IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 4, 2019, accepted August 27, 2019, date of publication September 2, 2019, date of current version November 20, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2938765

A Parallel Divide-and-Conquer-Based Evolutionary Algorithm for Large-Scale Optimization

PENG YANG, (Member, IEEE), KE TANG, (Senior Member, IEEE), AND XIN YAO, (Fellow, IEEE)

University Key Laboratory of Evolving Intelligent Systems of Guangdong Province, Shenzhen Key Laboratory of Computational Intelligence, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China

Corresponding author: Xin Yao (xiny@sustech.edu.cn)

This work was supported in part by the Natural Science Foundation of China under Grant 61806090 and Grant 61672478, in part by the Program for Guangdong Introducing Innovative and Enterpreneurial Teams under Grant 2017ZT07X386, in part by the Shenzhen Peacock Plan under Grant KQTD2016112514355531, and in part by the Program for University Key Laboratory of Guangdong Province under Grant 2017KSYS008.

ABSTRACT Large-scale optimization problems that involve thousands of decision variables have extensively arisen from various industrial areas. As a powerful optimization tool for many real-world applications, evolutionary algorithms (EAs) fail to solve the emerging large-scale problems both effectively and computationally efficiently. In this paper, we propose a novel Divide-and-Conquer (DC) based EA that can not only produce high-quality solutions by solving sub-problems separately, but also benefits significantly from the power of parallel computing by solving the sub-problems simultaneously. Existing DC-based EAs that were thought to enjoy the same advantages of the proposed algorithm, are shown to be practically incompatible with the parallel computing scheme, unless some trade-offs are made by compromising the solution quality.

INDEX TERMS Parallel evolutionary algorithms, large-scale optimization, divide-and-conquer.

I. INTRODUCTION

Nowadays, with the trend of globalization, traditional decision-making components appear to produce only local optima. For example, the design of the global supply chain involves network planning, manufacturing production, warehousing operations, transportation planning, urban delivering, and so on [1]. These components are heavily interdependent in nature, optimizing one may result in the degeneration of others. Only regarding them as an integrated optimization problem, could one produce a solution satisfying all components. In exchange, one has to optimize much more decision variables than usual, which brings in the widely-existed large-scale optimization problem [2], [3].

Evolutionary Algorithms (EAs), which work by searching the solution space of the targeted problem iteratively and in a randomized way, have shown powerful performance in solving many real-world optimization problems [4], [5]. Unfortunately, the search-based core makes EAs ineffective and inefficient for solving large-scale optimization problems for two reasons:

1) As the number of decision variables increases, the solution space of the problem increases exponentially, preventing EAs exploring effectively within reasonable amount of search iterations.
2) As the search operators are applied to high-dimensional candidate solutions, the computational time costs at each iteration can become expensive.

For many large-scale real-world optimization problems with real-time constraints, e.g., the placement of virtual machines in large-scale cluster [6], the above two drawbacks make EAs ill-equipped to handle them satisfactorily.

For the last decade, researchers have made great efforts on solving the above two drawbacks [7], [8]. Among the existing methods, the idea of Divide-and-Conquer (DC) has attracted most research attention, as it is often viewed as an integrated solution for both drawbacks [8], [9]. In brief, DC-based EAs first decompose the original problem into multiple small-scale sub-problems, and then solve them respectively by existing EAs. Since small-scale sub-problems are usually easier to solve, a significant reduction of the required

The associate editor coordinating the review of this article and approving it for publication was Ran Cheng.

number of search iterations can be expected, which in turn leads to a satisfactory improvement of solution quality if the number of total iterations is fixed. Besides, ideally, multiple sub-problems can be solved in parallel, which can decrease the computational costs at each iteration dramatically.

However, despite that existing DC-based EAs have remarkably reduced the search iterations [10]–[12], most of them cannot be directly implemented in parallel. This situation has been greatly neglected by researchers, because it is commonly believed that their serial workflow can be easily modified into parallel. In this paper, we first discuss that the modification of existing DC-based EAs into parallel could considerably compromise their effectiveness of reducing the search iterations, making it costly to accelerate the optimization via parallelization. Hence, a DC-based EA that is essentially parallelizable would be more meaningful and thus should be investigated.

