed distributed DE (DDE) algorithms also on a ring topology. Differently, Apolloni et al. [50] have proposed an island based DDE (IBDDE). Recently, Weber et al. [51], [52] have conducted investigations into PDE and IBDDE to test their performance with different parameters, offering good benchmarks. In applications, Salomon et al. [53] studied parallel DE for a medical image registration problem and De Falco et al. [54] studied DDE for a satellite image registration problem.

In this paper, we shall compare the Cloudde algorithm to be developed with both PDE and IBDDE, as they are two well-performed DDE variants according to the studies and parameters proposed in [51], [52].

3 The Cloudde Algorithm

3.1 Algorithm Framework

Based on the above analysis of existing DE algorithms and considering distributed island EA models, this section addresses the sensitivity issues of mutation and crossover in DE with a heterogeneous parallelism. In particular,

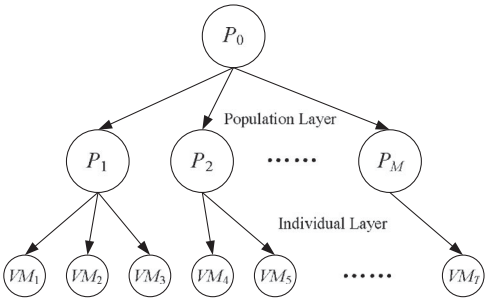


Fig. 1. Cloudde in a double-layered master-slave tree framework, which can also be viewed in a hub-and-spoke morphology, for heterogeneous populations and an efficient utilization of cloud computing resources.

different operators and/or parameters will be utilized in concurrent populations, such that various search requirements are met. With a flexible morphology, heterogeneous DE is developed here to realize in one of the following four forms:

1. As a sequential DE algorithm implemented on a single computer, where multiple populations accommodate various DE algorithms independently and then migration adaptively combines their advantages together for complex or demanding real-world problems;
2. As a distributed DE to tackle a problem concurrently, where a master node coordinates M multiple nodes in parallel hardware, forming a distributed layer that divide and conquer a complex problem and/or provides robustness through redundancy;
3. As a cloud DE, where all fitness evaluations of each population are distributed to a cloud computing layer, where VMs from the cloud resources pool take on these most computationally expensive tasks; and
4. A double-layered distributed DE combining realizations two and three, as illustrated in Fig. 1, for added power and sophistication.

Without loss of generality, the deliberations in this paper are focused on realization four shown in Fig. 1, where the '*population layer*' consists of M populations, P stands for a population and VM_i stands for one of the T VMs available. The multiple populations provide sophistication and robustness in dealing with various real-world problems or aspects even if no parallel hardware is used and in meeting various search requirements through different mutation schemes (e.g., the DE/best or DE/rand scheme) and/or different parameter settings (e.g., $CR = 0.1$ or $CR = 0.9$). The relatively independent populations are also expected to offer alternative or multi-objective solutions [11], [47], [48], [49], [50].

If these M populations are implemented on parallel hardware, this layer of distribution is controlled by a master node P_0, different from ring-based DE [47], [52]. All M populations are initiated to the same size as existing distributed DE variants [55], [56], [57]. The performance of each population can differ, and it is undesirable for all individuals in an inferior population to evaluate potentially wasteful solutions. Therefore, an adaptive migration strategy (AMS), to be detailed in Sections 3.2 and 3.3, is developed here to control the population size dynamically. During the evolutionary process, Cloudde determines which population is more suitable for the problem. The AMS thus guides individuals of a poor-performing population to migrate to better-performing populations. Consequently, the sizes of better-performing populations increase and the size of the poor-performing population decreases to reduce wasted fitness evaluations.

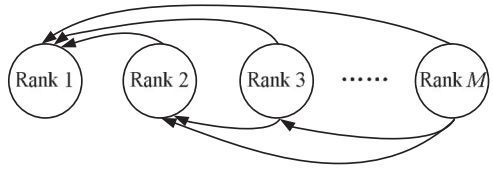


Fig. 2. AMS for the population layer in Cloudde.

The second layer, termed the '*individual layer*', distributes the individual evaluations of each DE population to a set of slave VMs flexibly to maximize the computational efficiency pertinent to the morphology of the cloud resource pool. As the populations change sizes during the evolution process under AMS, VMs are regrouped according to the population sizes and balance the resources demand and supply to reduce computational costs. The advantage of cloud computing is that it is easy to create sufficient VMs for Cloudde even though no sufficient physical machines may be available. Moreover, such an elastic virtual cloud makes it possible and easy to change the relationship between the DE populations and the VMs in implementing a cloud resource balancing strategy.

The T VMs available from the resources pool are grouped into M sets, with each corresponding to a DE population for fitness evaluations of the population, as shown in Fig. 1. At the beginning, the M populations are of the same population size and hence the T VM resources are grouped evenly. During the run, the population sizes change with migration and the VM resources have to be rebalanced. For this, a '*cloud resource balancing*' strategy is developed and detailed in Section 3.4.

3.2 AMS and Dynamic Migration Topology

With a heterogeneous island morphology, Cloudde allows the distributed populations to use different mutation schemes and different parameters. The populations migrate and mix under an AMS for a potentially improved evolution pool. The AMS lets some individuals migrate from a relatively poorly-performed population to a relatively well-performed population, so as to benefit reproduction from configurations of the well-performed population. There exist two issues to be addressed when designing an AMS. The first is to determine which population performs relatively better on a specific problem and the second is to define a dynamic migration topology. Here, the AMS calculates the mean fitness of each population and subsequently ranks the M populations from the best to the worst according to their mean fitness values. Then, the AMS assigns one individual in each population a probability to migrate to a better population, as shown in Fig. 2. That is, each of the 2nd to the M^{th} ranked populations migrates under a probability one individual to the 1st ranked population, each of the 3rd to the M^{th} ranked populations migrates under a probability another individual to the 2nd ranked population, and so on.

```
Procedure AMS
01 Begin
02   For m=1 to M Do
03     Calculate the mean fitness of all the individuals in population m
04   End of For
05   Rank the M populations according to the mean fitness
       //The better the mean fitness, the smaller the rank
06   For each two population m_i and m_j (1≤i, j≤M) Do
07     If (rank of m_j>rank of m_i) and (size of m_j>5) and (random(0, 1)<p_m)
08       Randomly select an individual from m_j and migrate it to m_i
09     End of If
10   End of For
11 End
```

Fig. 3. Pseudo-code of the AMS procedure.

3.3 Migration Occurrence and Control in AMS

The migration from a population occurs in every generation until the number of individuals is smaller than five, which is a critical size for any effective evolution towards a single objective, especially for the mutation operation [18]. We have experimentally studied various selection schemes for migration, including selecting the best individual, the worst individual, and a random individual. The results show that there is no significant difference and therefore, without loss of generality, we randomly select one individual to migrate.

A '*migration rate*', p_m, is used to control the probability at which migration occurs. If p_m is too small, migration seldom occurs and the populations cannot adequately share search information, but this is good at initial exploration. If p_m is too large, the migration can occur excessively and lead to the spread of local premature convergence, but this is good at final exploitation. The effects of p_m is carefully studied and detailed discussed in Section 4.5. The experimental results also show that small p_m is good for exploration while large p_m is good for exploitation. Therefore, a good choice is to let p_m be relative small at the beginning while growing large during the evolutionary process. Moreover, a nonlinearly increasing p_m may be more suitable for matching the nonlinearly evolutionary process. By considering these, a nonlinearly increasing p_m as:

$$p_m = 0.01 + 0.99\frac{\exp\left(\frac{10g}{G}\right) - 1}{\exp(10) - 1} \in [0.01, 1.00] \tag{5}$$

is adopted, where g and G are the current generation and the maximum generation pre-specified, respectively, resulting in small value 0.01 at the beginning and gradually increasing to 1.00 at the end of the evolutionary process.

The AMS procedure in pseudo-code is summarized in Fig. 3.

3.4 Cloud Resource Balancing

The individual layer of Cloudde is responsible for evaluating the individuals of each population on the VMs. As fitness evaluation is the most time-consuming task in an EA, the VMs help achieve maximum parallelism and the highest speed possible. In a cloud computing resources pool, load balancing helps distribute the calculations most efficiently.

With T VMs in the resources pool and M populations of the same size initially, the T VMs are evenly divided into M groups first. During the run, migration takes place and the population sizes change. Hence the VMs are regrouped to balance the cloud resource utilization according to the demands of different populations.

Assume that the initial population size of each population is N. Then there are $N \times M$ individuals in total and the average evaluation load of each VM is approximated as:

$$l = \frac{N \times M}{T}. \tag{6}$$

For the i^{th} population during the run, suppose its prevailing size is N_i and the number of VMs allocated to this population is T_i. Then the evaluation load of each VM with respect to this population is approximated as:

$$l_i = \frac{N_i}{T_i}. \tag{7}$$

Then using the euclidian metric we can define an average 'unbalance degree' u by the standard deviation as:

$$u = \frac{1}{M-1}\sqrt{\sum_{i=1}^{M}(l_i - l)^2}. \tag{8}$$

At the start of evolution, if T is a multiple of M, then $l_i = l$ and $u = 0$. If $M = 1$, u is also 0. During the run, the population sizes change, resulting in unbalanced load and hence a larger u. This is permitted until u becomes larger than a predefined threshold value $u_{\max}$. Then, to improve load balance and evaluation efficiency, the VMs are regrouped according to the population size proportion of each population. That is, with the following allocation to population i:

$$T_i = \left\lfloor \frac{N_i}{l} + 0.5 \right\rfloor = \left\lfloor \frac{N_i}{N} \cdot \frac{T}{M} + 0.5 \right\rfloor, \tag{9}$$

where $\lfloor x \rfloor$ stands for the maximum integer smaller than x.

