Distributed Cooperative Co-Evolution With Adaptive Computing Resource Allocation for Large Scale Optimization

Ya-Hui Jia, *Student Member, IEEE*, Wei-Neng Chen, *Senior Member, IEEE*, Tianlong Gu, Huaxiang Zhang, Hua-Qiang Yuan, Sam Kwong, *Fellow, IEEE*, and Jun Zhang, *Fellow, IEEE*

***Abstract*—Through introducing the divide-and-conquer strategy, cooperative co-evolution (CC) has been successfully employed by many evolutionary algorithms (EAs) to solve large-scale optimization problems. In practice, it is common that different subcomponents of a large-scale problem have imbalanced contributions to the global fitness. Thus, how to utilize such imbalance and concentrate efforts on optimizing important subcomponents becomes an important issue for improving performance of cooperative co-EA, especially in distributed computing environment. In this paper, we propose a two-layer distributed CC (dCC) architecture with adaptive computing resource allocation for large-scale optimization. The first layer is the dCC model which takes charge of calculating the importance of subcomponents and accordingly allocating resources. An effective allocating algorithm is designed which can adaptively allocate computing resources based on a periodic contribution calculating method. The second layer is the pool model which takes charge of making fully utilization of imbalanced resource allocation. Within this layer, two different conformance policies are designed to help optimizers use the assigned computing resources efficiently. Empirical studies show that the two conformance policies and the computing resource allocation algorithm are effective, and the proposed distributed architecture possesses high scalability and efficiency.**

Manuscript received April 9, 2017; revised September 8, 2017 and December 12, 2017; accepted March 8, 2018. Date of publication March 21, 2018; date of current version March 29, 2019. This work was supported in part by the National Natural Science Foundation of China under Grant 61622206, and Grant 61332002, in part by the Science of Technology Planning Project of Guangdong Province, China under Grant 2014B010118002, and in part by the Natural Science Foundation of Guangdong under Grant 2015A030306024. *(Corresponding authors: Wei-Neng Chen; Jun Zhang.)*

Y.-H. Jia is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, and also with the School of Data and Computer Science, Sun Yat-sen University, 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; junzhang@ieee.org).

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.-Q. 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/TEVC.2018.2817889

***Index Terms*—Cooperative co-evolution (CC), distributed evolutionary algorithm (EA), large-scale optimization, pool model, resource allocation.**

I. Introduction

EVOLUTIONARY computation (EC) has shown to be an effective technique for many difficult optimization problems [1], [2]. However, as the size and complexity of the problems grow rapidly, the performance of most existing evolutionary algorithms (EAs) tends to deteriorate quickly [3], [4]. Large-scale global optimization (LSGO) with high-dimensional decision variables and time-consuming objective functions has been increasingly considered as one of the most challenging issues in EC research.

To improve performance of EC algorithms for LSGO, numerous methods were proposed [5]–[11]. One typical approach is the cooperative co-evolution (CC) method that decomposes the problem into several low-dimensional subcomponents and handles them by several cooperative optimizers [12]. The first CC evolutionary algorithm (CCEA) was proposed by Potter and Jong [13], named CCGA. Afterward, researchers took advantage of the methodology of CC architecture and proposed a large number of CCEAs [14]–[26]. Since the interdependence between variables would greatly influence the performance of CCEAs especially for LSGO problems, so far most researches about CC concentrate on the scheme how to partition high-dimensional variables into groups. Roughly, they can be classified into three categories: 1) static grouping method which partitions the variables into a fixed number of static groups [4], [13], [14]; 2) random dynamic grouping method which changes the grouping structure dynamically during the optimization process [15]–[17]; and 3) learning-based dynamic grouping method which partitions the variables through interdependence detecting techniques [18]–[21].

The above works increase the accuracy of grouping by proposing new decomposition techniques. But most of them still use the traditional CC architecture which is an even round-robin strategy that treats all subcomponents equally. Few works considered the imbalance in the contribution of subcomponents to the objective value. In many real-world applications, different decision variables possess different priorities, having unequal contributions to the objective value. Therefore,

1089-778X © 2018 IEEE. Translations and content mining are permitted for academic research only. Personal use is also permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

it is inappropriate to treat all subcomponents equally in CC architecture. Seeing this drawback of traditional CC, Omidvar *et al.* [27] proposed a contribution-based CC (CBCC) framework. In CBCC, the subcomponent which has the largest contribution to the global objective value will get extra times of fitness evaluation to evolve. But the accumulative calculation of contributions makes it response to the local change of objective value too slowly even incorrectly [28]. To make up this shortcoming, they successively proposed three new versions, named CBCC1 [29], CBCC2 [29], and CBCC3 [30]. Yang *et al.* [28] also proposed a new CC framework, called CCFR, which could detect whether a subcomponent was stagnant. Then it would gradually discard stagnant subcomponents during the optimization process.

Though the above studies have further improved the performance of CC, most of them were considered in the serial computing environment. Due to the methodology of divide-and-conquer, CC possesses intrinsic parallelism which makes it easy to be implemented in distributed environment. Meanwhile, the great scalability of distributed CC (dCC) endows itself great ability to solve problems with much higher dimensions and higher computational burden [31]. This is especially meaningful in the age of big data. As point out by Cevher *et al.* [32], there are three key pillars of the optimization for big data: first-order methods, randomization, and parallel and distributed computation. Developing dCC in the EC field exactly corresponds to the latter two of them. Moreover, Sabar *et al.* [33] utilized CC algorithms for big data optimization problems recently and achieved very promising results, which verified the effectiveness of CC for big data optimization. Cao *et al.* [34] proposed a dCC algorithm for multiobjective large-scale optimization problems which also had proven to be efficient and effective. Thus, it is both necessary and meaningful to study the parallelization of CC in distributed environments.

In this paper, we intend to study how to utilize the imbalance in the contribution of subcomponents in distributed computing environment, and propose a double-layer dCC with adaptive computing resource allocation (DCCA) to solve LSGO problems. First, a two-layer hierarchical model is built by realizing two levels of distribution, i.e., distribution of dimension where subcomponents of the problem are assigned to different processors, and distribution of population where subpopulations are assigned to different processors. Different from other hybrid dEAs which only consider the same kind of distribution model [35], [36], DCCA is designed with two different kinds of models. In the first layer, dCC divides a high-dimensional problem into low-dimensional subcomponents, which is a dimension-distributed model. In the second layer, pool model [37] divides a population of EAs into subpopulations, which is a population-distributed model. In this way, DCCA on one hand obtains the dCC's scalability of dimension division to deal with high-dimensional problems, and on the other hand obtains the flexibility of pool model so that subcomponents can adapt to the changing of resource allocation quickly and utilize the assigned computing resource efficiently. Moreover, in the first layer, i.e., dCC, to measure the importance of subcomponents and reallocate computing resources,

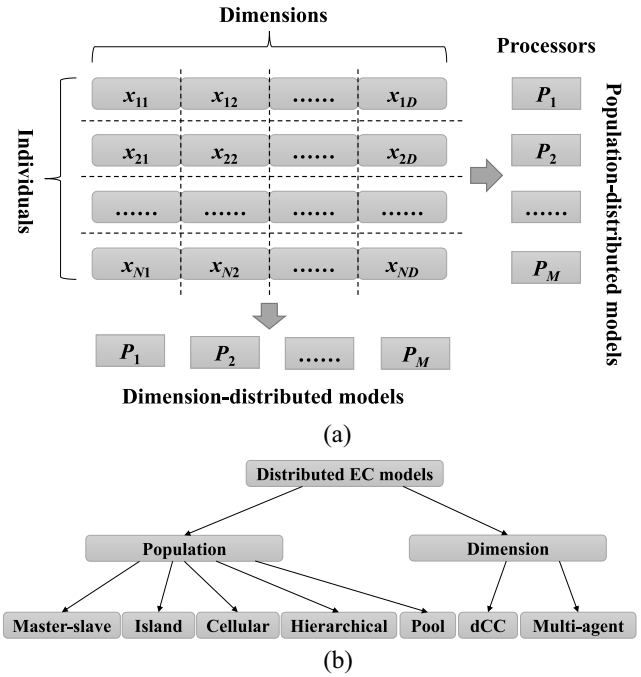

Fig. 1. Population-distributed and dimension-distributed models. (a) Task division manners. (b) Taxonomy.

an adaptive resource allocation scheme is designed which can dynamically sense the changing of the contributions of different subcomponents toward the global fitness. Accordingly it will reallocate computing resources periodically and adaptively in favor of the subcomponents with high priorities. In the second layer, i.e., pool model, two conformance policies, population policy and generation policy, are designed for adapting the applied EA to the reallocated computing resources. The population policy adjusts the optimizer's population size, and the generation policy adjusts the number of generations.

To prove the ability of DCCA in allocating and utilizing distributed computing resources, the CEC'2013 large scale benchmark functions [38] and 18 2000-D functions with imbalanced contributions are tested in distributed computing environment. Experimental results show that DCCA can significantly improve the efficiency of dCC and accelerate the optimization speed of CCEAs.

The rest of this paper is organized as follows. In Section II, two dEA models used in DCCA, i.e., CC model and pool model, are described. Then, Section III explains the proposed DCCA from two layers in detail. Experiments are conducted to verify the effectiveness of DCCA in Section IV. Finally, the conclusions are drawn in Section V.

II. Background

Basically, the state-of-the-art models of dEAs can be divided into two classes according to the method used to split a task into subtasks. As shown in Fig. 1(a), population-distributed models distribute individuals or subpopulations to multiple processors, while dimension-distributed models distribute the partitions of problem dimensions to processors.

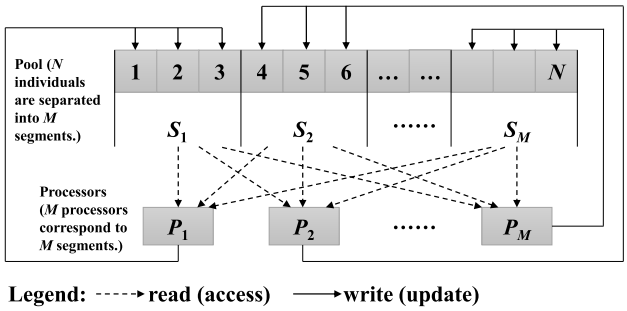

Fig. 2. Pool model.

Taxonomy of these distributed EC models are shown in Fig. 1(b). To our best knowledge, there are five frequently used population-distributed models: 1) master–slave model [39], [40]; 2) island model (also known as coarse-grained model) [41], [42]; 3) cellular model (also known as fine-grained model) [43], [44]; 4) hierarchical model (also known as hybrid model) [35], [36]; and 5) pool model [37], [45]. There are also two popular dimension-distributed models: 1) the CC model [46] and 2) the multiagent model [47].