In fact, the difficulty of parallelizing existing DC-based EAs lies in the construction of objective functions for sub-problems. Specifically, to build the objective function for a sub-problem, existing works require the best partial solutions from other sub-problems. On the other hand, to decide such a best partial solution to a sub-problem, existing works require knowing its objective function first. This circular dependency difficulty makes the sub-problems unable to be solved in parallel. In this paper, we propose a novel DC-based EA that is naturally suited to solve sub-problems in parallel. The core idea is to pre-select the best partial solution for each sub-problem via some subproblem-independent meta-model rather than the objective function of that sub-problem. By doing so, the difficulty encountered when parallelizing existing DC-based EAs can be fully avoided.

The reminder of this paper is as follow. Section II reviews the background of this work. The difficulties of parallelizing existing DC-based EAs are analyzed in Section III. Inspired by that, a novel algorithm that is naturally suited to solve sub-problems in parallel is proposed Section IV. The above studies are empirically verified in Section V, where both the advantages and disadvantages of the proposed algorithm are discussed. Section VI concludes this paper and gives some directions for the future.

II. BACKGROUND

A. THE DIFFICULTIES OF EAs IN LARGE-SCALE OPTIMIZATION

Given a real-world optimization problem, one can design an objective function either mathematically or by simulation. Without the loss of generality, let the minimization problem be an example. Generally, the optimization problem can be formulated as $x^* = \min_{\forall x \in \mathcal{X}} f(x)$, where $x = [x_1, x_2, \ldots, x_D]$ represents a D-dimensional candidate solution to f, $\mathcal{X}$ is the bounded D-dimensional solution space that contains all candidate solutions, and x^* is the optimum to f within $\mathcal{X}$.

To solve an optimization problem, EAs actually search $\mathcal{X}$ for x^* in an iterative way. Specifically, EAs work by first randomly initializing a population of (multiple) candidate solutions in $\mathcal{X}$, and then improving them iteratively during the search course, lastly outputting the best ever found candidate as the final solution. In the t-th iteration ($t < T$), a new population $\tilde{X}^t$ will be first generated based on the last population X^t via some specific randomized search operators, and then the candidate solutions $\tilde{x}^t$ in the new population $\tilde{X}^t$ will be evaluated with f. After that, the candidate solutions in both X^t and $\tilde{X}^t$ will be selected to form X^{t+1} for the next iteration, based on their evaluated function values.

When solving real-world optimization problems, the computational efficiency is an important requirement in addition to the solution quality, and sometimes can be a hard constraint. That is, the problem must be solved in a given time budget T, which would otherwise fail the quality of service provided by an application [13]. In general, the computational efficiency of an EA is governed by two independent factors: the iterations of the whole search process and the computational time costed in each iteration. Unfortunately, both factors are very sensitive to the dimensionality D of the problem. When D becomes large, the real-time requirement of the problem will impose great challenges on EAs. On the other hand, these large-scale problems are ubiquitous [14]. Therefore, the question of how to solve large-scale optimization problems with good enough solutions while keeping the computational costs sufficiently low lies at the core of the research of EAs.

B. THE DIVIDE-AND-CONQUER BASED EAs

In the literature, various ideas have been investigated for EAs on large-scale optimization problems. For reducing the search iterations, there are two major ways. One is to enhance the search ability of existing EAs by re-scheduling the local search operators [15]–[17]; while the other is to simplify the problem via Divide-and-Conquer (DC) [18]–[20] or Dimension Reduction [21], [22]. For saving the computational time spent in each iteration, parallel computing or distributed computing techniques are frequently employed to optimize individual solutions or decision variables on different threads [7], [8], [23], [24]. Among them, the DC methodology has been frequently introduced into EAs for large-scale optimization problems, since it can be regarded as an integrated solution to improve the above two factors for EAs.

The DC-based EAs consist of three major steps. First, the original D-dimensional problem is exclusively divided into M D-dimensional sub-problems, where $\sum_{i=1}^{M} d_i = D$. Second, each sub-problem is solved by an EA, respectively. Last, the d_i-dimensional partial solution to each sub-problem is merged to form the D-dimensional complete solution to the original problem as the final output. Ideally, in case the sub-problems can be made independent of each other, they can be solved separately and simultaneously. By 'separately', we mean that each EA only has to solve a d_i-dimensional small-scale sub-problem with a sub-space sized of $|\mathcal{X}|^{\frac{d_i}{D}}$, where $|\mathcal{X}|$ indicates the hyper-volume of the solution

space $\mathcal{X}$. As a result, the iterations of the whole search process can be considerably reduced by DC, as $\sum_{i=1}^{M} |\mathcal{X}|^{\frac{d_i}{D}} < |\mathcal{X}|$. By 'simultaneously', we mean that the M sub-problems can be parallel solved with the aid of multiple processors of a work-station, distributed computing resources in traditional on premise data center, or cloud computing services, where the computational cost in each iteration can be made at most M times cheaper. In case M can be made close to D, the computational cost in each iteration enlarged by the increase of D can be marginal.