In the case where the sum of $T_i (1 \le i \le M)$ is larger than T, we remove a VM from the group that has the maximum number of VMs. Conversely, if the sum is smaller than T, we add a remaining VM to the group that has the minimum number of VMs. This operation is to make the full use of the exact number of T VMs that are available for Cloudde.

Note that although individuals may also be assigned to all VMs evenly for fitness evaluations, the strategy of grouping VMs to subpopulations makes the distributed framework more structured, yielding simplicity with negligible overhead.

3.5 The Complete Algorithm

Without loss of generality, the complete Cloudde algorithm is illustrated in the most complete format, i.e., the double-layered distributed cloud realization four shown in Fig. 1, with M populations and T VMs. We adopt the message passing interface (MPI) protocol of the C programming language [58] to implement Cloudde. The pseudo-code of Cloudde is presented in Fig. 4, in three parts: P_0 (Fig. 4a), P_i ($1 \le i \le M$, Fig. 4b), and VM_j ($1 \le j \le T$, Fig. 4c).

A master node computer runs P_0, with other nodes running P_i and with VMs running VM_j concurrently. To begin, P_0 generates and initializes M equal-sized populations. In every generation, P_0 sends the M populations to $P_i (1 \le i \le M)$, as shown in Line 6 of Fig. 4a. Correspondingly, when P_i starts to run, it first waits to receive the population from P_0 in every generation, as shown in Line 3 of Fig. 4b.

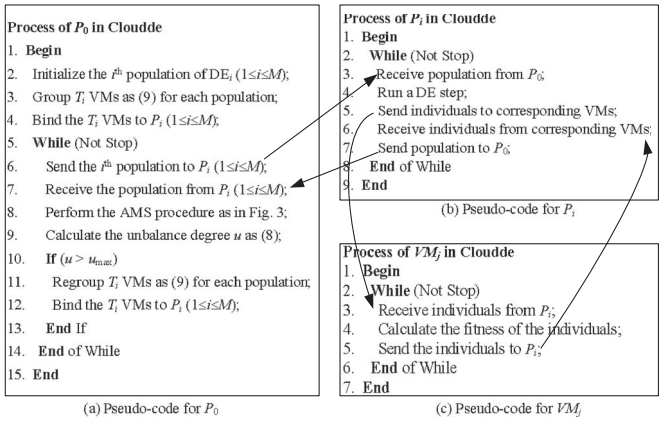


Fig. 4. Pseudo-code of the complete Cloudde.

Then P_i runs DE independently, including performing mutation and crossover on the population to evolve for a new population. To evaluate the fitness of individuals, P_i sends these individuals to the corresponding VMs to calculate, which upon competition send the individuals back to P_i, as shown in Lines 5 and 6 of Fig. 4b and the whole Fig. 4c. After P_i has collected all the evaluated fitness values from the VMs, it sends the population to P_0 for AMS and load re-balancing, as shown in Line 7 of Fig. 4b. At the same time, P_0 waits to receive all the M populations from $P_i (1 \le i \le M)$, as shown in Line 7 of Fig. 4a. Then, P_0 performs the AMS procedure for the M populations, as shown in Line 8 of Fig. 4a. The cloud resource balance strategy is also carried out from Line 9 to Line 13 of Fig. 4a. Although we can also let the M populations stay in each P_i during the process, this makes the AMS very difficult to control because the synchronization for migration among different nodes is time-consuming and difficult. Therefore, we send the population from P_i to P_0 and let P_0 do the AMS. This way, the individuals' migration in AMS only occurs in P_0, avoiding the overhead of migrating individuals among VMs. Moreover, as AMS uses randomly migration strategy, it can further reduce the overhead because it does not need to compare the quality of individuals.

4 Experimental Studies

4.1 Benchmarks and Algorithm Configurations

The experiments are conducted on two kinds of optimization problems. The first includes 13 functions ($f_1 - f_{13}$) as studied in [59] and listed in Table 1. The second kind is a real-world optimization problem in power electronic circuit design [60], [61], which is detailed in Section 5. The 13 test functions are selected because they span a wide range of characteristics like unimodal, multimodal, variable decoupled, and variable cross-coupled. Therefore, they are helpful in investigating the performance of DEs with different configurations when attempting problems with different characteristics. More importantly, these functions are of well-known typical characteristics that facilitate in-depth observations on how Cloudde may perform on various problems. Further, the PEC design is a complex problem in real-world applications, where its fitness evaluation is extremely expensive [61]. Therefore, it is useful to see how much it may benefit from Cloudde.

TABLE 1
The First Set of 13 Benchmark Functions

Name	Function	Range
Sphere	$f_1(x) = \sum_{i=1}^{D} x_i^2$	$[-100, 100]^{30}$
Sch's 2.22	$f_2(x) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	$[-10, 10]^{30}$
Quadric	$f_3(x) = \sum_{i=1}^{D} (\sum_{j=1}^{i} x_j)^2$	$[-100, 100]^{30}$
Sch's 2.21	$f_4(x) = \max_i (\lvert x_i \rvert, 1 \le i \le D)$	$[-100, 100]^{30}$
Step	$f_5(x) = \sum_{i=1}^{D} (\lfloor x_i + 0.5 \rfloor)^2$	$[-100, 100]^{30}$
Noise	$f_6(x) = \sum_{i=1}^{D} i x_i^4 + random[0,1)$	$[-1.28, 1.28]^{30}$
Rosenbrock	$f_7(x) = \sum_{i=1}^{D-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	$[-10, 10]^{30}$
Schwefel	$f_8(x) = \sum_{i=1}^{D} -x_i \sin(\sqrt{\lvert x_i \rvert}) + 418.9829 \times D$	$[-500, 500]^{30}$
Rastrigin	$f_9(x) = \sum_{i=1}^{D} [x_i^2 - 10\cos(2\pi x_i) + 10]$	$[-5.12, 5.12]^{30}$
Ackley	$f_{10}(x) = -20\exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}) - \exp(\frac{1}{D}\sum_{i=1}^{D} \cos 2\pi x_i) + 20 + e$	$[-32, 32]^{30}$
Griewank	$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos(\frac{x_i}{\sqrt{i}}) + 1$	$[-600, 600]^{30}$
Generalized penalized	$f_{12}(x) = \frac{\pi}{D}\{10\sin^2(\pi y_1) + \sum_{i=1}^{D-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_D - 1)^2\} + \sum_{i=1}^{D} u(x_i, 10, 100, 4)$ where $y_i = 1 + \frac{1}{4}(x_i + 1)$, $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \le x_i \le a \\ k(-x_i - a)^m, & x_i < -a \end{cases}$	$[-50, 50]^{30}$
	$f_{13}(x) = 0.1\{\sin^2(3\pi x_1) + \sum_{i=1}^{D-1}(x_i - 1)^2[1 + \sin^2(3\pi x_{i+1})] + (x_D - 1)^2[1 + \sin^2(2\pi x_D)]\} + \sum_{i=1}^{D} u(x_i, 5, 100, 4)$	$[-50, 50]^{30}$

In order to test the characteristics of Cloudde first as a new sequential DE algorithm, we compare it with conventional DEs, and also with a number of improved state-of-the-art DEs, like jDE [17], SaDE [18], JADE [19], and CoDE [24]. As mutation scheme and crossover rate CR are sensitive to the algorithm performance, we configure conventional DEs with various mutation schemes and CR values. Firstly, both the DE/best and DE/rand mutation schemes are investigated because they are the two most frequently used schemes and present significantly different abilities in unimodal and multimodal functions. Secondly, two representative values as $CR = 0.1$ and $CR = 0.9$ are investigated because they represent significantly small and significantly large values that may be required by different functions. Therefore, these four conventional DE variants are denoted as DE/best/0.1, DE/best/0.9, DE/rand/0.1, and DE/rand/0.9, respectively. Moreover, two DDE variants named PDE [47] and IBDDE [50], and two other population-based algorithms, the adaptive particle swarm optimization (APSO) [62] and the evolution strategy with covariance matrix adaptation (CMA-ES) [63], are used as contenders.

Population size is set to 50 and the amplification factor F is set to 0.5 for the conventional DEs. For DDEs, APSO, and CMA-ES, the algorithm settings are according to the proposals in their original papers.