Here, the pool model and the CC model which are related to DCCA will be shown in detail. Further explanation about other models can be found in [3].

A. Pool Model

As a population-distributed model, the pool model maintains the population by setting a shared pool. All processors can access the whole resource pool which contains the population. However, each processor can only update a segment of the pool assigned to it. Fig. 2 shows how a pool model works. In the example, the pool is designed as an array with N elements representing N individuals, and the array is partitioned into M segments corresponding to M processors. Each processor can read any individual from any segment, but is only allowed to write individuals back to its own segment.

The advantage of the pool model is its high scalability and flexibility. As all processors work on a same resource pool, it is easy to add new processors by increasing the size of the pool, or repartitioning the segments, and vice versa. Due to the same reason, EAs based on pool model can realize both synchronously and asynchronously easily. These features would facilitate the dynamic allocation of computing resources in DCCA.

B. Cooperative Co-Evolution

For any CCEA using the decomposition methods which generate fixed subcomponents, the whole process to solve an optimization problem consists of two steps: 1) decomposition and 2) optimization [28], which are shown in Fig. 3(a). In the first place, it decomposes a problem into several smaller subcomponents. Then each subcomponent is associated with a separate optimizer with its own population. These two steps are relatively independent which means one can choose any suitable decomposition method and EA according to the real

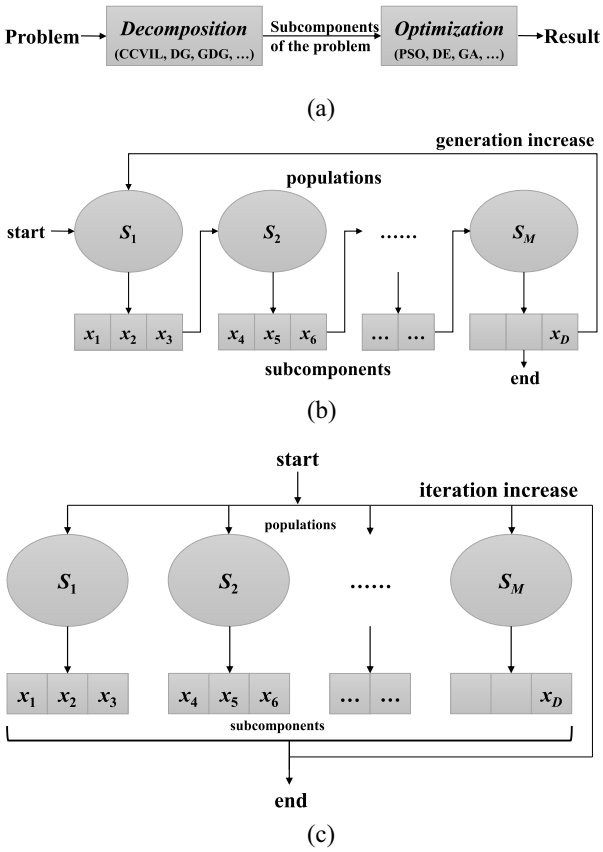

Fig. 3. Structure of CC. (a) Whole process of a CC approach which uses fixed grouping method. (b) Optimization step of (b) serial CC and (c) dCC.

application. The decomposition step is executed in the same way in both serial CC and dCC. In this paper, we focus on the optimization step, utilizing the imbalance among subcomponents to improve the performance. Meanwhile, what differs dCC from serial CC is also the optimization step.

For serial CC, whenever a population is evolving, the others are held fixed. Thus, it is a serial architecture because anytime there is only one population evolving. Graphical representation of the serial CC is shown in Fig. 3(b). In the example, a D-dimensional problem $C = \{x_1, x_2, \ldots, x_D\}$ is divided into M subcomponents

$$\begin{aligned} &C = C_1 \cup C_2 \cup \ldots \cup C_M \\ &\text{where} \\ &\text{for } \forall i \in [1, M], C_i \neq \emptyset \\ &\text{for } \forall i, j \in [1, M] \wedge i \neq j, C_i \cap C_j = \emptyset. \end{aligned} \tag{1}$$

Usually the fitness of an individual in a population is calculated by combining with the best individuals of other populations. In the jth generation, denoting the best individual in the ith population S_i as best_i^j, the fitness of one individual s^i in S_i is calculated as

$$\begin{aligned} &F(s^i) = f\left(s^i, \overline{C_i^j}\right) \\ &\text{where} \\ &\overline{C_i^j} = \left(\text{best}_1^j, \ldots, \text{best}_{i-1}^j, \text{best}_{i+1}^{j-1}, \ldots, \text{best}_M^{j-1}\right). \end{aligned} \tag{2}$$

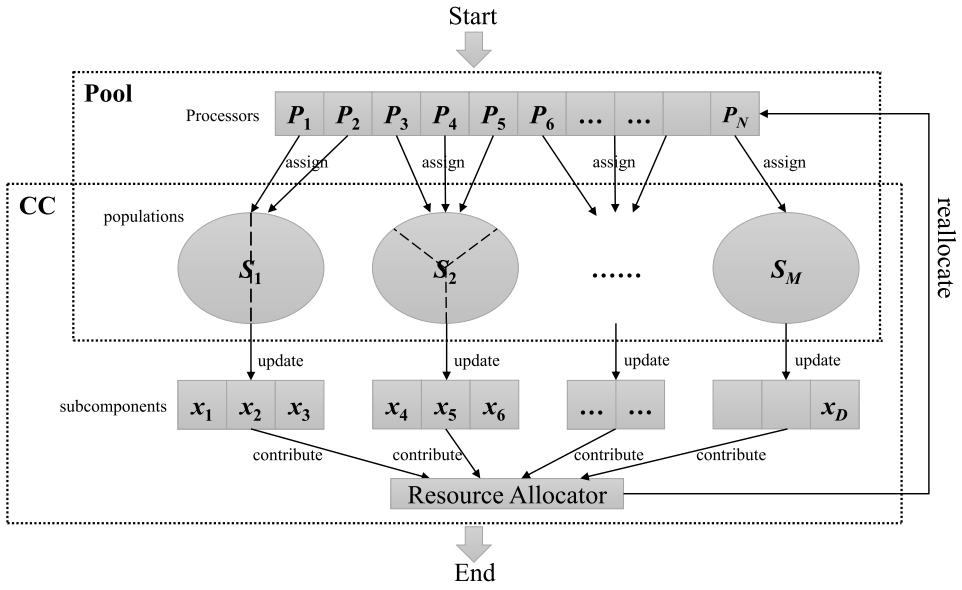

Fig. 4. Structure of DCCA.

Since s^i has only the dimensions of C_i, it is not a complete solution. Its fitness can only be calculated by assembling with $\overline{C_i^j}$. In each generation, each population evolves and updates its own best_i^j meanwhile updates the corresponding subcomponent of the global best solution $g$best

$$g\text{best} = \left(\text{best}_1^j, \ldots, \text{best}_{i-1}^j, \text{best}_i^j, \text{best}_{i+1}^{j-1}, \ldots, \text{best}_M^{j-1}\right). \tag{3}$$

In this way, every next population can immediately utilize the improvements achieved by all previously evolved populations.

Taking advantage of the fundamental divide-and-conquer methodology of CC, researchers naturally promoted serial CC into large-scale problems and developed its distributed version [20], [31], [34], [46], which is shown in Fig. 3(c). In dCC, to make full use of the computing resources, subcomponents are deployed onto different processors and evolve simultaneously which is different from the serial structure. Formula description will be shown in following sections. Generally, to reduce the cost of the communication between processors, the synchronization among subcomponents is not performed every generation, but only at intervals of some generations. It means that in dCC, $g$best is usually updated after several generations of evolution. Thus, in Fig. 3(c), the word "iteration" means several "generations." In dCC, every population can only utilize the improvements achieved by other populations in the previous iteration. It is also worth noting that since dCC itself does not have the ability to coordinate multiple processors to handle one subcomponent, usually a subcomponent is assigned to only one processor in canonical dCC [31]. Even if multiple processors are assigned to one subcomponent [34], there is not a mechanism to adjust the resource allocation.

Studying the imbalance in the contribution of subcomponents in distributed computing environment can be divided into two mutually related problems. The first one is how to partition a problem into subcomponents and assign processors according to their imbalanced contributions. The second one is how a subcomponent adapts to the changing of assigned processors and fully utilizes them. With respect to these two questions, dCC exactly has the scalability to divide a high-dimensional problem into subcomponents, but it cannot coordinate multiple unevenly allocated processors to handle the subcomponents. On the contrary, pool model is gifted in coordinating multiple processors to solve a task efficiently, but it does not have the ability to partition a large-scale problem. Combining these two models together and introducing a new resource allocating scheme, DCCA can partition the problem, analyze the imbalanced contribution and accordingly reallocate processors in its first layer, i.e., dCC, meantime utilize the assigned processors efficiently in its second layer, i.e., pool model.

III. Distributed Cooperative Co-Evolution With Adaptive Computing Resource Allocation

DCCA is demonstrated in a top-down way in this section. First, the overall structure and implementation of DCCA is shown. Then, key components, such as the adaptive resource allocation scheme, conformance policy, are explained.

A. Structure

DCCA is composed of two different kinds of dEA models, which are the dCC model and the pool model. Given a large-scale problem, DCCA first divides it into subcomponents from the perspective of dimension. Then for each subcomponent, a pool model is applied to divide the corresponding population into subpopulations. The structure of DCCA is shown in Fig. 4. Since the decomposition procedure has no difference in serial CC and dCC, it is omitted from the figure, and only the optimization step is shown. From Fig. 4, we can see that the problem has already been partitioned into M subcomponents. Without loss of generality, any decomposition method which generates fixed groups can

be applied in DCCA. Considering the grouping accuracy, the global differential grouping method [21] is recommended.

From the perspective of population-distribution, each subcomponent C_i is related to a population S_i. Each population and the processors assigned to it form a pool model together in DCCA

$$\begin{aligned} \text{Pool}_i &= \left(S_i, P^i\right) \\ P^i &= \left\{P_j \middle| P_j \text{ is assigned to } S_i\right\}. \end{aligned} \tag{4}$$

Based on the framework of the pool model, population S_i will be divided into $|P^i|$ subpopulations, and each subpopulation corresponds to one processor. (Here, "subpopulation" is conceptually the same as "segment" for the pool model.) In the example shown in Fig. 4, there are totally N processors $\{P_1, P_2, \ldots, P_N\}$. Two processors $\{P_1, P_2\}$ are assigned to the first population S_1, thus S_1 will be divided into two subpopulations. Three processors $\{P_3, P_4, P_5\}$ are assigned to the second population S_2, thus S_2 will be divided into three subpopulations.