In practice, the interdependencies among sub-problems are the main barrier that hinders the above two advantages of DC-based EAs, motivating rich volume of research efforts for eliminating them via advanced decomposition strategies [25]. Yang *et al.* [11] proposed a random grouping technique that periodically randomly decomposed the decision variables into equal sized groups, showing that the probability of generating interdependent groups dropped down as the optimization process went on. Omdivar *et al.* [26] and Yang *et al.* [27] later found that the group sizes and grouping frequency had great impacts on the performance of random grouping. Chen *et al.* [28] introduced the mathematical definition of interacting decision variables to guide the decomposition of sub-problems much more accurately. It checked for each pair of decision variables that whether the definition was violated. If so, such pair of decision variables were deemed to be interdependent and grouped together. More works [12], [18], [29] improved the grouping accuracy by referencing a tighter interdependency among decision variables, i.e., the additive separability.

Indeed, the above works have pushed the boundary of decomposition accuracy to a great extent, leading to significant reductions of search iterations on large-scale optimization problems. However, these works have not discussed how to parallelize solving sub-problems. In fact, most existing works cannot be directly implemented in parallel. Though some other researchers tried to propose parallel schemes for existing DC-based EAs [30], [31], in the next section, we discuss that those schemes are highly likely to dramatically destroy the effectiveness of the above algorithms.

III. THE DIFFICULTIES OF PARALLELIZING EXISTING DC-BASED EAs

To explain the difficulties of existing DC-based EAs on parallelization, we first describe the phase of defining objective functions for the sub-problems, which forms the key step of applying DC. Then we show that the mostly adopted way for defining the objective functions actually follows a serial workflow that cannot be directly parallelized. Lastly, we analyze that, to parallelize existing DC-based EAs, the effectiveness of reducing search iterations could be considerably degenerated.

A. DEFINING OBJECTIVE FUNCTIONS FOR SUB-PROBLEMS

By applying DC-based EAs to a problem f, one has to first decompose f into M sub-problems, and then solve each of them by an EA [32]. Before solving a sub-problem, its objective function should be known, so that its candidate partial solutions can be evaluated. Clearly, the original D-dimensional objective function f cannot be directly used to evaluate the partial solutions to each d_i-dimensional sub-problem, due to the mismatch of dimensionality. Thus, one has to derive M new d_i-dimensional objective functions from f for the sub-problems, denoted as f_i, respectively. For black-box optimization problems, where we may not know the explicit formula of f, one practical way to build f_i is by complementing. That is, given a d_i-dimensional sub-problem, one can transform the D-dimensional f into a d_i-dimensional f_i by fixing the rest $(D - d_i)$-dimensional decision variables involved in f with some feasible values. For simplicity, let us denote the set of decision variables belonging to the i-th sub-problem as $\mathcal{S}_i$, where $|\mathcal{S}_i| = d_i$, $i = 1, 2, \ldots, M$, then we have

$$f_i(x_{\mathcal{S}_i}) = f([v_{\mathcal{S}_1}, v_{\mathcal{S}_2}, \ldots, v_{\mathcal{S}_{i-1}}, x_{\mathcal{S}_i}, v_{\mathcal{S}_{i+1}}, \ldots, v_{\mathcal{S}_M}]), \quad (1)$$

where $v_{\mathcal{S}_i}$ denotes the fixed values for the decision variables in the j-th sub-problem, $j = 1, 2, \ldots, M$, and $j \neq i$. This is identical to first complement a d_i-dimensional partial solution into D-dimensional with those feasible values, and then evaluate it by f.

B. THE SERIAL WORKFLOW OF EXISTING DC-BASED EAs

For each i-th sub-problem, there are in total $|\mathcal{X}|^{D-d_i}$ possibilities to fix the $(D - d_i)$-dimensional decision variables, resulting in $|\mathcal{X}|^{D-d_i}$ candidate f_i. As f_i largely determines the search direction of an EA in the i-th sub-problem, setting different f_i will lead to different search trajectories. Eventually, it will result in different qualities of the output solution within a given time budget, e.g., number of iterations.