Specifically for Cloudde, we set $N = 50$ and $M = 4$ and hence the overall population size is $N \times M = 50 \times 4 = 200$. The four population nodes are configured as a conventional

TABLE 2
Comparisons Among Cloudde and Conventional DE Variants

Function	Cloudde	DE/best/0.1	DE/best/0.9	DE/rand/0.1	DE/rand/0.9
f_1	4.27E-139±1.71E-138 (3)*	3.24E-142±8.94 E-142 (2/−)	**2.93E-292±0** (1/−)	3.26E-89±3.25E-89 (5/+)	9.04E-106±3.65E-105 (4/+)
f_2	7.34E-20±1.98E-19 (4)	**6.43E-79±7.17E-79** (1/−)	6.68E-01±2.54E+00 (5/+)	1.73E-51±1.22E-51 (3/−)	1.71E-56±4.86E-56 (2/−)
f_3	**8.03E-27±2.00E-26** (1)	1.15E+02±4.98E+01 (3/+)	1.67E+02±9.13E+02 (4/+)	2.33E+03±4.79E+02 (5/+)	9.28E-15±1.49E-14 (2/+)
f_4	7.08E-02±7.98E-02 (2)	1.32E+01±4.35 E+00 (5/+)	1.03E+01±4.24E+00 (3/+)	**1.94E-10±5.70E-11** (1/−)	1.06E+01±4.51E+00 (4/+)
f_5	**0±0** (1)	**0±0** (1/≈)	9.43E+02±7.34E+02 (5/+)	**0±0** (1/≈)	**0±0** (1/≈)
f_6	**1.94E-03±1.47E-03** (1)	2.63E-03±6.22E-04 (2/+)	4.45E-02±4.05E-02 (5/+)	4.57E-03±1.39E-03 (4/+)	2.69E-03±8.13E-04 (3/+)
f_7	**5.96E-22±3.25E-21** (1)	3.04E+01±1.97E+01 (5/+)	1.20E+00±1.86E+00 (2/+)	2.36E+01±1.59E+00 (4/+)	1.72E+01±6.11E+00 (3/+)
f_8	**3.82E-04±0** (1)	5.50E+02±2.28E+02 (3/+)	4.40E+03±7.47E+02 (5/+)	**3.82E-04±0** (1/≈)	2.50E+03±6.03E+02 (4/+)
f_9	**0±0** (1)	1.23E+00±1.37E+00 (3/+)	7.82E+01±1.76E+01 (5/+)	**0±0** (1/≈)	1.27E+01±4.13E+00 (4/+)
f_{10}	4.97E-15±1.79E-15 (3)	1.22E-14±2.46E-15 (4/+)	1.15E+01±2.15E+00 (5/+)	**4.14E-15±0** (1/−)	**4.14E-15±0** (1/−)
f_{11}	**0±0** (1)	1.40E-03±3.81E-03 (3/+)	1.25E-01±1.55E-01 (5/+)	**0±0** (1/≈)	1.89E-03±4.37E-03 (4/+)
f_{12}	**1.57E-32±2.78E-48** (1)	**1.57E-32±2.78E-48** (1/≈)	5.01E+00±6.35E+00 (5/+)	**1.57E-32±2.78E-48** (1/≈)	3.11E-02±1.06E-01 (4/+)
f_{13}	**1.35E-32±5.57E-48** (1)	**1.57E-32±2.78E-48** (1/≈)	6.64E+00±1.19E+01 (5/+)	**1.35E-32±5.57E-48** (1/≈)	1.20E-01±6.57E-01 (4/+)
Rank	**21/13 = 1.62 (1)**	34/13 = 2.62 (3)	55/13 = 4.23 (5)	29/13 = 2.23 (2)	40/13 = 3.08 (4)
Number of '+'/'≈'/'−'		8/2/3	12/0/1	4/5/4	8/3/2
Algorithm	Cloudde	DE/best/0.1	DE/best/0.9	DE/rand/0.1	DE/rand/0.9

*The number is the rank according to the mean results '+', '≈', and '−' mean Cloudde performs significantly better than, similar to, and significantly worse than the corresponding algorithm, respectively.

DE with $F = 0.5$ and with DE/best/0.1, DE/best/0.9, DE/rand/0.1, and DE/rand/0.9 for each node. Starting from 50, each population size is dynamically controlled by the AMS during the evolutionary process and the migration rate p_m varies according to Eq. (5).

When testing in a cloud environment, the individual layer of nodes for fitness evaluations are dynamically distributed to 16 VMs in the cloud computing resources pool. The load imbalance parameter $u_{\max}$ is set to 7 and its influence on computational time is investigated in Section 4.7.

As different algorithms have different population sizes, for a fair comparison, all the algorithms use the same maximum number of FEs of 3.0×10^5 for each test function, as all the functions are of $D = 30$ dimensions [59]. In order to make the comparisons more readable, the absolute average and the standard deviation of the error values are used, and all the optimum targets are 0. For the purpose of reducing statistical errors, each test problem is attempted 30 times independently and the average results are used in the comparison. Moreover, Wilcoxon's rank sum tests with a significant level $\alpha = 0.05$ are conducted to make the comparison statistically sound.

4.2 Comparisons with Conventional DE Variants

Cloudde is first implemented on a single computer herein for tests as an ordinary, sequential DE algorithm, where the heterogeneous populations run the 4 DE algorithms above: DE/best/0.1, DE/best/0.9, DE/rand/0.1, and DE/rand/0.9. However, it is worth noting that all four realizations of Cloude deliver very similar results in terms of search quality. Therefore, comparisons are only made between results of Cloudde in realization four and results of conventional DEs in Table 2, where the best results are in **boldfaces**. The symbols '+', '≈', and '−' are used to indicate that Cloudde performs significantly better than, similar to, and significantly worse than the DE compared, respectively, when measured by the Wilcoxon's rank sum tests with significance level $\alpha = 0.05$.

The comparisons in Table 2 show that the performance of conventional DEs on different problems is seriously affected by the operators and parameters, indicating that not a single DE variant can perform well on all kinds of problems. As Cloudde uses multiple populations with different operators and different parameters, it is expected that Cloudde is suitable for different kinds of problems. Indeed, the experimental results support this intuition by showing that Cloudde performs well on all the 13 functions tested. The results in Table 2 show that in all the 30 independent runs Cloudde can obtain the global optimum for all the 13 functions, being unimodal or multimodal functions and being functions with cross-coupled variables or decoupled variables. For further comparison, we rank the algorithms according to the average results obtained. The results show that Cloudde has achieved a good rank on most of the functions and the first place on $f_3, f_5, f_6, f_7, f_8, f_9$, and f_{11}. The mean rank of Cloudde on all the 13 functions is 1.62, indicating that it is the best algorithm, comparing with 2.23 achieved by DE/rand/0.1, 2.62 by DE/best/0.1, 3.08 by DE/rand/0.9, and 4.23 by DE/best/0.9. The Wilcoxon's rank sum tests also show that generally Cloudde significantly outperforms the compared conventional DEs with different configurations.

From the comparisons, we can also observe that DE using a specific parameter or operator can perform well on a number of functions but perform poorly on other functions. As Cloudde can combine the features of different algorithm configurations, it is more suitable for different kinds of functions. Experimental studies have shown these advantages of Cloudde.

4.3 Comparisons with State-of-the-Art DE Variants

The performance of Cloudde is further evaluated by comparing with a number of state-of-the-art DE variants. These algorithms are jDE [17], SaDE [18], JADE [19], and CoDE [24]. The mean solution error and standard deviation of the 30 independent runs are compared in Table 3.

It can be observed from Table 3 that all these DE algorithms have the capability to obtain the global optimum for most of the benchmark functions. However, for the simple unimodal functions like f_1, CoDE and jDE perform poorer than SaDE, JADE, and Cloudde. This may be due to that jDE

TABLE 3
Comparisons Among Cloudde and the State-of-the-Art DE Variants

Function	Cloudde	jDE	SaDE	JADE	CoDE
f_1	**4.27E-139±1.71E-138** (1)	1.17E-66±2.39E-66 (4/+)	1.05E-132±2.68E-132 (2/+)	4.63E-129±2.22E-128 (3/+)	4.81E-54±6.47E-54 (5/+)
f_2	7.34E-20±1.98E-19 (5)	1.92E-39±2.53E-39 (2/−)	**1.86E-69±1.31E-69** (1/−)	1.84E-34±9.91E-34 (3/−)	1.45E-29±1.22E-29 (4/−)
f_3	8.03E-27±2.00E-26 (2)	2.69E-07±6.26E-07 (4/+)	4.51E-16±5.89E-16 (3/+)	**5.40E-43±1.37E-42** (1/−)	2.44E-01±2.80E-01 (5/+)
f_4	7.08E-02±7.98E-02 (3)	3.53E-01±3.73E-01 (5/+)	1.48E-01±2.78E-01 (4/≈)	**2.85E-16±9.27E-16** (1/−)	1.91E-10±1.03E-10 (2/−)
f_5	**0±0 (1)**	**0±0** (1/≈)	**0±0** (1/≈)	**0±0** (1/≈)	**0±0** (1/≈)
f_6	1.94E-03±1.47E-03 (3)	3.26E-03±7.29 E-04 (4/+)	1.45E-03±4.09E-04 (2/≈)	**1.25E-03±4.64E-04** (1/−)	3.87E-03±1.19E-03 (5/+)
f_7	**5.96E-22±3.25E-21** (1)	6.44E+00±2.84E+00 (5/+)	3.99E-01±1.22E+00 (3/+)	9.30E-01±1.71E+00 (4/+)	0.002165±0.001319 (2/+)
f_8	**3.82E-04±0** (1)	**3.82E-04±0** (1/≈)	2.09E+02±1.55E+02 (5/+)	1.11E+02±1.24E+02 (4/+)	**3.82E-04±0** (1/≈)
f_9	**0±0 (1)**	**0±0** (1/≈)	2.39E+00±1.71E+00 (5/+)	**0±0** (1/≈)	**0±0** (1/≈)
f_{10}	4.97E-15±1.79E-15 (4)	**4.14E-15±0** (1/−)	**4.14E-15±0** (1/−)	5.56E-15±1.77E-15 (5/≈)	**4.14E-15±0** (1/−)
f_{11}	**0±0** (1)	**0±0** (1/≈)	**0±0** (1/≈)	1.97E-03±4.21E-03 (5/+)	**0±0** (1/≈)
f_{12}	**1.57E-32±2.78E-48** (1)	**1.57E-32±2.78E-48** (1/≈)	3.46E-03±1.89E-02 (5/≈)	**1.57E-32±2.78E-48** (1/≈)	**1.57E-32±2.78E-48** (1/≈)
f_{13}	**1.35E-32±5.57E-48** (1)	**1.35E-32±5.57E-48** (1/≈)	**1.35E-32±5.57E-48** (1/≈)	**1.35E-32±5.57E-48** (1/≈)	**1.35E-32±5.57E-48** (1/≈)
rank	**25/13 = 1.92 (1)**	31/13 = 2.38 (3)	34/13 = 2.62 (5)	31/13 = 2.38 (3)	30/13 = 2.31 (2)
Number of '+'/'≈'/'−'		5/6/2	5/6/2	4/5/4	4/6/3
Algorithm	Cloudde	jDE	SaDE	JADE	CoDE