As introduced in Section II-A, generally we synchronize the populations at intervals of a certain number of generations. During synchronization, the contribution of each subcomponent to the global objective is collected by the resource allocator as well. Based on the contributions, the resource allocator reallocates the computing resources through an allocation algorithm which will be shown in the following sections.

B. Implementation

In accordance with the structure, the implementation of DCCA is also composed of two layers. In the first layer, we realize the dCC model through multiprocess techniques, where a host-process is set to be the resource allocator, meanwhile each subcomponent is handled by a subprocess corresponding to a population. In the second layer, the pool model is realized by multithread techniques, where a host-thread is set to maintain the shared pool, i.e., the population, and subthreads are created to handle the subpopulations. This is a feasible way to realize DCCA, but without loss of generality, DCCA can be also implemented by other distributed or parallel techniques. In order to prevent any potential confusion, the words "process" and "thread" will be still adopted in the following description of DCCA, respectively, corresponding to the dCC model and the pool model. In addition, based on the distributed computing framework composed of CC model and pool model, any EA can be employed in DCCA as the optimizer.

Fig. 5 shows the flowchart of DCCA. It consists of two diagrams. The left diagram represents the host-process; the right one represents a subprocess; the dotted arrow between two diagrams represents the interaction between the host-process and subprocesses. At first, a problem is decomposed into subcomponents within the host-process and equal numbers of subprocesses are created for subcomponents. Before evolving, each subcomponent C_i initializes its own populations S_i. Since the priorities of the subcomponents are still undetermined at that point, processors are allocated evenly to all subcomponents for the first time. Then, *MaxItr* iterations of evolution will be executed. Within each iteration, subprocesses

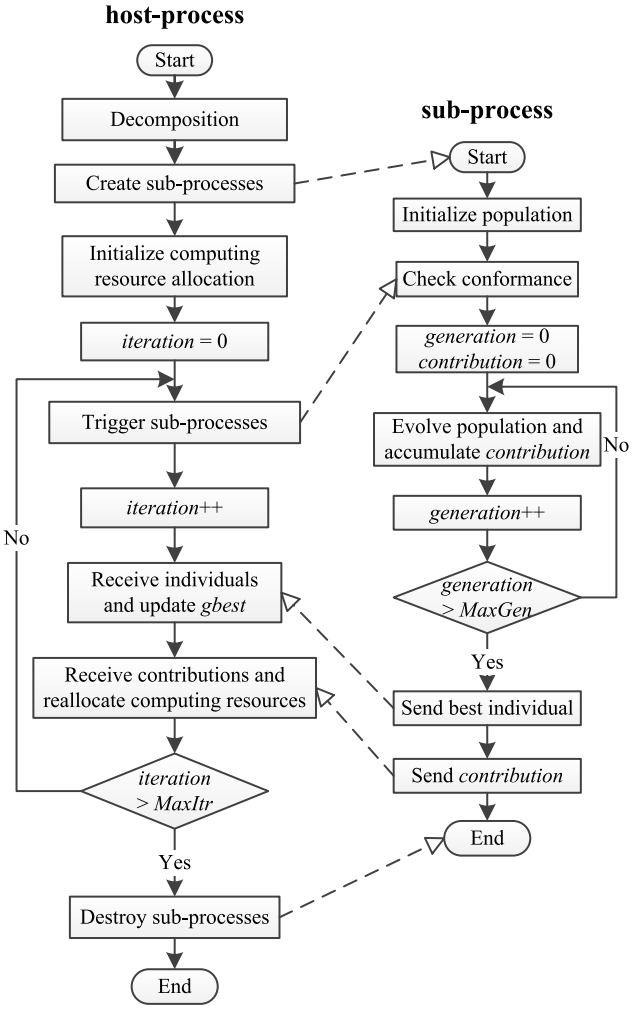

Fig. 5. Flowchart of DCCA.

are first activated by the host-process to make a conformance test to check whether the amount of the assigned processors has changed. Afterward, each population evolves for *MaxGen* generations. In the jth iteration ($j \geq 1$), the best individual of population S_i, denoted as best_i^j, is collected

$$\begin{aligned} \text{best}_i^j &= \arg\min_{s^i \in S_i} f\left(s^i, \overline{C_i^{j-1}}\right) \\ \overline{C_i^{j-1}} &= \left(\text{best}_1^{j-1}, \ldots, \text{best}_{i-1}^{j-1}, \text{best}_{i+1}^{j-1}, \ldots, \text{best}_M^{j-1}\right) \end{aligned} \tag{5}$$

where s^i represents one individual in S_i, $\overline{C_i^{j-1}}$ represents the global best solution $g\text{best}^{j-1}$ without the dimensions of C_i. Also the contribution of each population to the global objective accumulates. After *MaxGen* generations, subprocesses send their best individuals best_i^j and contributions to the host-process, then suspend themselves. Host-process uses the received best individuals to update the global best solution $g\text{best}^j$

$$\begin{aligned} g\text{best}^j &= \left(\text{best}_1^j, \text{best}_2^j, \ldots, \text{best}_M^j\right) \\ &= \left(\arg\min_{s^1 \in S_1} f\left(s^1, \overline{C_1^{j-1}}\right), \ldots, \arg\min_{s^M \in S_M} f\left(s^M, \overline{C_M^{j-1}}\right)\right). \end{aligned} \tag{6}$$

Contributions are utilized to reallocate the processors according to the allocation algorithm. If the stop criterion is not met, the host-process will activate the subprocesses once again. Otherwise, it will finally kill all subprocesses, end the present optimization process, and return the global best solution *g*best.

To implement DCCA on an existing CCEA, there are also two corresponding steps: 1) decomposition mapping and 2) optimization mapping. Decomposition in serial CC is the same as in dCC, thus the decomposition method used in the CCEA can be directly mapped onto the decomposition procedure in the host-process of DCCA. As to the optimization mapping, generally EAs have two basic parts: 1) initialization and 2) iterative evolution. These two parts can be mapped onto the "initialize population" and "evolve population" procedures in the subprocess, respectively. But the applied EA should be implemented in the form of the pool model.

C. Adaptive Computing Resource Allocation

Whether in distributed environment or not, the basic thought to utilize the imbalance among subcomponents is to assign more processors to the subcomponents which contribute more to the global objective. The resource allocator of DCCA takes charge of gathering contributions of all subcomponents in each iteration, and accordingly reallocating processors. Its fundamental idea is to take limited computing resource from the stagnant subcomponents which contribute less, and give them to the subcomponents which contribute more. The contribution of each subcomponent to the global objective is calculated in its corresponding subprocess. It is initialized to 0 and accumulates during each generation. Denoting the best individual of the ith population in the kth generation ($k \geq 1$) of the jth iteration as $\text{best}_i^{j,k}$, the contribution of the ith subcomponent is calculated by

$$\begin{aligned} \Delta c &= f\left(\text{best}_i^{j,k-1}, \overline{C_i^{j-1}}\right) - f\left(\text{best}_i^{j,k}, \overline{C_i^{j-1}}\right) \\ \text{contribution}_i &= \text{contribution}_i + \Delta c. \end{aligned} \tag{7}$$

Then, the resource allocator adjusts the allocation in a moderate way. Each time, two subcomponents are selected, and the subcomponent with lower priority will donate a processor to the other one. Priority is calculated in the host-process which is defined as

$$\text{priority}_i = \frac{\text{contribution}_i}{\left|P^i\right|} \tag{8}$$

representing the contribution brought by one processor. Moreover, to avoid extreme imbalance of resource allocation, a lower bound and an upper bound are set to limit the minimum and the maximum number of processors a subcomponent may own, respectively, denoted as *rlb* and *rub*. The pseudo code of the resource allocation is shown in Algorithm 1.

Algorithm 1 Adaptive Computing Resource Allocation

Input: number of subcomponents M, contributions *contribution*, every pool $\{Pool_1, Pool_2, \ldots, Pool_M\}$, lower and upper bound *rlb* and *rub*.

```
priority = 0_{1×M};
for i = 1 to M do
    calculate priority_i according to (8);
end for
sort {Pool_1, Pool_2, ..., Pool_M} based on priority in ascending order;
receiver = −1, donor = −1;
for i = M to 1 do
    if |P^i| < rub then
        receiver = i; break;
    end if
end for
for i = 1 to receiver − 1 do
    if |P^i| > rlb then
        donor = i; break;
    end if
end for
if donor != −1 then
    Pool^donor denotes a processor to Pool^receiver;
end if
```

D. Conformance Policy

"Conformance" in DCCA refers to the matching between the strategy of evolution of a population and the number of processors it owns, and "unconformity" only appears when the number of processors assigned to a subcomponent changes, which is defined as follows.

Definition 1: Given a pool $Pool_i$, denoting the processors it owns in the jth iteration as $P^{i,j}$, unconformity exists in $Pool_i$ if and only if $P^{i,j} \neq P^{i,j+1}$ and the conformance policy of $Pool_i$ in the $j+1$th iteration has not been executed.

Once a subprocess senses the unconformity, it applies a conformance policy immediately to correct it in order to take full advantage of the assigned computing resources. For most EAs, the performance and the convergence behavior can be influenced by two factors: 1) the population size and 2) the number of generations. Based on these two factors, two conformance policies are designed: 1) named population policy and 2) generation policy. Meanwhile the DCCA model which applies the population model is denoted as DCCA-P; the other is denoted as DCCA-G.

1) Population Policy: For most EAs based on swarm-intelligence, increasing population size within a reasonable range can improve the performance, and vice versa. Thus, in DCCA-P, whenever a subcomponent gets more processors, the size of its corresponding population will increase. On the contrary, whenever a subcomponent loses processors, its population size will decrease. In the implementation of DCCA-P, each processor is associated with a certain number of individuals. Denoting the number as *inp*, if the ith subcomponent owns $|P^i|$ processors, its population size $|S_i|$ is determined as

$$|S_i| = inp \times \left|P^i\right|. \tag{9}$$

After the host-process scatters the allocation information to subprocesses, each subprocess first checks whether the number of assigned processors has changed. If the number increases, the subprocess will create *inp* more individuals for its population. If the number decreases, the subprocess will remove *inp* individuals from its population. To generate *inp* new individuals, the current minimum value and maximum value of each dimension of the population are calculated at first. Then

new individuals are generated randomly within these ranges. To remove *inp* individuals, the population is sorted according to the fitness at first. Then, based on the elitism, DCCA-P will remove the worst *inp* individuals from the population.

Since every processor is associated with equal amount of individuals and all individuals evolve for equal amount of generations in an iteration, theoretically no waste of computation power will be caused by the imbalance of fitness calculation amount in the population policy. Thus, the processors will be fully utilized.