To achieve good results, existing DC-based EAs have tried different ways to build f_i [33]–[38]. In general, the values of the $(D - d_i)$-dimensional decision variables are usually fixed by filling with dedicated partial solutions from the existing (sampled) population of other $M - 1$ sub-problems. In other words, to build f_i using Eq.(1), any $v_{\mathcal{S}_j}$ is a d_j-dimensional partial solution selected from the population that has already been generated in the j-th sub-problem. The difference among those related works lies in how the fixed values are selected. For example, Potter and De Jong [39] randomly selects the partial solutions in the current population (iteration) of other $M - 1$ sub-problems. The algorithms in [12], [18], [19], [27], [28], [40] select the best partial solutions in the current population of other $M - 1$ sub-problems. Other works like [35], [36], [40] select the best partial solutions from historical populations of other $M - 1$ sub-problems. Among them, the f_i built with the best partial solutions in the current population of other $M - 1$ sub-problems has been empirically shown to perform the best, and thus has been mostly adopted in nowadays DC-based EAs.

Specifically, let us denote the best partial solution in the t-th population of the j-th sub-problem as $b^t_{\mathcal{S}_j}$. Then,

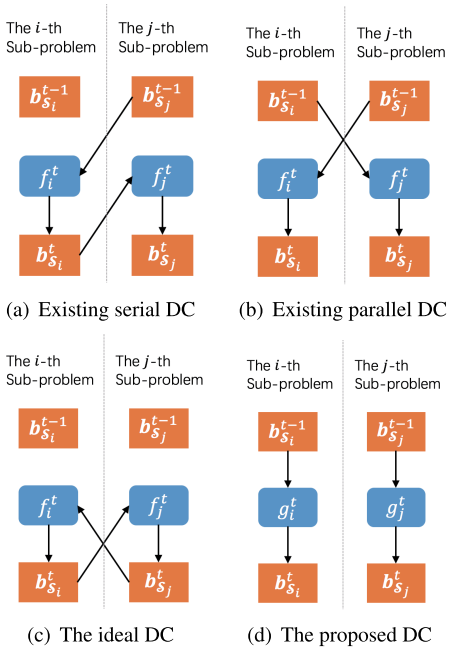


FIGURE 1. The illustrations of different workflows of DC-based EAs. Among them, (a) shows the workflow of existing serial DC-based EAs, (b) shows the workflow of existing parallel DC-based EAs, (c) shows the workflow of the ideal way of defining consistent sub-problem objectives, and (d) shows the workflow of the proposed algorithm. For simplicity, we only depict two sub-problems for each workflow. In each workflow, a black arrow from "a" to "b" means to obtain "b", "a" should be known first.

the mostly adopted objective function for the i-th sub-problem can be described as Eq.(2).

$$f_i^t(x^t_{\mathcal{S}_i}) = f([b^t_{\mathcal{S}_1}, b^t_{\mathcal{S}_2}, \ldots, b^t_{\mathcal{S}_{i-1}}, x^t_{\mathcal{S}_i}, b^{t-1}_{\mathcal{S}_{i+1}}, \ldots, b^{t-1}_{\mathcal{S}_M}]). \quad (2)$$

We specially denote the objective function with a superscript t as f_i^t in Eq.(2), because it changes over iterations once some $b^t_{\mathcal{S}_j}$ varies from iteration to iteration, which is highly likely. Notice that, for the sub-problems indexed from the first to the $(i-1)$-th, their best partial solutions are selected from the t-th population. This scheme actually prevents the existing works from being directly parallelizable. That is, to build f^t_{i+1}, it requires $b^t_{\mathcal{S}_i}$ to be known. On the other hand, to obtain each $b^t_{\mathcal{S}_i}$, one has to evaluate all the candidate partial solutions in the i-th sub-problem via the objective function f^t_i. In a word, f^t_i should be built before f^t_{i+1}. As a result, for any pair of sub-problems, they cannot be directly solved in parallel. For illustration, the workflow of Eq.(2) are shown in Fig.1(a).