'+', '≈', and '−' mean Cloudde performs significantly better than, similar to, and significantly worse than the corresponding algorithm, respectively.

uses the DE/rand mutation scheme which has slow convergence, while JADE uses a greedy mutation scheme (DE/current-to-*p*best) [19], which is powerful on unimodal functions, and SaDE hybridizes both the random and greedy schemes. Cloudde is also suitable for unimodal function because it uses both DE/best and DE/rand mutation schemes, and the AMS can guide individuals to migrate from the DE/rand populations to the DE/best populations, making Cloudde perform well on unimodal functions (this will be investigated in Section 0-0). Although CoDE also uses different mutation schemes, it incurs the same computational burden to the different schemes, resulting in poorer performance than Cloudde on a simple unimodal function like f_1.

In general, the results in Table 3 have shown that Cloudde is significantly better than jDE, SaDE, JADE, and CoDE on f_1 and yields better or very competitive performance on other unimodal functions. For the multimodal functions f_7-f_{13}, only can Cloudde obtain the global optima of all the multimodal functions, while jDE is trapped by f_7, SaDE by f_7, f_8, f_9, and f_{12}, JADE by f_7, f_8, and f_{11}, and CoDE by f_7. Moreover, Cloudde performs the best on the multimodal functions $f_7, f_8, f_9, f_{11}, f_{12}$, and f_{13}.

Considering all the 13 functions with unimodal, multimodal, decoupled, and cross-coupled variable characteristics, the Wilcoxon's rank sum tests results show that Cloudde does significantly better than jDE, SaDE, JADE, and CoDE on 5, 5, 4, and 4 functions respectively, while does significantly worse than them on 2, 2, 4, and 3 functions respectively. Therefore, Cloudde generally does better than jDE, SaDE, JADE, and CoDE. Moreover, the average rank over all the 13 functions show that Cloudde has the best rank and followed by CoDE, JADE, jDE, and SaDE.

4.4 Comparisons with DDEs, APSO, and CMA-ES

Table 4 presents the results of Cloudde compared with two DDEs and two other population-based optimization

TABLE 4
Comparisons Among Cloudde, DDEs, APSO, and CMA-ES

Function	Cloudde	PDE	IBDDE	APSO	CMA-ES
f_1	**4.27E-139±1.71E-138** (2)	1.96E-65±8.68E-65 (3/+)	4.34E-16±3.94E-16 (4/+)	**1.10E-224±0** (1/−)	1.15E-15±5.30E-16 (5/+)
f_2	7.34E-20±1.98E-19 (2)	1.00E+00±3.05E+00 (5/+)	8.02E-08±4.82E-08 (4/+)	**2.90E-124±1.53E-123** (1/−)	3.09E-11±4.61E-12 (3/+)
f_3	8.03E-27±2.00E-26 (2)	1.11E-10±2.95E-10 (4/+)	7.50E-01±4.98E-01 (5/+)	**9.07E-17±1.41E-16** (1/−)	3.24E-15±1.10E-15 (3/+)
f_4	7.08E-02±7.98E-02 (3)	1.49E+01±4.68E+00 (5/+)	1.63E-01±2.68E-01 (4/≈)	3.43E-05±7.98E-05 (2/−)	**3.45E-11±6.45E-12** (1/−)
f_5	**0±0 (1)**	7.47E+02±5.25E+02 (5/+)	**0±0** (1/≈)	**0±0** (1/≈)	**0±0** (1/≈)
f_6	**1.94E-03±1.47E-03** (1)	2.82E-01±1.82E-01 (5/+)	1.04E-02±2.18E-03 (3/+)	4.05E-03±1.86E-03 (2/+)	6.08E-02±2.08E-02 (4/+)
f_7	**5.96E-22±3.25E-21** (1)	6.64E-01±1.51E+00 (4/+)	1.29E+01±1.13E+00 (5/+)	6.18E-02±8.35E-02 (3/+)	2.81E-15±1.03E-15 (2/+)
f_8	**3.82E-04±0** (1)	4.67E+03±5.09E+02 (3/+)	6.00E+03±7.75E+02 (5/+)	3.82E-04±3.11E-06 (2/≈)	2.90E+03±1.20E+03 (4/+)
f_9	**0±0 (1)**	9.15E+01±3.06E+01 (4/+)	1.78E+02±1.13E+01 (5/+)	2.13E-14±7.89E-14 (2/+)	7.07E-01±5.40E-01 (3/+)
f_{10}	4.97E-15±1.79E-15 (1)	1.10E+01±2.66E+00 (4/+)	6.82E-09±3.29E-09 (3/+)	1.56E-14±2.37E-14 (2/+)	1.87E+01±6.34E+01 (5/+)
f_{11}	**0±0** (1)	3.11E-01±5.98E-01 (5/+)	2.47E-04±1.35E-03 (3/+)	1.76E-02±1.95E-02 (4/+)	2.14E-15±1.03E-15 (2/+)
f_{12}	**1.57E-32±2.78E-48** (1)	2.68E+00±3.57E+00 (5/+)	6.94E-17±9.76E-17 (3/+)	9.78E-31±4.85E-30 (2/+)	3.19E-15±1.16E-15 (4/+)
f_{13}	**1.35E-32±5.57E-48** (1)	6.13E+00±1.01E+01 (5/+)	3.34E-16±3.35E-16 (2/+)	3.66E-03±5.27E-03 (4/+)	3.26E-15±9.65E-16 (3/+)
rank	**18/13 = 1.38 (1)**	57/13 = 4.38 (5)	47/13 = 3.62 (4)	27/13 = 2.08 (2)	40/13 = 3.08 (3)
Number of '+'/'≈'/'−'		13/0/0	11/2/0	7/2/4	11/1/1
Algorithm	Cloudde	PDE	IBDDE	APSO	CMA-ES

'+', '≈', and '−' mean Cloudde performs significantly better than, similar to, and significantly worse than the corresponding algorithm, respectively.

TABLE 5
Comparisons Among Cloudde with its Variants Using Different Migration Rates