2) Generation Policy: Some EAs are not sensitive to the population size, and may even get worse when the population grows larger [11]. The generation policy is developed for these EAs to adjust generation number rather than population size to conform to the change of the quantity of processors. In brief, populations with more processors can evolve more generations. In DCCA-G, each processor is associated with a certain number of generations. Denoting the number as *gnp*, if the *i*th subcomponent owns $|P^i|$ processors, its population can evolve

$$\text{MaxGen}_i = gnp \times \left|P^i\right| \tag{10}$$

generations within one iteration. Still, individuals in the *i*th population are divided into $|P^i|$ parts corresponding to $|P^i|$ processors. In this way, the workload of each processor is also balanced.

3) Theoretical Comparison Between Two Policies: Compared with the population policy, the generation policy is a more direct way to exhibit and utilize the imbalance in the contribution of subcomponents. In DCCA-P, by increasing the population size of the subcomponents with higher priorities, more search diversity can be brought, but all populations are still evolved by the same number of generations. In contrast, DCCA-G directly changes the number of generations. As a consequence, the evolutionary level of the populations with higher priorities will be higher than those with lower priorities. Take a simple form of function $f(x, y)=ax^2+by^2$ where x and y are two subcomponents as an example. Consider a situation where a is equal to b and due to the unbalanced sampling of the applied EA, $x = 0.1$ and $y = 10$ at the end of one iteration. If not trapped in a local optima, generally y will have a longer step length than x in the following iteration, thus it will contribute more. In this situation, the generation policy will give y more generations to evolve so that it can catch up with x. Thus, the whole convergence speed will be accelerated. On the contrary, if the population policy is applied, it will increase the size of the population of y, which usually will not accelerate the convergence speed of y too much. Consider another situation where a is much smaller than b, and x has the same level of evolution with y at the end of one iteration. Clearly y is more important than x now. Under the circumstance, the population policy will give y more individuals to increase its diversity, protecting the important subcomponent from premature. But the generation policy will unilaterally accelerate the convergence speed of y, as a consequence y may be easy to be trapped. To draw a conclusion, theoretically speaking, the generation policy is more capable of accelerating the convergence speed of the optimization, and the population policy tends to protect the optimization from premature.

IV. Experiment

Different from the previous works which focus on proposing new algorithms to solve the LSGO problems in serial computing environment, DCCA is a distributed evolutionary computing model, thus all of the following experiments are conducted in distributed computing environment. In the following experiments, the CEC'2013 LSGO problems are chosen as the basic benchmark [38]. It consists of 15 1000-D functions with three different types: 1) totally separable; 2) partially separable; and 3) overlapping. Compared with the CEC'2010 benchmark, the imbalance in the contribution of subcomponents is introduced in CEC'2013 LSGO problems. Meanwhile, more overlapping functions are considered. These characters can help us check the ability of DCCA in handling the imbalanced case and allocating computing resources. Moreover, as the imbalance in CEC'2013 benchmark functions is only made by setting nonuniform subcomponent sizes which is too simple to simulate practical applications, some higher-dimensional functions with more complex imbalanced situations are designed based on the CEC'2013 benchmark.

In DCCA, the self-adaptive differential evolution with neighborhood search algorithm (SaNSDE) [48] is applied as the optimizer. This algorithm has been applied in several CC-based algorithms and achieved good performance [20], [28]. As DCCA is the first dCC model which considers the imbalanced contribution, there is not a similar model we can compare with. Therefore, to examine the effectiveness of DCCA, a dCC architecture without adaptive resource allocation is designed as the control group, named SCC ("S" stands for "static resource allocation"). Totally three approaches: 1) DCCA-P; 2) DCCA-G; and 3) SCC are tested on the benchmark functions. Every approach is executed on the functions for independent 30 times to get the mean value and the median value of the objective. Moreover, a Wilcoxon rank sum test is made to show whether DCCA brings significant improvement to SCC. Max iteration number *MaxItr* is set to 250. Each subcomponent is assigned with two processors initially for all functions. For DCCA-P, *inp* is set to 30 which means each processor is associated with 30 individuals. For DCCA-G, *gnp* is set to 10 which means each processor is associated with ten generations. Thus, for DCCA-P and SCC, each iteration contains 20 generations, namely *MaxGen* = 20; for DCCA-G and SCC, each population has 60 individuals, namely $|S|=60$. *rlb* is set to 1 which means that each subcomponent at least can own one processor. For fair comparison, parameters of SaNSDE are set as in [48]. The CPU model is Intel Core i7-4790 with 3.60 GHz. Multiprocess technique is realized through MPI. Mult*i*thread technique is realized through Pthreads.

First, we directly test DCCA on the CEC'2013 benchmark, checking its performance and selecting a relatively better *rub* setting. Then, according to the results, 18 2000-D functions are designed to further test the DCCA's ability in handling the cases with varied imbalanced contributions. Afterward, the situation of processor allocation and the corresponding

TABLE I
RESULTS ON CEC'2013 BENCHMARK FUNCTIONS

Function	Quality	SCC	DCCA-G				DCCA-P			
	rub		4	6	8	10	4	6	8	10
f_1	Mean	2.94E-18	4.83E-22	4.65E-22	9.06E-22	1.04E-21	6.08E-21	5.69E-21	6.11E-21	7.06E-21
	Median	1.29E-21	3.11E-22	3.04E-22	1.80E-22	3.78E-22	1.69E-21	2.48E-21	2.16E-21	2.80E-21
	Std.	1.08E-17	4.87E-22	4.48E-22	1.90E-21	2.11E-21	1.19E-20	9.44E-21	9.61E-21	9.94E-21
	p-value		8.15E-05*	1.11E-04*	4.08E-05*	2.16E-03*	6.20E-01	3.48E-01	3.04E-01	4.04E-01
f_2	Mean	3.06E+01	3.14E+01	4.25E+01	2.91E+01	2.83E+01	1.55E+02	1.69E+02	1.68E+02	1.60E+02
	Median	2.59E+01	2.59E+01	3.73E+01	2.94E+01	2.74E+01	1.53E+02	1.66E+02	1.73E+02	1.69E+02
	Std.	1.63E+01	1.55E+01	2.05E+01	1.89E+01	1.54E+01	3.03E+01	4.08E+01	3.98E+01	3.89E+01
	p-value		7.79E-01	2.86E-02#	5.01E-01	5.74E-01	3.00E-11#	3.01E-11#	3.00E-11#	3.00E-11#
f_3	Mean	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.00E+01
	Median	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.00E+01
	Std.	7.34E-05	1.35E-04	1.62E-04	1.33E-04	1.30E-04	9.78E-05	8.84E-05	8.14E-05	7.48E-05
	p-value		1.73E-11#	1.83E-11#	1.73E-11#	1.69E-11#	8.26E-09#	7.24E-10#	6.93E-10#	1.58E-10#
f_4	Mean	4.19E+09	5.83E+09	2.67E+09	3.27E+09	2.68E+09	1.41E+09	1.57E+10	2.09E+10	3.61E+10
	Median	4.15E+09	4.33E+09	1.94E+09	2.20E+09	1.33E+09	1.22E+09	5.90E+08	3.62E+08	1.20E+09
	Std.	1.58E+09	5.11E+09	2.54E+09	3.41E+09	3.20E+09	9.81E+08	3.27E+10	4.00E+10	4.84E+10
	p-value		6.95E-01	3.99E-04*	3.03E-03*	1.68E-04*	1.70E-08*	7.66E-05*	1.95E-03*	6.84E-01
f_5	Mean	3.42E+06	3.28E+06	2.85E+06	2.97E+06	2.95E+06	3.02E+06	2.92E+06	3.26E+06	3.15E+06
	Median	3.44E+06	3.35E+06	2.84E+06	2.90E+06	3.02E+06	2.96E+06	2.87E+06	3.14E+06	3.12E+06
	Std.	5.04E+05	4.60E+05	4.25E+05	4.70E+05	3.63E+05	3.97E+05	4.39E+05	4.97E+05	3.30E+05
	p-value		2.17E-01	8.66E-05*	1.44E-03*	6.91E-04*	3.34E-03*	3.99E-04*	2.06E-01	4.84E-02*
f_6	Mean	1.05E+06	1.06E+06	1.06E+06	1.06E+06	1.06E+06	1.05E+06	1.05E+06	1.05E+06	1.05E+06
	Median	1.05E+06	1.06E+06	1.06E+06	1.06E+06	1.06E+06	1.05E+06	1.05E+06	1.05E+06	1.05E+06
	Std.	1.75E+03	8.66E+00	1.27E+03	1.41E+03	1.59E+03	1.76E+03	1.92E+03	2.30E+03	1.80E+03
	p-value		1.14E-04#	1.25E-07#	1.36E-07#	4.80E-07#	1.59E-02#	7.78E-03#	1.01E-01	4.51E-02#
f_7	Mean	1.10E+08	1.30E+08	7.71E+07	6.62E+07	5.87E+07	1.12E+08	1.36E+08	9.65E+07	9.95E+07
	Median	7.79E+07	6.57E+07	5.14E+07	4.66E+07	3.97E+07	1.02E+08	1.21E+08	8.61E+07	8.94E+07
	Std.	1.14E+08	1.78E+08	9.91E+07	6.69E+07	9.18E+07	4.67E+07	6.85E+07	4.30E+07	4.53E+07
	p-value		8.07E-01	1.05E-01	7.48E-02	7.62E-03*	2.32E-02#	1.03E-02#	2.40E-01	1.58E-01
f_8	Mean	2.04E+14	1.55E+14	1.29E+14	1.45E+14	1.24E+14	5.22E+13	2.60E+14	9.90E+14	1.52E+15
	Median	1.80E+14	1.33E+14	9.10E+13	1.15E+14	7.41E+13	3.81E+13	2.87E+13	3.03E+13	6.08E+14
	Std.	1.12E+14	1.14E+14	1.22E+14	1.71E+14	1.29E+14	3.76E+13	7.95E+14	1.51E+15	1.70E+15
	p-value		3.78E-02*	1.17E-03*	1.52E-03*	6.55E-04*	7.12E-09*	1.73E-07*	6.15E-02	8.24E-02
f_9	Mean	6.05E+08	8.06E+08	4.95E+08	5.94E+08	5.76E+08	3.36E+08	2.88E+08	3.85E+08	3.54E+08
	Median	4.91E+08	5.49E+08	4.52E+08	5.15E+08	4.87E+08	2.91E+08	2.86E+08	3.33E+08	3.54E+08
	Std.	3.63E+08	9.45E+08	1.61E+08	2.68E+08	2.56E+08	1.21E+08	5.78E+07	2.02E+08	9.83E+07
	p-value		3.04E-01	2.28E-01	7.17E-01	9.35E-01	8.20E-07*	2.67E-09*	6.77E-05*	5.46E-06*
f_{10}	Mean	9.28E+07	9.31E+07	9.29E+07	9.30E+07	9.29E+07	9.29E+07	9.29E+07	9.29E+07	9.29E+07
	Median	9.28E+07	9.31E+07	9.30E+07	9.30E+07	9.30E+07	9.28E+07	9.28E+07	9.29E+07	9.29E+07
	Std.	4.05E+05	3.15E+05	3.40E+05	2.74E+05	3.48E+05	2.98E+05	3.42E+05	3.12E+05	3.36E+05
	p-value		1.11E-03#	1.05E-01	1.70E-02#	9.63E-02	4.51E-01	3.79E-01	3.40E-01	3.48E-01
f_{11}	Mean	9.39E+11	9.32E+11	9.26E+11	9.27E+11	9.28E+11	9.28E+11	9.23E+11	9.24E+11	9.24E+11
	Median	9.34E+11	9.28E+11	9.24E+11	9.25E+11	9.23E+11	9.25E+11	9.22E+11	9.23E+11	9.22E+11
	Std.	1.17E+10	8.90E+09	7.56E+09	7.12E+09	8.34E+09	7.60E+09	4.57E+09	4.40E+09	8.73E+09
	-value		4.43E-03*	3.83E-06*	1.09E-05*	2.68E-04*	1.34E-05*	7.77E-09*	2.60E-08*	1.39E-06*
f_{12}	Mean	1.69E+03	1.69E+03	1.64E+03	1.70E+03	1.59E+03	1.61E+03	1.62E+03	1.65E+03	1.65E+03
	Median	1.66E+03	1.66E+03	1.64E+03	1.68E+03	1.58E+03	1.61E+03	1.59E+03	1.68E+03	1.67E+03
	Std.	1.55E+02	1.42E+02	1.17E+02	1.38E+02	1.36E+02	1.39E+02	1.36E+02	1.71E+02	1.30E+02
	p-value		9.23E-01	3.04E-01	6.84E-01	2.92E-02*	5.01E-02	1.05E-01	4.73E-01	4.83E-01
f_{13}	Mean	7.92E+09	7.45E+09	6.70E+09	7.39E+09	7.83E+09	6.64E+09	5.79E+09	5.31E+09	4.84E+09
	Median	7.84E+09	7.00E+09	6.58E+09	7.33E+09	7.35E+09	6.33E+09	5.34E+09	5.19E+09	4.77E+09
	Std.	2.20E+09	1.75E+09	1.29E+09	1.68E+09	3.92E+09	2.90E+09	2.54E+09	1.83E+09	1.64E+09
	p-value		2.71E-01	1.12E-02*	4.04E-01	2.97E-01	2.15E-02*	2.39E-04*	9.51E-06*	1.03E-06*
f_{14}	Mean	8.97E+10	8.84E+10	7.65E+10	7.95E+10	9.44E+10	5.34E+10	4.11E+10	3.84E+10	3.41E+10
	Median	7.93E+10	7.62E+10	7.23E+10	7.69E+10	8.02E+10	5.05E+10	3.95E+10	3.96E+10	3.16E+10
	Std.	4.29E+10	4.93E+10	3.33E+10	3.21E+10	6.03E+10	2.03E+10	1.45E+10	1.70E+10	1.85E+10
	p-value		9.00E-01	2.97E-01	4.92E-01	9.00E-01	4.08E-05*	4.69E-08*	1.56E-08*	3.08E-08*
f_{15}	Mean	3.32E+11	3.21E+11	2.33E+11	1.78E+11	1.80E+11	7.49E+10	9.96E+10	8.33E+10	1.31E+11
	Median	2.26E+11	1.22E+11	1.16E+11	7.61E+10	6.86E+10	5.96E+10	5.51E+10	5.63E+10	7.99E+10
	Std.	2.97E+11	5.65E+11	3.72E+11	4.05E+11	2.56E+11	5.06E+10	1.57E+11	1.45E+11	2.26E+11
	p-value		3.03E-02*	9.88E-03*	4.08E-05*	1.68E-03*	2.03E-07*	3.26E-07*	4.69E-08*	2.00E-05*