C. THE GAPS BETWEEN SERIAL AND PARALLEL WORKFLOWS

One can alternatively parallelize existing DC-based EAs by defining f^t_i with historical best partial solutions [40], in which all the f^t_i can be built at the same time. A typical implementation of this idea can be described as Eq.(3),

$$f_i^t(x^t_{\mathcal{S}_i}) = f([b^{t-1}_{\mathcal{S}_1}, b^{t-1}_{\mathcal{S}_2}, \ldots, b^{t-1}_{\mathcal{S}_{i-1}}, x^t_{\mathcal{S}_i}, b^{t-1}_{\mathcal{S}_{i+1}}, \ldots, b^{t-1}_{\mathcal{S}_M}]), \quad (3)$$

where all the fixed values are selected as the best partial solutions from the $(t-1)$-th population [40]. An illustration of the workflow of Eq.(3) is given in Fig.1(b). Because Eq.(3) looks quite close to Eq.(2), researchers often assume that the existing DC-based EAs can easily be parallelized with neglectable compromises on effectiveness [8], [25], [40]. However, in the following, we discuss that using Eq.(3) can lead to much worse optimization results than using Eq.(2), inferring that it is difficult to implement the existing DC-based EAs in parallel.

Recall that, the final output of a DC-based EA is the merge of the best partial solutions to each sub-problem at the final T-th population. Also note that, any partial solution $x^T_{\mathcal{S}_i}$ with good quality to f^T_i may not necessarily be a component of a good complete solution to f. This divergence stems from complementing $x^T_{\mathcal{S}_i}$ while being evaluated. To calculate the final solution quality with f, $x^T_{\mathcal{S}_i}$ is complemented with $b^T_{\mathcal{S}_j}$ in each j-th sub-problem, since the best partial solutions will be merged at the T-th population. However, neither Eq.(2) nor Eq.(3) can guarantee the same complement to $x^T_{\mathcal{S}_i}$ while defining f^T_i. That is, in some sub-problems, if not all, the best partial solutions in the $(T-1)$-th population will be selected, which are not necessarily to be equal to $b^T_{\mathcal{S}_j}$. The situation remains the same for any t-th ($t < T$) intermediate population. Therefore, the divergence between each f^t_i and f is accumulated along the whole search process, leading the existing DC-based EAs to possibly produce low-quality solution to f.

Based on the discussion above, we can analyze the gap between Eq.(2) and Eq.(3), i.e., the serial and parallel workflows of existing DC-based EAs, as follows. We first describe the above-mentioned ideal way of defining each f^t_i as Eq.(4), which is consistent with the evaluation of the final solution quality to f.

$$f_i^t(x^t_{\mathcal{S}_i}) = f([b^t_{\mathcal{S}_1}, b^t_{\mathcal{S}_2}, \ldots, b^t_{\mathcal{S}_{i-1}}, x^t_{\mathcal{S}_i}, b^t_{\mathcal{S}_{i+1}}, \ldots, b^t_{\mathcal{S}_M}]). \quad (4)$$

Then we can measure the divergence of Eq.(2) and Eq.(4), and the divergence of Eq.(3) and Eq.(4). Finally, we give the gap between the serial and parallel workflows as the difference between the two divergences. For illustration, the workflow of Eq.(4) is shown in Fig.1(c), which clearly shows a circular dependency dilemma as mentioned previously.

The divergence is calculated as the probability that either Eq.(2) or Eq.(3) works differently from Eq.(4) at each t-th iteration. By 'different', we mean that not all $b^{t-1}_{\mathcal{S}_j}$ in Eq.(2) or Eq.(3) will remain in the t-th population, i.e., $b^{t-1}_{\mathcal{S}_j} \neq b^t_{\mathcal{S}_j}$. For simplicity, let us assume that each sub-problem is equally important to the original problem, and the search operator employed in each sub-problem independently has a probability $1-p$ to produce a new better partial solution, i.e., $b^{t-1}_{\mathcal{S}_i} \neq b^t_{\mathcal{S}_i}$, $i = 1, 2, \ldots, M$. Thus, with a probability p, we have $b^{t-1}_{\mathcal{S}_i} = b^t_{\mathcal{S}_i}$ for each i-th sub-problem. Let us also denote the divergence of Eq.(2) and Eq.(4) over all sub-problems as div_1, and the divergence of Eq.(3) and Eq.(4) over all sub-problems

as div_2. Then we have the following equations.