Func	Cloudde	Cloudde-w/o-AMS	Cloudde ($p_m = 0.01$)	Cloudde ($p_m = 0.05$)	Cloudde ($p_m = 0.1$)	Cloudde ($p_m = 0.5$)
f_1	4.27E-139±1.71E-138	5.54E-64±3.03E-63 (+)	2.16E-112±1.18E-111 (+)	7.81E-198±0 (−)	2.65E-233±0 (−)	**1.35E-264±0** (−)
f_2	7.34E-20±1.98E-19	1.10E-18±4.70E-19 (+)	1.03E-19±1.88E-19 (+)	1.84E-22±1.01E-21 (−)	6.02E-67±3.30E-66 (−)	**1.36E-85±7.43E-85** (−)
f_3	8.03E-27±2.00E-26	3.19E-11±9.98E-11 (+)	3.99E-22±1.43E-21 (+)	6.09E-38±2.27E-37 (−)	2.57E-44±1.37E-43 (−)	**2.92E-52±1.31E-51** (−)
f_4	7.08E-02±7.98E-02	9.79E-02±1.89E-02 (+)	1.01E-01±3.34E-02 (+)	5.46E-02±1.45E-01 (−)	2.45E-12±6.85E-12 (−)	**8.08E-16±3.92E-15** (−)
f_5	**0±0**	**0±0** (≈)	**0±0** (≈)	**0±0** (≈)	**0±0** (≈)	6.8±5.25 (+)
f_6	1.94E-03±1.47E-03	9.00E-03±2.11E-03 (+)	4.06E-03±2.83E-03 (+)	**1.40E-03±4.50E-04** (≈)	1.40E-03±6.65E-04 (≈)	2.63E-03±1.27E-03 (+)
f_7	**5.96E-22±3.25E-21**	2.11E+00±4.72E+00 (+)	9.94E-01±3.41E+00 (+)	3.99E-01±1.22E+00 (≈)	1.06E+00±1.79E+00 (≈)	1.20E+00±1.86E+00 (≈)
f_8	**3.82E-04±0**	7.90E+00±3.00E+01 (≈)	3.82E-04±1.23E-12 (≈)	1.18E+01±3.61E+01 (+)	7.61E+01±1.05E+02 (+)	1.59E+03±5.11E+02 (+)
f_9	**0±0**	1.14E-06±3.18E-06 (+)	1.20E-08±6.58E-08 (≈)	5.92E-17±3.24E-16 (≈)	4.64E-01±1.33E+00 (+)	2.64E+01±8.49E+00 (+)
f_{10}	**4.97E-15±1.79E-15**	1.27E-14±1.77E-15 (+)	8.05E-15±1.71E-15 (+)	8.40E-15±1.96E-15 (+)	2.08E-14±3.31E-14 (+)	1.99E+00±5.78E-01 (+)
f_{11}	**0±0**	**0±0** (≈)	**0±0** (≈)	5.75E-04±2.21E-03 (+)	4.93E-04±1.88E-03 (+)	4.75E-02±5.44E-02 (+)
f_{12}	**1.57E-32±2.78E-48**	3.82E-25±2.09E-24 (≈)	**1.57E-32±2.78E-48** (≈)	3.12E-31±9.64E-31 (+)	2.14E-31±3.31E-31 (+)	2.11E-01±4.39E-01 (+)
f_{13}	**1.35E-32±5.57E-48**	**1.35E-32±5.57E-48** (≈)	**1.35E-32±5.57E-48** (≈)	5.74E-32±1.95E-31 (+)	5.73E-31±1.62E-30 (+)	1.65E-01±3.73E-01 (+)
Number of '+'/'≈'/'−'		8/5/0	7/6/0	5/4/4	6/3/4	8/1/4
Algor.	Cloudde	Cloudde-w/o-AMS	Cloudde ($p_m = 0.01$)	Cloudde ($p_m = 0.05$)	Cloudde ($p_m = 0.1$)	Cloudde ($p_m = 0.5$)

'+', '≈', and '−' mean Cloudde performs significantly better than, similar to, and significantly worse than the corresponding algorithm, respectively.

algorithms. The results show that Cloudde does much better than both PDE [47] and IBDDE [50], and thus extends current DDE to solving a wider range of problems with various characteristics.

The better performance is likely brought about by the AMS. In PDE, each subpopulation is with a fixed size and sends its best solution with probability to its neighbor subpopulation in a unidirectional ring topology. Although such a migration helps good solution propagate among different populations, the one-by-one ring topology makes the information propagation slow. In IBDDE, periodicity migration happens between two neighboring subpopulations defined by a ring topology similar to PDE. In the migration, the migrating individuals and the replaced individuals are selected by two different functions, respectively. However, the migration in both PDE and IBDDE does not consider the performance of different subpopulations or change the population size. Differently, AMS in Cloudde migrates randomly selected individuals in a poor-performing subpopulation to well-performing subpopulations. This allows the expansion of the subpopulation with good performance and the reduction of the subpopulation with poor performance. Since the competition begins with fair initialization and the same population size for all subpopulations, a subpopulation of higher performance is regarded as more competitive. The AMS helps a higher computational budget be assigned to relatively good subpopulations, as well as increases the subpopulation diversity to enhance performance. Thus, it is unsurprising that Cloudde delivers better results than PDE and IBDDE. Further analysis of the effect of the AMS and a varying population size will be discussed in Sections 4.5 and 4.6 in detail.

Comparing with non-DE algorithms APSO [62] and CMA-ES [63], the results also show that Cloudde delivers very promising performance. For the unimodal functions, APSO is the most efficient algorithm and offers the highest accuracy, which may be due to its addictiveness and fast convergence [62]. However, Cloudde offers a higher efficiency than both APSO and CMA-ES when dealing with multimodal functions, and shows promising performance for functions with coupled variables. Over all the 13 functions, APSO obtains best results on 4 functions, CMA-ES on 2 functions, while Cloudde on 9 functions.

By comparison on the Wilcoxon's rank sum tests, Cloudde does significantly better than PDE, IBDDE, APSO, and CMA-ES on 13, 11, 7, and 11 functions respectively, while these four algorithms significantly outperform Cloudde on 0, 0, 4, and 1 functions respectively. Therefore, Cloudde is very promising for problems with different or heterogeneous characteristics.

4.5 Contribution of the AMS

In this section, we investigate the influence of the AMS on the Cloudde performance and analyze contributions of dynamic population sizing under the AMS. First, we investigate Cloudde without incorporating an AMS. Denote this special version as Cloudde-w/o-AMS, whose population sizes remain unchanged during the evolutionary process. Then we investigate the effect of the migration rate p_m in detail.

The results are presented in Table 5. Comparing Cloudde with Cloudde-w/o-AMS, the results show that the AMS indeed helped Cloudde perform better on all the 13 functions. Without using the AMS, Cloudde wasted computation on poor-performing populations and obtained only low accuracy solutions on the simple unimodal function f_1, for example. When the AMS was used, Cloudde migrated individuals to well-performing populations and improved computational efficiency and efficacy, thereby resulting in a higher convergence speed. Moreover, the AMS offered significant contribution to Cloudde when solving multimodal functions. The advantages of the AMS are much more evident on the Rosenbrock (f_7) and Schwefel functions (f_8), where Cloudde-w/o-AMS was trapped by local optima. One reason is that the algorithm spread computational resources to populations that were not suitable to these problems.

Concluding from the above, the AMS for different populations improves efficacy of a distributed algorithm.

However, how to control the migration would be a salient issue further to address. The results in Table 5

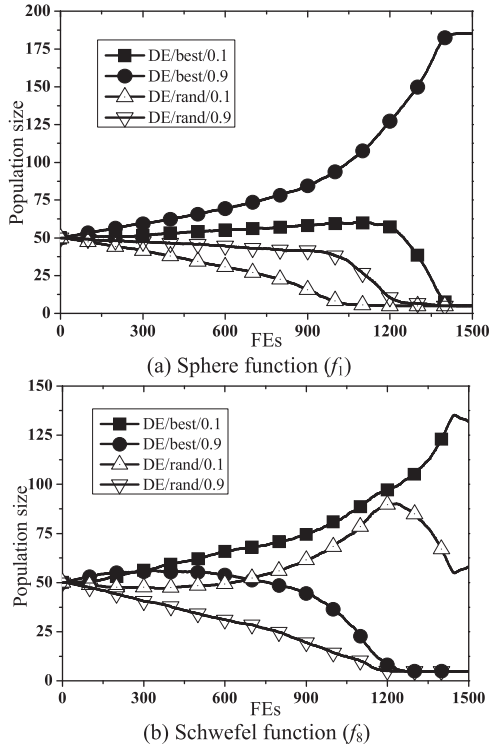


Fig. 5. Dynamic population sizing during heterogeneous evolution. (a) Sphere function (f_1). (b) Schwefel function (f_8).

appear to support this intuition, where p_m has been tested for 0 (i.e., Cloudde-w/o-AMS), 0.01, 0.05, 0.1, and 0.5. It is seen that a relatively small p_m, e.g., 0.01 or 0.05, offers better performance on multimodal than unimodal functions. As p_m increases to 0.1 or 0.5, migrations become more frequent, and populations of faster convergence such as those with the DE/best mutation scheme or a large *CR* become dominant. This results in better solutions on unimodal functions, as the evolution processes lose diversity on multimodal functions and are trapped in local optima, as revealed by Table 5. Therefore, Cloudde has been designed to take into account such effects through the nonlinearly time-varying p_m of Eq. (5), which starts with a small value and gradually ends with a more sharply increased value. In an early stage of evolution, each subpopulation performs similarly and differences between different subpopulations may be caused by randomness. A small p_m thus allows the subpopulations to maintain their own evolutionary behaviors and to have more time to improve. This avoids a too large difference between well-performing and poor-performing subpopulations. With the evolution progressing, each subpopulation converges within themselves and hence differences among them expand. A larger p_m moves more poor individuals to well-performing subpopulations and diverse the nearly converging population, so as to give the evolution further space to improve.

Experimental results have shown that this automatic rate offers significantly better performance than fixing p_m to 0, 0.01, 0.05, 0.1, and 0.5 on 8, 7, 5, 6, and 8 functions, although significantly worse than them on 0, 0, 4, 4, and 4 functions, respectively. More importantly, owing to the above advantages, the automatic rate appears to be less sensitive to the characteristics of different problems and hence performs well on many of the tested functions.

4.6 Contribution of Dynamic Population Sizing

As can be observed from the results in Table 2, the DE/best mutation scheme may perform better on some functions while DE/rand better on some other. Some functions may require a larger *CR* while some others require a smaller *CR*. To quantify and visualize these, we plot the dynamic population sizes during the evolution process optimizing the Sphere function (f_1) and the Schwefel function (f_8) in Figs. 5a and 5b, respectively. The Sphere and Schwefel functions are chosen without loss of generality because they are two typical unimodal and multimodal functions.