* The results of DCCA-G or DCCA-P are significantly better than SCC according to a Wilcoxon rank sum test at level 0.05;
\# The results of DCCA-G or DCCA-P are significantly worse than SCC according to a Wilcoxon rank sum test at level 0.05.

convergence behavior are analyzed in detail. Finally, the scalability of DCCA is discussed by analyzing the time consumption.

Moreover, due to the page limit, more experiments are provided in the supplemental material, such as the discussion about the adaptability and practicability of DCCA, the

TABLE II
OPTIMIZING RANGE

Function	Original Value	SCC Median	Magnitude Difference
f_1	4.98766E+11	1.29000E-21	32
f_2	1.46509E+05	2.58689E+01	4
f_3	2.17183E+01	2.00018E+01	0
f_4	2.59868E+14	4.15009E+09	5
f_5	1.37712E+08	3.44294E+06	2
f_6	1.08159E+06	1.05370E+06	0
f_7	4.14636E+17	7.78605E+07	10
f_8	1.37531E+19	1.80396E+14	5
f_9	8.94863E+19	4.91232E+08	11
f_{10}	9.83454E+07	9.27950E+07	0
f_{11}	6.98404E+17	9.34327E+11	6
f_{12}	9.63743E+12	1.66224E+03	9
f_{13}	5.98283E+21	7.84371E+09	12
f_{14}	2.18481E+23	7.92557E+10	13
f_{15}	7.62043E+18	2.26245E+11	7

TABLE III
2000-D FUNCTIONS

F	f_x+f_y	Type of F	Range of f_x	Range of f_y	Range relation
F_1	f_1+f_2	S + S	+11\-22	+5\+1	inclusive
F_2	f_4+f_5	P + P	+14\+8	+8\+6	disjoint
F_3	f_4+f_7	P + P	+14\+8	+17\+7	inclusive
F_4	f_4+f_9	P + P	+14\+8	+19\+8	inclusive
F_5	f_8+f_9	P + P	+19\+13	+19\+8	inclusive
F_6	f_8+f_{11}	P + P	+19\+13	+17\+11	intersecting
F_7	f_9+f_{11}	P + P	+19\+8	+17\+11	inclusive
F_8	$f_{12}+f_{13}$	O + O	+12\+3	+21\+9	intersecting
F_9	$f_{13}+f_{14}$	O + O	+21\+9	+23\+10	intersecting
F_{10}	$f_{14}+f_{15}$	O + O	+23\+10	+18\+10	inclusive
F_{11}	f_1+f_7	S + P	+11\-22	+17\+7	intersecting
F_{12}	f_1+f_9	S + P	+11\-22	+19\+8	intersecting
F_{13}	f_1+f_{15}	S + O	+11\-22	+18\+10	intersecting
F_{14}	f_2+f_{13}	S + O	+5\+1	+21\+9	disjoint
F_{15}	f_5+f_{15}	P + O	+8\+6	+18\+10	disjoint
F_{16}	f_7+f_{13}	P + O	+17\+7	+21\+9	intersecting
F_{17}	f_9+f_{15}	P + O	+19\+8	+18\+10	inclusive
F_{18}	$f_{11}+f_{13}$	P + O	+17\+11	+21\+9	inclusive

S=Totally Separable, P=Partially Separable, O=Overlapping.

analysis of time consumption of DCCA-G and DCCA-P, the comparison between DCCA and some other models and algorithms.

A. Performance on CEC'2013 Benchmark

Since different problems have different degrees of imbalance, four values of *rub* are tested for each function: 4, 6, 8, and 10. Results are shown in Table I. Minimum median value and mean value of each test are highlighted.

1) Performance Analysis: From Table I, we can see that DCCA-G performs significantly better than SCC on nine functions according to the p-value of the Wilcoxon rank sum test. Meanwhile, DCCA-P performs significantly better than SCC on eight functions. Most of these functions are partially separable or overlapping, which implies that DCCA-P and DCCA-G are useful in handling problems with imbalanced contributions. However, there are still four functions where SCC gets better objective values. In order to find the situation in which DCCA can success, we randomly generate ten solutions and calculating the mean of their function values, which is shown in Table II. Also, the median values of SCC in Table I are shown in Table II to make a clear sight how much the objective value is optimized by the algorithm.

Checking the magnitude difference, we can find that, except for f_2, DCCA successes on most functions that have been optimized more or less. Its poor performance only appears on the three barely optimized functions: f_3, f_6, and f_{10}. This phenomenon tells us that as long as the optimization process can move on, DCCA is able to facilitate solving the problem and accelerate the speed of optimization. Because the contributions of subcomponents can only be calculated along with the progress of the optimization. If the CC methodology and applied EA originally cannot handle the problem, just like the three barely optimized functions, allocating processors becomes meaningless.

2) Selection of Rub: Ignoring f_3, f_6, and f_{10}, the other results show that the best *rub* setting depends on not only the problem but also the conformance policy. For DCCA-G, taking the median value as measurement, setting *rub* to 6 and 10 seems better than 4 and 8. For DCCA-P, setting *rub* to 6 favors 6 functions. Other settings each favors 2 functions. Although there is truly not a comprehensively best setting of *rub*, in most cases, setting *rub* to 6 can lead to relatively better result. Thus, in the following experiments, *rub* is moderately set to 6.

B. 2000-Dimensional Functions With Dynamic Imbalance

Although the imbalance in the contribution of subcomponents is considered in the CEC'2013 benchmark, it has two unpractical shortcomings [38]. First, the imbalance is realized by introducing nonuniform subcomponents which means the contribution of one subcomponent is tied to its size. Second, every function is only made by rotating or shifting only one base function, such as the sphere function, the elliptic function, etc. Once the coefficients are determined, the proportion of contributions to the global objective among all subcomponents is actually fixed during the whole optimizing process, so that the importance of subcomponents will be kept static during the optimizing process. This is a coarse simulation of real imbalanced cases.

In practical applications, not only the importance of different components is different, but also the importance of a single subcomponent is usually fluctuant and dynamic during the optimizing process. In order to introduce such dynamism, 18 2000-D functions are made by adding two different functions of the CEC'2013 benchmark, which are shown in Table III. Taking F_1 for instance, it is composed of two separable functions f_1 and f_2. The optimizing range of f_1 is (XE-22, YE+11) which contains the range of f_2 (XE+1, YE+5). Theoretically, at the early stage of optimization, subcomponents of f_1 are more important than subcomponents of f_2 because at that time, the function value of f_1 is much larger than the value of f_2. However, at the later stage of the optimization, the subcomponents of f_2 would become more important since the function value of f_1 is easy to be optimized lower than $E+1$ but it is much harder to optimize the function value of f_2 to the same level.