$$\begin{aligned} div_1 &= 1 - \prod_{i=1}^{M} P(x^{t-1}_{\mathcal{S}_M} = x^t_{\mathcal{S}_M}, \ldots, x^{t-1}_{\mathcal{S}_i} = x^t_{\mathcal{S}_i}) \\ &= 1 - \prod_{i=1}^{M} \prod_{j=i+1}^{M} P(x^{t-1}_{\mathcal{S}_j} = x^t_{\mathcal{S}_j}) \\ &= 1 - \prod_{i=1}^{M} p^{M-i+1} \\ &= 1 - p^{\frac{M(M-1)}{2}} \end{aligned} \tag{5}$$

$$\begin{aligned} div_2 &= 1 - \prod_{i=1}^{M} P(x^{t-1}_{\mathcal{S}_M} = x^t_{\mathcal{S}_M}, \ldots, x^{t-1}_{\mathcal{S}_{i+1}} = x^t_{\mathcal{S}_{i+1}}, \\ &\qquad x^{t-1}_{\mathcal{S}_{i-1}} = x^t_{\mathcal{S}_{i-1}}, \ldots, x^{t-1}_{\mathcal{S}_1} = x^t_{\mathcal{S}_1}) \\ &= 1 - \prod_{i=1}^{M} \prod_{j=1, j\neq i}^{M} P(x^{t-1}_{\mathcal{S}_j} = x^t_{\mathcal{S}_j}) \\ &= 1 - \prod_{i=1}^{M} p^{M-1} \\ &= 1 - p^{M(M-1)} \end{aligned} \tag{6}$$

where the last term in either Eq.(5) or Eq.(6) describes the probability that all $b^{t-1}_{\mathcal{S}_j}$ in Eq.(2) or Eq.(3) remain in the t-th population, i.e., $b^{t-1}_{\mathcal{S}_j} = b^t_{\mathcal{S}_j}$. Therefore, by defining f^t_i with Eq.(2), there is a probability div_1 that not all $b^t_{\mathcal{S}_j}$ is a good component to f. Similarly, by defining f^t_i with Eq.(3), there is a probability div_2 that not all $b^t_{\mathcal{S}_j}$ is a good component to f. Then the gap between Eq.(2) and Eq.(3) at each iteration can be expressed as $\frac{div_2}{div_1} = 1 + p^{\frac{M(M-1)}{2}} > 1$, and the accumulated gap will exponentially explode as $(\frac{div_2}{div_1})^T|_{T\to\infty} = (1 + p^{\frac{M(M-1)}{2}})^T|_{T\to\infty} \to \infty$. As a result, by parallelizing existing DC-based EAs with Eq.(3), the reduction of the search iterations, or the quality of the final output if T is fixed, can be considerably deteriorated.

IV. THE PROPOSED PARALLEL DC-BASED EA

As discussed above, Eq.(4) can evaluate partial solutions with the quality consistent with f. Unfortunately, existing works cannot adopt it, due to a difficulty similar to the so-called circular dependency. That is, in Eq.(4), to obtain each $b^t_{\mathcal{S}_j}$, one has to evaluate all the candidate partial solutions in the j-th sub-problem via the objective function f^t_j. On the other hand, to build each f^t_i, $i = 1, 2, \ldots, M$ and $i \neq j$, one has to first obtain the best partial solutions $b^t_{\mathcal{S}_j}$ in other j-th sub-problems. In a word, f^t_i and f^t_j are mutually prior conditions required by each other. As a result, for any pair of sub-problems, their objective functions cannot be built simultaneously and thus themselves cannot be solved in parallel. For illustration, the workflow of Eq.(4) are shown in Fig.1(c).

In this paper, we propose not to decide each $b^t_{\mathcal{S}_j}$ by f^t_j. Instead, each $b^t_{\mathcal{S}_j}$ is first pre-selected via a meta-model, denoted as g^t_j, which is not built upon any $b^t_{\mathcal{S}_i}$. By this means, the circular dependency difficulty of Eq.(4) is tackled, as shown in Fig.1(d). Based on that, the resultant algorithm, called Naturally Parallelizable Divide-and-Conquer (NPDC), is thus both naturally parallelizable and consistent with f. In this section, the meta-model g^t_j is first detailed, then the proposed NPDC algorithm is described, lastly the parallelism of NPDC is discussed in details.