From Fig. 5a, it can be observed that the population size of DE/best/0.9 kept increasing during the unimodal evolutionary process, while other populations declined when converging. This confirms that the DE/best mutation scheme and a large *CR* offer directional dominance for added speed, which echoes the results in Table 2, where DE/best/0.9 was shown to be the best performing algorithm on this unimodal function. This also confirms the efficacy of the AMS in making the best use of performance differences between the population nodes for automatic migration and enhanced generational performance.

When optimizing a multimodal function such as the Schwefel function, however, the AMS is anticipated to facilitate a contraction of the DE/best/0.9 population. This is confirmed by Fig. 5b. The figure also shows that a smaller *CR* such as that in DE/rand/0.1 and DE/best/0.1 helps exploring multiple troughs/peaks, resulting in larger populations. The curves also show that in the final phase of evolution the population size of DE/rand/0.1 decreased while DE/best/0.1 still increased. This may be due to that the global optimum was found and hence DE with a greedy mutation scheme dominated the convergence.

5 Case Study: Application to the Design of a Nonlinear Circuit

In this section, we shall apply Cloudde to a real-world problem, the design of power electronic circuit, whose fitness evaluation is computationally expensive. We evaluate the Cloudde's suitability to a distributed cloud computing resources, especially the second layer of the double-layered distributed framework that further distributes the individuals to a set of dynamic VMs, and test the efficiency of the cloud layer for fitness calculations that significantly reduces the computational time in parallel in solving real-world problems.

The PEC design problem described in [60] is studied here. The circuit is a buck regulator with overcurrent protection, as shown in Fig. 6. The design task involves sizing seven components for four objectives under seven constraints.

We compare the CPU time of the comprehensive Cloudde (realization four) and of a Cloudde without VMs (realization two, denoted Cloudde-w/o-VMs). The results of a combined single fitness value as per Eq. (7) in [60] and of the CPU time are compared in Table 6, which are the mean results of 30 independent runs. The Cloudde parameters are the same as the ones described in Section 4.1 and the maximum generation is set to 30 due to the relatively long CPU time in evaluating solutions to the PEC.

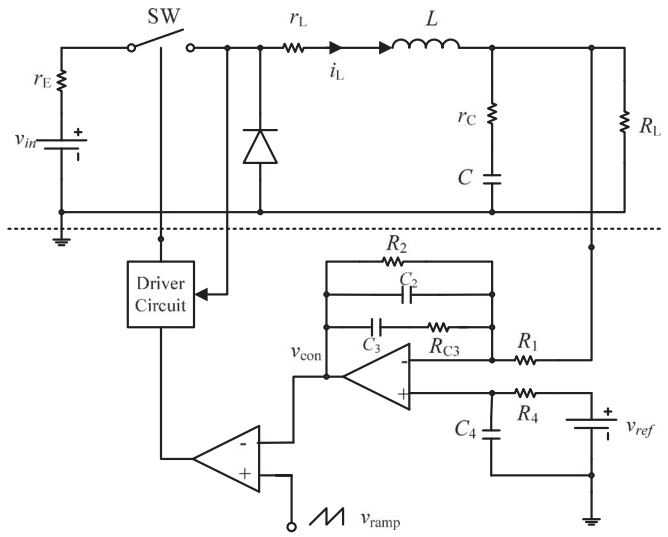


Fig. 6. A buck regulator with overcurrent protection designed via Cloudde.

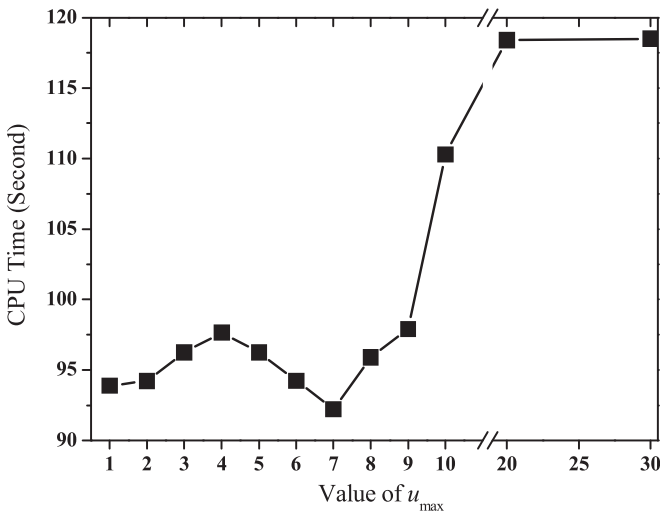


Fig. 7. The influence of $u_{\max}$ on the Cloudde performance.

Therefore, the total number of FEs is $200 \times 30 = 6,000$, which is much smaller than the 15,000 used in [60].

The results in Table 6 show that the fitness values obtained by Cloudde and Cloudde-w/o-VMs are almost the same, while slight differences are anticipated. This is due to that the two Cloudde variants have the same algorithmic logic with the only difference being that Cloudde distributes individuals to multiple VMs for fitness evaluations. Therefore, the search performance remains the same and the slight difference is mainly due to the nondeterministic nature of the algorithms. However, the CPU time differs significantly, where that of Cloudde is a fourth of that of Cloudde-w/o-VMs. As we use 16 VMs in the second layer for the four populations in the first layer, the expected result is that each population can use four VMs to calculate fitness when balanced. Indeed, Table 6 confirms this with a speedup factor of $374.250/92.395 = 4.05$. This number is larger than the theoretically possible four, due to the random nature of the runs, but it confirms that the load rebalancing is very effective and that communication overheads in the cloud layer of distribution are fully negligible especially when fitness evaluation is expensive like in this case. Therefore, the double-layered distribution framework makes Cloudde very suitable for cloud realizations for reduced computational time and costs, especially when objective evaluations are time-consuming to obtain, like in many real-world applications.

To study the effects of dynamic load balancing in details, we have also compared Cloudde with its variant short of resource balancing (denoted Cloudde-w/o-Balance) in Table 6. The CPU time is measured in second. The results show that the two algorithms obtain similar solution quality as anticipated, but Cloudde runs faster than Cloudde-w/o-Balance owing to the efficiency that the cloud resource balance strategy brings. Without it, the speedup factor is 3.16, less than that of 4.05.

Clearly, cloud resource balancing varies with the imbalance parameter $u_{\max}$. To test its influence under the average VM load of $l = 200/16 = 12.5$, we set $u_{\max}$ to 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, and 30, and plot their corresponding CPU times in Fig. 7. This curve shows a very interesting result. When $u_{\max}$ is smaller than four (the number of population islands), the CPU time increases with $u_{\max}$. This may be due to that with a larger $u_{\max}$, the Cloudde algorithm is in an unbalanced state longer, which increases the computational time. However, when $u_{\max}$ is larger than 4, the CPU time decreases as the $u_{\max}$ increases. This may be due to that with a larger $u_{\max}$, Cloudde does not implement the balance strategy so frequently, which can reduce the communicational overhead. However, when $u_{\max}$ keeps rising, the islands become more and more unbalanced. This means that the benefit brought about by the cloud resource balance strategy does override any communication overheads resulting from VM regrouping. When $u_{\max}$ reaches 20 (nearly double the average VM load), the balancing effect diminishes, where the CPU time is seen similar to that of Cloudde-w/o-Balance shown in Table 6.

In summary, setting $u_{\max}= 7$ (about half of the average VM load) offers a good choice in this PEC case study. The study demonstrates that practical speedup can be readily realized on cloud, offering potential for Cloudde to be deployed as a service on the internet. This would pave a way to encourage more widely innovative activities and more potential inventions of intellectual property (IP) blocks of nonlinear circuits as well as other practical IP blocks and systems.

TABLE 6
Comparisons Among Cloudde with $u_{max}= 7$, Cloudde-w/o-VMs, and Cloudde-w/o-Balance

Algorithms	Mean	Std. Dev.	Best	Worst	CPU Time	Speedup on cloud
Cloudde	175.137	16.849	185.051	161.223	92.395	4.05
Cloudde-w/o-VMs	175.280	15.960	186.051	165.223	374.250	-
Cloudde-w/o-Balance	175.056	13.960	185.210	167.243	118.515	3.16

6 Conclusion

In this paper, a novel, heterogeneous differential evolution algorithm, Cloudde, has been developed. It can be realized in one of the four forms: as a sequential DE, as a distributed DE, as a cloud DE, and as a double-layered distributed DE also suitable for cloud computing implementations. In the first layer of Cloudde, different DE populations with various parameters and/or operators run concurrently and independently. Different populations can accommodate various search requirements of different problems, and migration helps combine the advantages of mixed populations adaptively. In the second layer of Cloudde, a set of VMs run in parallel in calculating fitness values of corresponding DE populations, which reduces computational time.

Experimental results on benchmarks with various search requirements and a case study on a PEC with expensive design evaluations have shown that Cloudde offers generally improved performance and reduced computational time. The speedup is significant on expensive problems and offers good potential in a broad range of real-world applications.

Moreover, being important issues in distributed computing, analyses of the population migration and dynamic sizing have been conducted to ascertain the contributions of Cloudde. Experimental results have shown that the AMS offers an ability to control the population size dynamically, thereby guiding individuals migrate from poorer-performing populations to better-performing populations. This way, Cloudde does not waste computational resources on populations that offer no potential to solve the problem, and thus enhances the execution efficiency. As Cloudde migrates individuals from populations to populations, it uses a resource balancing strategy to regroup the VMs so as to maintain an overall efficiency. The effectiveness of this strategy and its influence factor have also been investigated. Experimental results show that migrating too frequently will lead to unnecessary communication burden, whilst too seldom will result in wasted CPU time on the unbalanced populations. Therefore, a trade-off should be made to the value of the imbalance parameter $u_{\max}$.