TABLE IV
RESULTS ON 2000-D FUNCTIONS

Function	Quality	SCC	DCCA-G	DCCA-P	Function	Quality	SCC	DCCA-G	DCCA-P
F_1	Mean	3.15E+01	2.84E+01	2.14E+02	F_{10}	Mean	6.79E+11	3.15E+11	3.23E+11
	Median	2.98E+01	2.09E+01	2.11E+02		Median	3.31E+11	2.07E+11	1.87E+11
	Std.	1.54E+01	1.76E+01	3.93E+01		Std.	1.21E+12	2.92E+11	3.54E+11
	p-value		2.97E-01	3.01E-11#		p-value		4.64E-03*	3.67E-03*
F_2	Mean	5.64E+09	3.27E+09	2.11E+10	F_{11}	Mean	9.46E+07	1.12E+08	9.38E+07
	Median	5.37E+09	2.46E+09	4.76E+08		Median	7.92E+07	5.92E+07	8.80E+07
	Std.	3.15E+09	2.73E+09	4.20E+10		Std.	9.95E+07	1.58E+08	3.36E+07
	p-value		2.89E-03*	4.22E-04*		p-value		4.29E-01	1.58E-01
F_3	Mean	5.95E+09	4.60E+09	3.73E+10	F_{12}	Mean	5.08E+08	5.98E+08	3.00E+08
	Median	4.44E+09	2.20E+09	1.98E+09		Median	4.92E+08	5.02E+08	2.74E+08
	Std.	4.07E+09	8.66E+09	5.06E+10		Std.	1.39E+08	3.02E+08	1.01E+08
	p-value		9.03E-04*	7.51E-01		p-value		4.83E-01	1.25E-07*
F_4	Mean	7.70E+09	3.75E+09	3.16E+10	F_{13}	Mean	5.63E+11	2.32E+11	2.19E+11
	Median	6.75E+09	2.73E+09	2.92E+09		Median	2.16E+11	3.69E+10	1.54E+11
	Std.	5.18E+09	2.91E+09	4.08E+10		Std.	1.11E+12	6.29E+11	2.53E+11
	p-value		1.41E-04*	4.73E-01		p-value		6.55E-04*	7.98E-02
F_5	Mean	2.15E+14	9.58E+13	2.77E+14	F_{14}	Mean	7.67E+09	7.44E+09	6.03E+09
	Median	2.05E+14	8.43E+13	2.33E+13		Median	7.68E+09	6.92E+09	5.33E+09
	Std.	1.13E+14	5.62E+13	9.45E+14		Std.	1.76E+09	1.88E+09	5.07E+09
	p-value		2.15E-06*	2.02E-08*		p-value		3.95E-01	2.39E-04*
F_6	Mean	2.26E+14	1.36E+14	3.90E+14	F_{15}	Mean	4.13E+11	4.99E+11	4.48E+11
	Median	1.74E+14	9.10E+13	1.95E+13		Median	2.66E+11	6.67E+10	2.11E+11
	Std.	1.87E+14	1.34E+14	1.08E+15		Std.	3.97E+11	1.47E+12	7.28E+11
	p-value		3.03E-03*	1.49E-06*		p-value		1.11E-03*	1.33E-01
F_7	Mean	9.34E+11	9.30E+11	9.26E+11	F_{16}	Mean	8.06E+09	7.23E+09	5.14E+09
	Median	9.33E+11	9.28E+11	9.24E+11		Median	8.37E+09	7.06E+09	4.44E+09
	Std.	6.25E+09	8.45E+09	7.01E+09		Std.	2.00E+09	1.62E+09	2.61E+09
	p-value		1.17E-02*	2.15E-06*		p-value		9.63E-02	1.39E-06*
F_8	Mean	7.95E+09	7.22E+09	6.66E+09	F_{17}	Mean	3.89E+11	1.52E+11	3.54E+11
	Median	7.78E+09	6.87E+09	5.39E+09		Median	1.83E+11	4.91E+10	1.84E+11
	Std.	2.15E+09	1.70E+09	4.99E+09		Std.	8.40E+11	2.90E+11	5.97E+11
	p-value		1.91E-01	1.06E-03*		p-value		4.08E-05*	4.83E-01
F_9	Mean	9.12E+10	9.27E+10	4.63E+10	F_{18}	Mean	9.41E+11	9.36E+11	9.33E+11
	Median	8.72E+10	8.84E+10	4.63E+10		Median	9.39E+11	9.32E+11	9.31E+11
	Std.	3.59E+10	2.45E+10	1.71E+10		Std.	8.46E+09	9.05E+09	7.65E+09
	p-value		5.49E-01	1.36E-07*		p-value		9.47E-03*	2.25E-04*

* The results of DCCA-G or DCCA-P are significantly better than SCC according to a Wilcoxon rank sum test at level 0.05;
\# The results of DCCA-G or DCCA-P are significantly worse than SCC according to a Wilcoxon rank sum test at level 0.05.

Thus, by combining two elementary functions with inclusive or intersecting range relation, not only the ties between the subcomponent size and its importance is unlocked, but also the dynamic importance can be realized. Following, the DCCA's ability of handling the dynamic importance is tested.

C. Performance on 2000-D Benchmark Functions

Still DCCA-G, DCCA-G, and SCC are tested on the 2000-D benchmark. Other experimental settings are kept unchanged, except that the resource upper bound *rub* is set fixed to 6 for two DCCA approaches. Numeric results are shown in Table IV.

Mean values show that the two DCCA approaches perform better on 17 functions out of these 18 functions. The situation reflected by the median values is even better where two DCCA approaches get better results on all 18 functions. According to the results of Wilcoxon rank sum test, DCCA-G performs significantly better than SCC on 11 functions. DCCA-P also performs significantly better than SCC on 11 functions. However, on F_1, DCCA-P is worse than SCC, owing to the inability in optimizing the elementary function f_2.

All values of these measurements exhibit the capability of DCCA in dealing with the imbalanced case with dynamic importance. Meanwhile, as on most functions DCCA performs well, we can also make the conclusion that DCCA is universally applicable for any function type and range relation.

D. Processor Allocation

In order to examine whether DCCA can sense the dynamic changing of each subcomponent's importance, the data about how many processor an elementary function possesses during each iteration is collected. The processor allocating ratio between the number of processors of elementary function f_x and f_y is calculated as:

$$\text{ProcessorRatio} = \sum_{i=1}^{M_x} pn_i / \sum_{j=1}^{M_y} pn_j \tag{11}$$

where M_x and M_y are the subcomponent amounts of f_x and f_y, respectively. Incorporating with the convergence behavior of each approach, we can clearly see how processors shift with the changing of the global objective value. Fig. 6 shows the results. Figures with odd number index display the results of

processor allocation, such as Fig. 6(a_1) and (b_1), etc. Figures with even number index display the results of convergence behavior of the approaches, such as Fig. 6(a_2) and (b_2), etc.

In the first place, we classify the 18 functions according to their range relations. For the three functions with disjoint range relation, F_2, F_{14}, and F_{15}, theoretically computing resources should shift to the elementary function whose value of optimization range is larger. The fact shown in Fig. 6(b_1), (n_1), and 6(o_1) exactly fulfills our expectation.

For the eight functions with inclusive range relation: F_1, F_3–F_5, F_7, F_{10}, F_{17}, and F_{18}, there is not a comprehensively suitable situation about how the processors should be allocated, since there is overlapping optimization range between two elementary functions and the optimizing speed on the two functions is also different which leads to constantly changing contributions. On F_1, mass-migration of computing resources has occurred three times shown in Fig. 6(a_1). Similar situation also happens on F_3 and F_4, but the migration seems much smaller than on F_1. Although on F_5, Fig. 6(e_1) shows a different situation where most processors are assigned to its first elementary function f_8 from beginning to end with only little vibration on the allocation curve, we can still see that every vibration is coming with a knee point in Fig. 6(e_2), which tells us both DCCA-P and DCCA-G have responded to the changing of the optimizing speed of each subcomponent. For F_7, F_{10}, and F_{17}, similar scenes are shown in their allocation figures where the vast majority of processors are assigned to the elementary function whose lower limit of the optimization range is larger.

For the rest seven functions with intersecting range relation: F_6, F_8, F_9, F_{11}–F_{13}, and F_{16}, basically there are two scenarios about the processor allocation. The first one is similar to the situation of the disjoint range relation, where most computing resources will be allocated to the elementary function with larger value of optimization range, such as F_8, F_{13}, and F_{16}. The reason is that although the optimization ranges of two elementary functions are intersecting, there are still great differences between the two ranges. In the second situation, there is not a big gap between two elementary functions' optimization range, thus processors will shift to and fro between two elementary functions, and finally be biased to the function whose lower limit of optimization range is larger, such as F_6, F_9, F_{11}, and F_{12}.

Through observing the figures of resource allocation and the figures of convergence behavior, we can find that the scheme of the adaptive computing resource allocation of DCCA is useful.

E. Guideline of Selection Between DCCA-G and DCCA-P

Throughout all figures of three approaches' convergence behavior, we can find that DCCA-G and DCCA-P improve the performance of dCC in two different ways. By giving more generations to evolve the populations with higher priorities, DCCA-G gains a fast convergence speed at the beginning of the optimization process. However, such strategy makes the algorithm converges too fast that the optimization process is easy to be trapped afterward. Contrary to DCCA-G, by increasing the size of the populations with higher priorities, the convergence speed of DCCA-P is actually decreased in the beginning. Thus, we can find that in many cases, such as Fig. 6(b_2) and (c_2), DCCA-P does a very poor job in the early stage of the optimization. But through elevating the level of diversity of important populations, DCCA-P successfully protects the algorithm from premature convergence, and it keeps a steady and valid convergence speed to optimize the problem and finally surpasses the other two approaches. Thus, the interests of two DCCA conformance policies are different. DCCA-G focuses on improving the exploitation ability of the algorithm, while DCCA-P focuses on improving the exploration ability of the algorithm.

To select a suitable policy for a real-world application or a specific CCEA, a general guideline is made by considering three conditions.

1) *Time of the Optimization:* Short or long. If the problem needs to be solved in a very short period, DCCA-G is much preferred, because generally DCCA-G can accelerate the optimization speed. If enough time is prepared for the optimization, we recommend DCCA-P because maintaining a steady convergence speed and avoiding premature will be more beneficial.
2) *Complexity of the Problem:* Complex or simple. If the problem to be solved is very complex which contains many local optima, we think DCCA-P may be better because many researches have shown that the exploration ability of the algorithm is more important than the exploitation ability under the circumstance [11], [49]. If the problem is relatively simple, such as the unimodal function f_1, DCCA-G is preferred.
3) *Character of the EA:* Exploration or exploitation. We can see from the experiments that the two policies are both effective no matter the optimizer is SaNSDE or Cauchy and Gaussian particle swarm optimization [17] (shown in supplemental material). Thus, if one can select a policy according to the former two conditions, this condition can be taken as an auxiliary condition. If this condition must be taken primarily, we recommend to first check whether the EA can benefit from DCCA-P. For some EAs like CMA-ES, where their population sizes are determined by the dimensionality, and for some EAs like CSO, where increasing their population sizes may get negative effects, DCCA-P cannot be used. For other EAs, a general idea provided is to make up the deficiency. If the EA is good at exploitation, DCCA-P is preferred. Otherwise DCCA-G is preferred.