As VMs can be created easily in a cloud computing environment, Cloudde is scalable for more populations. This offers an advantage of our cloud-ready DE algorithm. Moreover, based on the heterogeneous Cloudde framework, we can also use any other enhanced DE variants in a population. This way, Cloudde may offer even higher performance in terms of speed, accuracy, and robustness.

Future work can also involve investigations on the migration rate p_m because it impacts on the algorithm performance. One solution could be to make it adaptive [62]. Moreover, the Cloudde algorithm will be further tested against more complex benchmark functions, multi-objective optimization problems [64], and in more real-world applications problems.

Acknowledgments

This work was partially supported by the National Natural Science Foundations of China (NSFC) with No. 61402545, the Natural Science Foundations of Guangdong Province for Distinguished Young Scholars with No. 2014A030306038, the Project for Pearl River New Star in Science and Technology with No. 201506010047, GDUPS (2016), and the NSFC Key Program with No. 61332002 and No. U1201258. Zhi-Hui Zhan and Jun Zhang are the Corresponding Authors.

References

[1] M. D. Dikaiakos, G. Pallis, D. Katsaros, P. Mehra, and A. Vakali, "Cloud computing: Distributed internet computing for IT and scientific research," *IEEE Internet Comput.*, vol. 13, no. 5, pp. 10–13, Sep. 2009.

[2] S. Bera, S. Misra, and J. Rodrigues, "Cloud computing applications for smart grid: A survey," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1477–1494, May 2015.

[3] Z. H. Zhan, X. F. Liu, Y. J. Gong, J. Zhang, H. S. H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Comput. Surveys*, vol. 47, no. 4, pp. 1–33, Jul. 2015, Art. no. 63.

[4] Z. Zhu, G. Zhang, M. Li, and X. Liu, "Evolutionary multi-objective workflow scheduling in cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1344–1357, May 2016.

[5] X. F. Liu, Z. H. Zhan, K. J. Du, and W. N. Chen, "Energy aware virtual machine placement scheduling in cloud computing based on ant colony optimization approach," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2014, pp. 41–47.

[6] H. F. Sheikh, I. Ahmad, and D. Fan, "An evolutionary technique for performance-energy-temperature optimized scheduling of parallel tasks on multi-core processors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 3, pp. 668–681, Mar. 2016.

[7] W. Guo, J. Li, G. Chen, Y. Niu, and C. Chen, "A PSO-optimized real-time fault-tolerant task allocation algorithm in wireless sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3236–3249, Dec. 2015.

[8] H. K. Hsin, E. J. Chang, and A. Y. Wu, "Spatial-temporal enhancement of ACO-based selection schemes for adaptive routing in network-on-chip systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 6, pp. 1626–1637, Jun. 2014.

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

[10] 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.

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

[12] E. Dilettoso, S. A. Rizzo, and N. Salerno, "A parallel version of the self-adaptive low-high evaluation evolutionary-algorithm for electromagnetic device optimization," *IEEE Trans. Magn.*, vol. 50, no. 2, pp. 1–4, Feb. 2014.

[13] R. Storn and K. Price, "Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[14] R. Storn, "On the usage of differential evolution for function optimization," in *Proc. North Amer. Fuzzy Inf. Process. Soc. Conf.*, 1996, pp. 519–523.

[15] E. Mezura-Montes, J. Velázquez-Reyes, and C. A. C. Coello, "A comparative study of differential evolution variants for global optimization," in *Proc. Genetic Evol. Comput. Conf.*, 2006, pp. 485–492.

[16] A. Salman, A. P. Engelbrecht, and M. G. H. Omran, "Empirical analysis of self-adaptive differential evolution," *Eur. J. Oper. Res.*, vol. 183, no. 2, pp. 785–804, 2007.

[17] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," *IEEE Trans. Evol. Comput.*, vol. 10, no. 6, pp. 646–657, Dec. 2006.

[18] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 398–417, Apr. 2009.

[19] J. Q. 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.

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

[21] Q. Liu, W. Wei, H. Yuan, Z. H. Zhan, and Y. Li, "Topology selection for particle swarm optimization," *Inf. Sci.*, vol. 363, no. 1, pp. 154–173, Oct. 2016.

[22] Y. H. Li, Z. H. Zhan, S. J. Lin, J. Zhang, and X. N. Luo, "Competitive and cooperative particle swarm optimization with information sharing mechanism for global optimization problems," *Inf. Sci.*, vol. 293, no. 1, pp. 370–382, Feb. 2015.

[23] W. J. Yuet al., "Differential evolution with two-level parameter adaptation," *IEEE Trans. Cybern.*, vol. 44, no. 7, pp. 1080–1099, Jul. 2014.

[24] Y. Wang, Z. X. Cai, and Q. F. Zhang, "Differential evolution with composite trial vector generation strategies and control parameters," *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 55–66, Feb. 2011.

[25] X. F. Liu, Z. H. Zhan, and J. Zhang, "Dichotomy guided based parameter adaptation for differential evolution," in *Proc. Genetic Evol. Comput. Conf.*, 2015, pp. 289–296.

[26] S. M. Islam, S. Das, S. Ghosh, S. Roy, and P. N. Suganthan, "An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization," *IEEE Trans. Syst. Man Cybern. B*, vol. 42, no. 2, pp. 482–500, Apr. 2012.

[27] S. M. Elsayed, R. A. Sarker, and D. L. Essam, "An improved self-adaptive differential evolution algorithm for optimization problems," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 89–99, Feb. 2013.

[28] B. Dorronsoro and P. Bouvry, "Improving classical and decentralized differential evolution with new mutation operator and population topologies," *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 67–98, Feb. 2011.

[29] M. G. Epitropakis, D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis, "Enhancing differential evolution utilizing proximity-based mutation operators," *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 99–119, Feb. 2011.

[30] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama, "Opposition-based differential evolution," *IEEE Trans. Evol. Comput.*, vol. 12, no. 1, pp. 64–79, Feb. 2008.

[31] N. Noman and H. Iba, "Accelerating differential evolution using an adaptive local search," *IEEE Trans. Evol. Comput.*, vol. 12, no. 1, pp. 107–125, Feb. 2008.

[32] U. K. Chakraborty, S. Das, and A. Konar, "Differential evolution with local neighborhood," in *Proc. IEEE Congr. Evol. Comput.*, 2006, pp. 2042–2049.

[33] Y. L. Li, Z. H. Zhan, Y. J. Gong, W. N. Chen, J. Zhang, and Y. Li, "Differential evolution with an evolution path: A DEEP evolutionary algorithm," *IEEE Trans. Cybern.*, vol. 45, no. 9, pp. 1798–1810, Sep. 2015.

[34] Z. H. Zhan and J. Zhang, "Enhance differential evolution with random walk," in *Proc. Genetic Evol. Comput. Conf.*, 2012, pp. 1513–1514.

[35] M. Vasile, E. Minisci, and M. Locatelli, "An inflationary differential evolution algorithm for space trajectory optimization," *IEEE Trans. Evol. Comput.*, vol. 15, no. 2, pp. 267–281, Apr. 2011.

[36] A. Basak, S. Das, and K. C. Tan, "Multimodal optimization using a biobjective differential evolution algorithm enhanced with mean distance-based selection," *IEEE Trans. Evol. Comput.*, vol. 17, no. 6, pp. 666–685, Dec. 2013.

[37] R. L. Becerra and C. A. C. Coello, "Cultured differential evolution for constrained optimization," *Comput. Methods Appl. Mechanics Eng.*, vol. 195, pp. 4303–4322, Jul. 2006.

[38] A. Biswas, S. Dasgupta, S. Das, and A. Abraham, "A synergy of differential evolution and bacterial foraging algorithm for global optimization," *Neural Netw. World*, vol. 17, no. 6, pp. 607–626, 2007.

[39] R. Storn, "System design by constraint adaptation and differential evolution," *IEEE Trans. Evol. Comput.*, vol. 3, no. 1, pp. 22–34, 1999.

[40] J. H. Zhong, M. Shen, J. Zhang, H. Chung, Y. H. Shi, and Y. Li, "A differential evolution algorithm with dual populations for solving periodic railway timetable scheduling problem," *IEEE Trans. Evol. Comput.*, vol. 17, no. 4, pp. 512–527, Aug. 2013.

[41] F. Neri and E. Mininno, "Memetic compact differential evolution for Cartesian robot control," *IEEE Comput. Intell. Mag.*, vol. 5, no. 2, pp. 54–65, May 2010.

[42] N. Noman and H. Iba, "Inferring gene regulatory networks using differential evolution with local search heuristics," *IEEE/ACM Trans. Comput. Biol. Bioinform.*, vol. 4, no. 4, pp. 634–647, Oct. 2007.

[43] B. V. Babu and R. Angira, "Modified differential evolution (MDE) for optimization of non-linear chemical processes," *Comput. Chem. Eng.*, vol. 30, no. 6/7, pp. 989–1002, 2006.

[44] G. Chen, A. Sarrafzadeh, and S. Pang, "Service provision control in federated service providing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 3, pp. 587–600, Mar. 2013.