F. Scalability

The scalability of DCCA should be embodied in two aspects. The first aspect is the ability to handle the growing size of target problems. The other is the efficiency to accelerate the optimization process with more processors. For the first aspect, since DCCA is proposed based on the CC architecture, as long as the target problem can be handled by CCEAs, the scalability of DCCA in dealing with the growth of the problem size can be ensured. The above experimental results also have

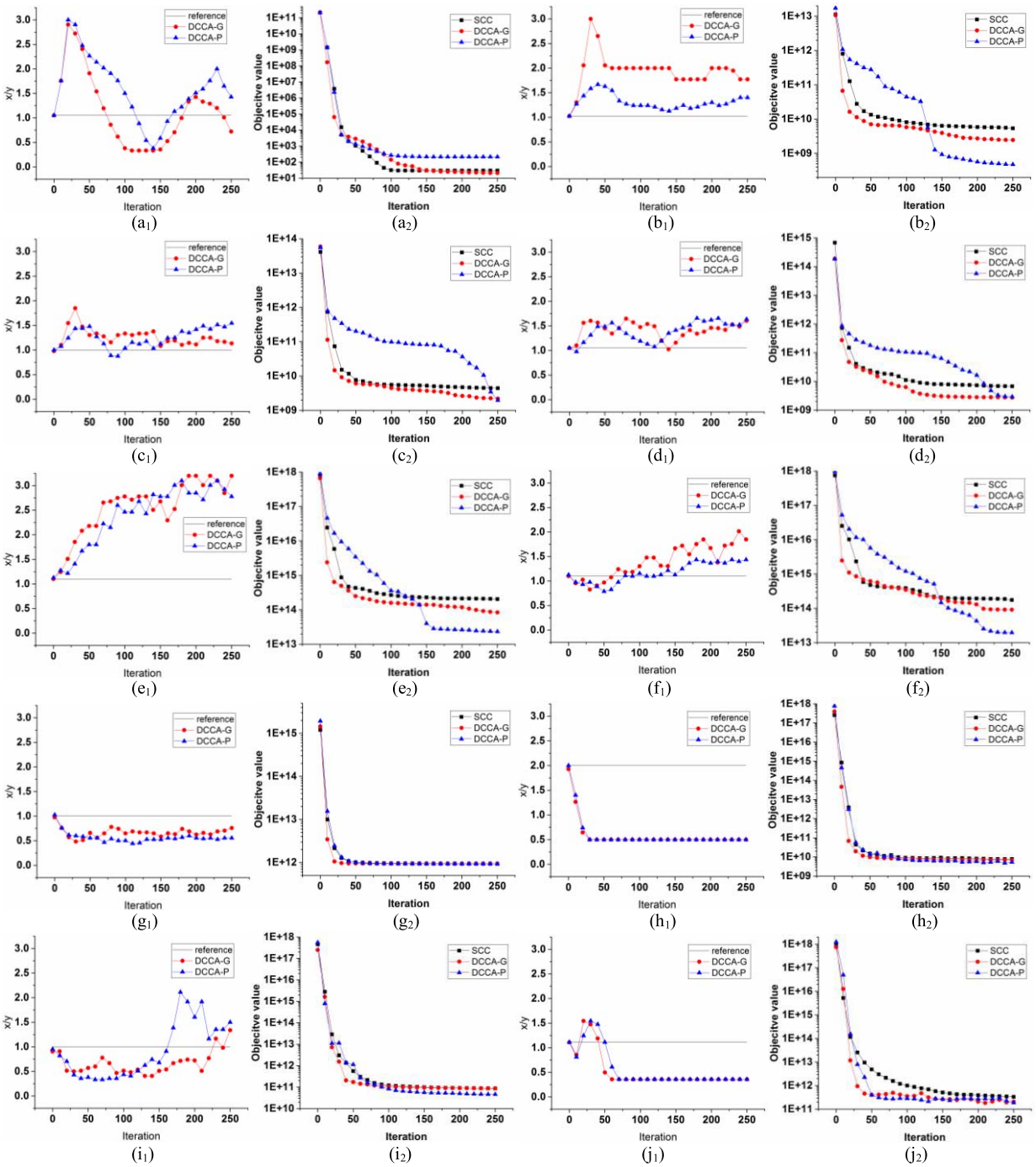

Fig. 6. Continued.

demonstrated this point. Thus, we only discuss the scalability of DCCA in using computing resource here.

DCCA-G is applied to represent DCCA and the conventional dCC [31] is set as the control group. In the first place, we give each function of CEC'2013 LSGO benchmark a prepared value according to the results got in Section IV-A. Since the functions in CEC'2013 benchmark are relatively simple that their execution time is too short to analyze the runtime efficiency of DCCA for solving real world problems in distributed computing environments, we lift up their execution time to the range (2E-3, 3E-3) seconds. Then dCC is tested on each function to check how much time it needs to optimize the fitness value of the target function to the prepared value. (Essentially, the conventional dCC can be seen as a special case of DCCA where *rub* and *rlb* are both set to 1.) Afterward, we gradually increase the processor number and run DCCA on each function to see how much time it needs to reach the prepared value. Each case is tested 30 times independently. The median values of the execution time are show in Fig. S1 in the supplemental material.

Checking the results generally, except the three functions which DCCA cannot handle: f_3, f_6, and f_{10}, we can see clear

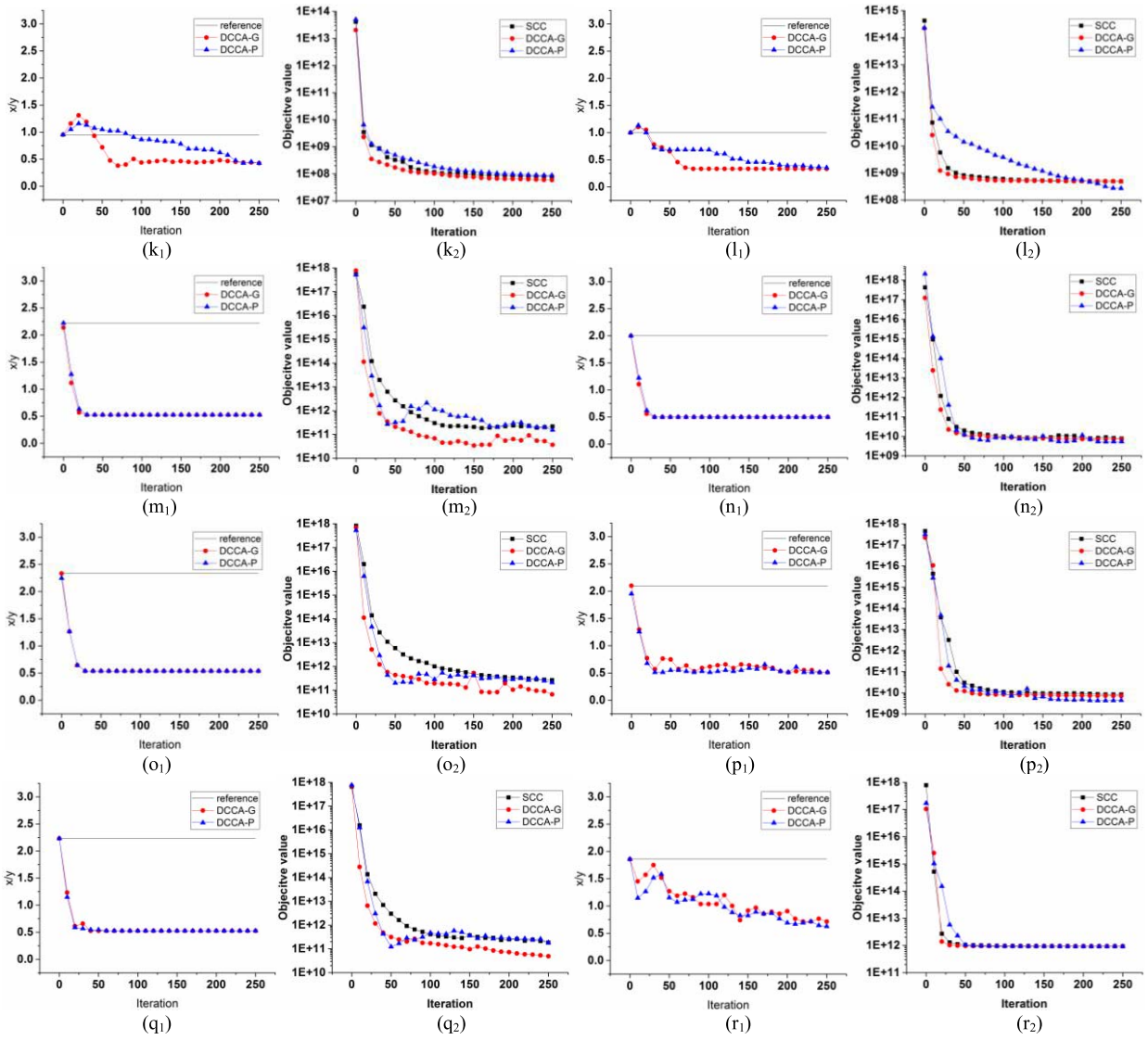

Fig. 6. Processor allocation and convergence behavior. Figures with odd number index show the results of processor allocation, such as (a_1), (b_1), etc. Horizontal axis represents the iteration number. Ordinate axis represents the processor allocating ratio between the number of processors of f_x and f_y. The black reference line represents the initial ratio. Figures with even number index show the results of convergence behavior, such as (a_2), (b_2), etc. Horizontal axis represents the iteration number. Ordinate axis represents the objective value.

decline of the execution time of DCCA on the other test cases. Specifically, for the totally seperable functions with evenly distributed importance, DCCA shows nearly linear speedup on f_1 and a little bit lower speedup on f_2. For the partially seperable functions: f_4, f_5, f_7–f_9, f_{11}, DCCA shows superlinear speedup. The reason of the superlinear speedup comes from the imbalance in the contribution of subcomponents. As in partially seperable functions of CEC'2013 benchmark, always some subcomponents are more important than the others. DCCA can focus major computing resources on them. The number of processors assigned to them during the execution is much more than evenly assigned. Using the extra assigned processors, these subcomponents can accelerate the optimization process in a superlinear way. Finally, for the overlapping functions f_{12}–f_{15}, DCCA still shows about linear speedup.

Overall, for the problems which have imbalanced subcomponents, DCCA can accelerate the optimizing process superlinearly. For the problems with even contribution, DCCA is still efficient that the growth of the processor quantity can bring nearly linear speedup.

V. Conclusion

In this paper, a distributed cooperation co-evolution architecture with adaptive computing resource allocation named DCCA is proposed to solve LSGO problems. The fundamental idea of the adaptive allocation is to utilize the imbalance of subcomponents' contributions to the global objective, and assign more processors to the subcomponents which have higher priorities. Meanwhile, two conformance policies named population policy and generation policy are designed to accommodate to the changing of the computing resource allocation. Experimental results show that DCCA is

effective to improve the performance of dCC, and also scalable and efficient in time consumption.

In future works, there are three main directions in which we can further explore the usability of dCC. The first one is to customize conformance policy for specific algorithm or problem. Since different algorithm has different convergence behavior, conformance policy in DCCA thus should be designed in accordance with the characteristic of the algorithm. Certainly, it also relates to the problem as shown in the experiment, but what the relationship between conformance policy and the type of the problem still remains an open issue that needs to be further studied. The second direction is to develop other ways to find and calculate the importance of subcomponents. Although the approach proposed in this paper works well on most of the benchmark functions, we find it highly depends on the chosen optimizer. If the chosen EA fails in optimizing some subcomponents, DCCA will consider them unimportant. Thus, algorithm-independent method should be developed to measure the importance of subcomponents. Third, as dynamic grouping methods still have advantages in optimizing overlapping functions, they should be introduced into dCC and corresponding computing resource allocation scheme should be developed for them.

REFERENCES

[1] D. Simon, *Evolutionary Optimization Algorithms*. Hoboken, NJ, USA: Wiley, 2013.

[2] R. Sarker, M. Mohammadian, and X. Yao, *Evolutionary Optimization*. Norwell, MA, USA: Kluwer Acad., 2002.

[3] 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.

[4] 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.

[5] A. LaTorre, S. Muelas, and J.-M. Peña, "A comprehensive comparison of large scale global optimizers," *Inf. Sci.*, vol. 316, pp. 517–549, Sep. 2015.

[6] 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.

[7] 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.

[8] H. Wang, S. Rahnamayan, and Z. Wu, "Parallel differential evolution with self-adapting control parameters and generalized opposition-based learning for solving high-dimensional optimization problems," *J. Parallel Distrib. Comput.*, vol. 73, no. 1, pp. 62–73, 2013.

[9] Y. Wang *et al.*, "Two-stage based ensemble optimization framework for large-scale global optimization," *Eur. J. Oper. Res.*, vol. 228, no. 2, pp. 308–320, 2013.

[10] T. Takahama and S. Sakai, "Large scale optimization by differential evolution with landscape modality detection and a diversity archive," in *Proc. IEEE CEC*, Brisbane, QLD, Australia, 2012, pp. 1–8.

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

[12] M. A. Potter and K. A. De Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evol. Comput.*, vol. 8, no. 1, pp. 1–29, 2000.

[13] M. A Potter and K. A. D. Jong, "A cooperative coevolutionary approach to function optimization," in *Proc. 3rd Conf. Parallel Problem Solving Nat.*, 1994, pp. 249–257.

[14] M. El-Abd, "A cooperative approach to the artificial bee colony algorithm," in *Proc. IEEE CEC*, Barcelona, Spain, 2010, pp. 1–5.

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

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

[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] W. Chen, T. Weise, Z. Yang, and K. Tang, "Large-scale global optimization using cooperative coevolution with variable interaction learning," in *Proc. PPSN*, 2010, pp. 300–309.

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

[20] 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.

[21] 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, p. 13, 2016.

[22] M. N. Omidvar, Y. Mei, and X. Li, "Effective decomposition of large-scale separable continuous functions for cooperative co-evolutionary algorithms," in *Proc. IEEE CEC*, Beijing, China, 2014, pp. 1305–1312.

[23] Y. Ren and Y. Wu, "An efficient algorithm for high-dimensional function optimization," *Soft Comput.*, vol. 17, no. 6, pp. 995–1004, 2013.

[24] 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 CEC*, Barcelona, Spain, 2010, pp. 1–8.

[25] X. Li and X. Yao, "Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms," in *Proc. IEEE CEC*, Trondheim, Norway, 2009, pp. 1546–1553.

[26] E. Sayed, D. Essam, and R. Sarker, "Dependency identification technique for large scale optimization problems," in *Proc. IEEE CEC*, Brisbane, QLD, Australia, 2012, pp. 1–8.

[27] M. N. Omidvar, X. Li, and X. Yao, "Smart use of computational resources based on contribution for cooperative co-evolutionary algorithms," in *Proc. 13th ACM GECCO*, 2011, pp. 1115–1122.

[28] M. Yang *et al.*, "Efficient resource allocation in cooperative co-evolution for large-scale global optimization," *IEEE Trans. Evol. Comput.*, vol. 21, no. 4, pp. 493–505, Aug. 2017.

[29] B. Kazimipour, M. N. Omidvar, X. Li, and A. K. Qin, "A sensitivity analysis of contribution-based cooperative co-evolutionary algorithms," in *Proc. IEEE CEC*, Sendai, Japan, 2015, pp. 417–424.

[30] M. N. Omidvar, B. Kazimipour, X. Li, and X. Yao, "CBCC3—A contribution-based cooperative co-evolutionary algorithm with improved exploration/exploitation balance," in *Proc. IEEE CEC*, Vancouver, BC, Canada, 2016, pp. 3541–3548.

[31] C. C. Vu, H. H. Nguyen, and L. T. Bui, "A parallel cooperative coevolution evolutionary algorithm," in *Proc. IEEE KSE*, 2011, pp. 48–53.

[32] V. Cevher, S. Becker, and M. Schmidt, "Convex optimization for big data: Scalable, randomized, and parallel algorithms for big data analytics," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 32–43, Sep. 2014.

[33] N. R. Sabar, J. Abawajy, and J. Yearwood, "Heterogeneous cooperative co-evolution memetic differential evolution algorithm for big data optimization problems," *IEEE Trans. Evol. Comput.*, vol. 21, no. 2, pp. 315–327, Apr. 2017.

[34] B. Cao, J. Zhao, Z. Lv, and X. Liu, "A distributed parallel cooperative coevolutionary multi-objective evolutionary algorithm for large-scale optimization," *IEEE Trans. Ind. Informat.*, vol. 13, no. 4, pp. 2030–2038, Aug. 2017.

[35] 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.

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

[37] G. Roy *et al.*, "A distributed pool architecture for genetic algorithms," in *Proc. IEEE CEC*, Trondheim, Norway, 2009, pp. 1177–1184.

[38] X. Li, K. Tang, M. N. Omidvar, Z. Yang, and K. Qin, "Benchmark functions for the CEC 2013 special session and competition on large-scale global optimization," Evol. Comput. Mach. Learn. Group, RMIT Univ., Melbourne, VIC, Australia, Rep., 2013. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.481.8986&rep=rep1&type=pdf

[39] 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.

[40] A. Mendiburu, J. A. Lozano, and J. Miguel-Alonso, "Parallel implementation of EDAs based on probabilistic graphical models," *IEEE Trans. Evol. Comput.*, vol. 9, no. 4, pp. 406–423, Aug. 2005.

[41] 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.
[42] 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.
[43] 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.
[44] 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.
[45] M. Davis, L. Liu, and J. G. Elias, "VLSI circuit synthesis using a parallel genetic algorithm," in *Proc. IEEE CEC*, Orlando, FL, USA, 1994, pp. 104–109.
[46] R. Subbu and A. C. Sanderson, "Modeling and convergence analysis of distributed coevolutionary algorithms," *IEEE Trans. Syst. Man Cybern. B Cybern.*, vol. 34, no. 2, pp. 806–822, Apr. 2004.
[47] F. Seredynski, "Competitive coevolutionary multi-agent systems: The application to mapping and scheduling problems," *J Parallel Distrib. Comput.*, vol. 47, no. 1, pp. 39–57, 1997.
[48] Z. Yang, K. Tang, and X. Yao, "Self-adaptive differential evolution with neighborhood search," in *Proc. IEEE CEC*, 2008, pp. 1110–1116.
[49] 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.

Ya-Hui Jia (S'14) received the bachelor's degree from Sun Yat-sen University, Guangzhou, China, in 2013, where he is currently pursuing the Ph.D. degree.

He is 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 software engineering, cloud computing, and intelligent transportation.

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

He is currently a Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou. He has published over 80 papers in international journals and conferences, including over 20 papers in IEEE TRANSACTIONS journals. His current research interests include swarm intelligence algorithms and their applications on cloud computing, operations research, and software engineering.

Dr. Chen was a recipient of the IEEE Computational Intelligence Society Outstanding Dissertation Award in 2016 for his doctoral thesis, and the National Science Fund for Excellent Young Scholars in 2016.

Tianlong Gu received the M.Eng. degree from Xidian University, Xi'an, China, in 1987, and the Ph.D. degree 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 Post-Doctoral 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 from Shanghai Jiaotong University, Shanghai, China, in 2004.

He is currently a Professor with the School of Information Science and Engineering, Shandong Normal University, Jinan, China. He was an Associate Professor with the Department of Computer Science, Shandong Normal University from 2004 to 2005. 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.

Hua-Qiang Yuan received the Ph.D. degree 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. and M.A.Sc. degrees in electrical engineering from the State University of New York at Buffalo, Buffalo, NY, USA, and the University of Waterloo, Waterloo, ON, Canada, in 1983 and 1985, respectively, and the Ph.D. degree from the University of Hagen, Hagen, Germany, in 1996.

From 1985 to 1987, he was a Diagnostic Engineer with the Control Data Canada, Mississauga, ON, Canada, where he designed the diagnostic software to detect the manufacture faults of the very large scale integration chips in the Cyber 430 machine. He later joined the Bell Northern Research Canada as a Member of Scientific Staff. In 1990, he joined the City University of Hong Kong, Hong Kong, as a Lecturer with the Department of Electronic Engineering. He is currently a Professor with the Department of Computer Science. His current research interests include pattern recognition, evolutionary computations, and video analytics.

Dr. Kwong has been the Vice President for IEEE Systems, Man and Cybernetics for Conferences and Meetings since 2014. He was elevated to IEEE Fellow for his contributions on Optimization Techniques for Cybernetics and Video coding in 2014. He is also appointed as a IEEE Distinguished Lecturer for IEEE SMC Society from 2017.

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 and the ACM Guangzhou Chapter.