[45] Y. L. Li, Z. H. Zhan, Y. J. Gong, J. Zhang, Y. Li, and Q. Li, "Fast micro-differential evolution for topological active net optimization," *IEEE Trans. Cybern.*, vol. 46, no. 6, pp. 1411–1423, Jun. 2016.

[46] Z. H. Zhan and J. Zhang, "Differential evolution for power electronic circuit optimization," in *Proc. Conf. Technol. Appl. Artificial Intell.*, 2015, pp. 158–163.

[47] D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis, "Parallel differential evolution," in *Proc. IEEE Congr. Evol. Comput.*, 2004, pp. 2023–2029.

[48] W. Kwedlo and K. Bandurski, "A parallel differential evolution algorithm," in *Proc. IEEE Int. Symp. Parallel Comput. Electr. Eng.*, 2006, pp. 319–324.

[49] K. N. Kozlov and A. M. Samsonov, "New migration scheme for parallel differential evolution," in *Proc. Int. Conf. Bioinfor. Genome Regulation Structure*, 2006, pp. 141–144.

[50] J. Apolloni, G. Leguizamón, J. García-Nieto, and E. Alba, "Island based distributed differential evolution: An experimental study on hybrid testbeds," in *Proc. IEEE Int. Conf. Hybrid Intell. Syst.*, 2008, pp. 696–701.

[51] M. Weber, F. Neri, and V. Tirronen, "A study on scale factor in distributed differential evolution," *Inf. Sci.*, vol. 181, pp. 2488–2311, 2011.

[52] M. Weber, F. Neri, and V. Tirronen, "A study on scale factor/crossover interaction in distributed differential evolution," *Artificial Intell. Rev.*, vol. 39, no. 3, pp. 195–224, 2013.

[53] M. Salomon, G. R. Perrin, F. Heitz, and J. P. Armspach, "Parallel differential evolution: Application to 3-D medical image registration," *Differential Evolution*. Berlin, Germany: Springer, 2005, 353–411.

[54] I. De Falco, A. Della Cioppa, D. Maisto, U. Scafuri, and E. Tarantino, "Satellite image registration by distributed differential evolution," *Applications of Evolutionary Computing*. Berlin, Germany: Springer, 2007, pp. 251–260.

[55] W. J. Yu and J. Zhang, "Multi-population differential evolution with adaptive parameter control for global optimization," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2011, pp. 1093–1098.

[56] R. Mallipeddi, P. N. Suganthan, Q. K. Pan, and M. F. Tasgetiren, "Differential evolution algorithm with ensemble of parameters and mutation strategies," *Appl. Soft Comput.*, vol. 11, no. 2, pp. 1679–1696, Mar. 2011.

[57] T. Ishimizu and K. Tagawa, "Experiment study of a structured differential evolution with mixed strategies," in *Proc. 2nd World Congr. Nature Biologically Inspired Comput.*, 2010, pp. 591–596.

[58] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel Comput.*, vol. 22, no. 6, pp. 789–828, Sep. 1996.

[59] Z. H. Zhan, J. Zhang, Y. Li, and Y. H. Shi, "Orthogonal learning particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 15, no. 6, pp. 832–847, Dec. 2011.

[60] Z. H. Zhan and J. Zhang, "Orthogonal learning particle swarm optimization for power electronic circuit optimization with free search range," in *Proc. IEEE Congr. Evol. Comput.*, 2011, pp. 2563–2570.

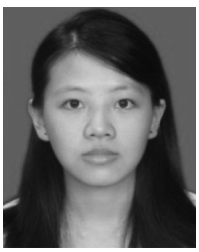
[61] X. F. Liu, Z. H. Zhan, J. H. Lin, and J. Zhang, "Parallel differential evolution on distributed computational resources for power electronic circuit optimization," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2016, pp. 117–118.

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

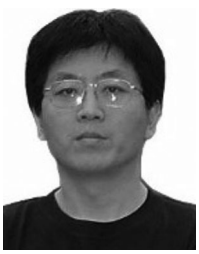
[63] A. Auger and N. Hansen, "Performance evaluation of an advanced local search evolutionary algorithm," in *Proc. IEEE Congr. Evol. Comput.*, 2005, pp. 1777–1784.

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

Zhi-Hui Zhan (S'09-M'13) received the Bachelor's and the PhD degree from the Department of Computer Science, Sun Yat-sen University, Guangzhou, China, in 2007 and 2013, respectively. He is currently a professor in the School of Computer Science and Engineering, South China University of Technology. He is also appointed as the Pearl River Scholar Young professor in 2016. His current research interests include evolutionary computation algorithms, swarm intelligence algorithms, and their applications in real-world problems, and in environments of cloud computing and big data. His doctoral dissertation was awarded the China Computer Federation Outstanding Dissertation in 2013. He received the Natural Science Foundation for Distinguished Young Scientists of Guangdong Province, China, in 2014 and was awarded the Pearl River New Star in Science and Technology in 2015. He is listed as one of the Most Cited Chinese Researchers in Computer Science. He is a member of the IEEE.



Xiao-Fang Liu (S'14) received the BS degree in computer science from Sun Yat-sen University, Guangzhou, China, in 2015, where she is currently working toward the PhD degree. She also currently works in Intelligent Computation and Cloud Computing Lab, South China University of Technology. Her current research interests include artificial intelligence, evolutionary computation, swarm intelligence, and their applications in design and optimization such as cloud computing resources scheduling. She is a student member of the IEEE.



Huaxiang Zhang received the PhD degree from Shanghai Jiaotong University, in 2004. He worked as an associated professor in the Department of Computer Science, Shandong Normal University from 2004 to 2005, where he is currently a professor in the School of Information Science and Engineering, Shandong Normal University, China. He has authored more than 100 journal and conference papers and has been granted eight invention patents. His current research interests include machine learning, pattern recognition, evolutionary computation, web information processing, etc.

Zhengtao Yu received the PhD degree in computer application technology from Beijing Institute of Technology, Beijing, China, in 2005. He is currently a professor in the School of Information Engineering and Automation, Kunming University of Science and Technology, China. His main research interests include natural language processing, information retrieval and machine learning.

Jian Weng received the MS and BS degrees in computer science and engineering from South China University of Technology, in 2004 and 2000, respectively, and the PhD degree in computer science and engineering from Shanghai Jiao Tong University, in 2008. From April 2008 to March 2010, he was a postdoc in the School of Information Systems, Singapore Management University. Currently, he is a professor and executive dean in the School of Information Technology, Jinan University. His current research interests include cryptography, information security, and computational intelligence. He has published more than 60 papers in cryptography conferences and journals, such as CRYPTO, EUROCRYPT, ASIACRYPT, the *Transactions on Cloud Computing*, the *IEEE Transactions on Dependable and Secure Computing*, the *IEEE Transactions on Information Forensics and Security*, etc. He served as PC co-chairs or PC member for more than 20 international conferences. He has won the 2014 cryptographic innovation award from Chinese Association for Cryptographic Research, the best paper award from the 28th Symposium on Cryptography and Information Security (SCIS 2011), and the best student award from the 8th International Conference on Provable Security (ProvSec 2014).

Yun Li (S'87-M'90) received the BS degree in electronics science, the MEng degree in electronic engineering, and the PhD degree in computing and control, in 1984, 1987, and 1990, respectively. During 1989-1990, he was with United Kingdom National Engineering Laboratory and Industrial Systems and Control Ltd, Glasgow. He joined the University of Glasgow as lecturer in 1991, where he is currently a professor of systems engineering. He served as founding director of the University of Glasgow Singapore during 2011-2013. He established the IEEE Control System Society and European Network of Excellence workgroups on computer-automated design (CAutoD) in 1998 and wrote the popular online courseware 'GA Demo' for interactive evolutionary computation in 1997. He is currently researching into smart design with market informatics via the cloud to complete the value chain for Industry 4.0. He is currently professor and visiting professor of the University of Glasgow, University of Electronic Science and Technology of China, and Dongguan University of Technology. He is a chartered engineer and an associate editor of the *IEEE Transactions on Evolutionary Computation* and of the *SM Journal of Engineering Sciences*. He has 200 publications. He is a member of the IEEE.

Tianlong Gu received the MEng degree from Xidian University, China, in 1987, and the PhD degree from Zhejiang University, China, in 1996. From 1998 to 2002, he was a research fellow in the School of Electrical and Computer Engineering, Curtin University of Technology, Australia, and a post-doctoral fellow in the School of Engineering, Murdoch University, Australia. He is currently a professor in the School of Computer Science and Engineering, Guilin University of Electronic Technology, China. His research interests include formal methods, data and knowledge engineering, software engineering, and information security protocol.

Jun Zhang (M'02–SM'08) received the PhD degree in electrical engineering from City University of Hong Kong, in 2002. He is currently a professor with South China University of Technology, China. He has authored seven research books and book chapters, and more than 50 IEEE Transactions papers in his research areas. His current research interests include computational intelligence, cloud computing, wireless sensor networks, operations research, and power electronic circuits. He was a recipient of the National Science Fund for Distinguished Young Scholars in 2011 and the First-Grade Award in Natural Science Research from the Ministry of Education, China, in 2009. He was also appointed as the Changjiang chair professor in 2013. 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. He is a senior member of the IEEE.