A. THE PROPOSED META-MODEL OF NPDC

The meta-model is basically designed for the (1 + 1)-EA paradigm [41], [42]. The (1 + 1)-EA paradigm is defined as that an offspring solution will be first generated based its parent solution, and then directly compared to the parent for survival. Therefore, in the j-th sub-problem at the t-th population, g^t_j works for pair-wise comparing one parent partial solution and its offspring partial solution, and deciding the better one. The underlying reason is two-fold. First, the (1 + 1)-EA paradigm is very simple, for which the meta-model will be kept computationally efficient and easily understandable. Second, the (1 + 1)-EA paradigm is widely applied in many well-established EAs, e.g., Differential Evolution (DE) [43], Particle Swarm Optimization (PSO) [44], for which the offspring and its parent has the one-on-one relationship. The meta-model can easily be generalized to those kinds of population-based EAs. To this end, one can simply employ λ meta-models if the population size is λ, each of which works for an individual in the population under the (1+1)-EA paradigm. For other EAs like Estimation of Distribution Algorithms [45] that directly operate at the population level rather than the individual level, there does not exist the one-on-one relationship between parents and offspring. Thus, the proposed meta-model cannot be directly applied to them.

The meta-model is also designed for the 1-dimensional sub-problems. In the literature, there are lots of meta-models that aim to evaluate the quality of a candidate solution instead of the original computationally expensive objective function [46], [47]. However, these meta-models often require huge number of labelled data since they usually simulate the global solution space with relatively high-dimension, which in turn require large numbers of fitness function evaluations. As there are $M \times \lambda$ meta-models in NPDC, directly employing existing meta-models in this paper is computationally unaffordable. Considering this, in order to keep it efficient enough, the proposed meta-model only focuses on estimating the local area of a 1-dimensional sub-space, where the number of labelled data can be minimized. In this regard, f is actually decomposed into D sub-problems, each of which is 1-dimensional, i.e., $M = D$.

For simplicity and clarity, we will illustrate the meta-model in case of $\lambda = 1$. To be specific, under the (1 + 1)-EA paradigm, the best partial solution at the last iteration can be simply regarded as the parent at the current iteration. Therefore, the parent partial solution in the j-th sub-problem

at the t-th population can be denoted as b_j^{t-1}. Here we use the denotation b_j^{t-1}, rather than $b_{\mathcal{S}_j}^{t-1}$ as in the previous sections, because each j-th sub-problem is actually 1-dimensional and thus $\mathcal{S}_j = \{j\}$. Then, g_j^t works for pair-wise comparing b_j^{t-1} and $\tilde{x}_j^t$, and deciding the better one as b_j^t, as Eq.(7) shows.

$$b_j^t = g_j^t(b_j^{t-1}, \tilde{x}_j^t), \tag{7}$$

where $\tilde{x}_j^t$ is the offspring partial solution generated based on its parent b_j^{t-1}.

As it is in 1-dimensional, b_j^{t-1} and $\tilde{x}_j^t$ can be directly compared by their values. Therefore, the meta-model g_j^t actually learns the relationship between their qualities and values. More specifically, the value of the offspring $\tilde{x}_j^t$ can be either larger or smaller to the value of its parent b_j^{t-1}, and the quality of $\tilde{x}_j^t$ can be either superior or inferior to the quality of b_j^{t-1}.[1] If the landscape of the solution space changes smoothly, g_j^t can use the historical comparison results to infer the current comparison between b_j^{t-1} and $\tilde{x}_j^t$ in a probabilistic way. Note that, the local landscape is highly likely to be asymmetrical to b_j^{t-1}. Then we can use PL_j^t to denote the probability that the value of $\tilde{x}_j^t$ is larger than the value of b_j^{t-1} and the quality of $\tilde{x}_j^t$ is superior to the quality of b_j^{t-1}. Similarly, PS_j^t denotes the probability that the value of $\tilde{x}_j^t$ is smaller than the value of b_j^{t-1} and the quality of $\tilde{x}_j^t$ is superior to the quality of b_j^{t-1}. Based on that, g_j^t can be defined as Eq.(8).

$$g_j^t(b_j^{t-1}, \tilde{x}_j^t) = \begin{cases} \tilde{x}_j^t & \text{if} \quad \tilde{x}_j^t < b_j^{t-1} \text{ and } PS_j^t < r \\ & \text{or } \tilde{x}_j^t > b_j^{t-1} \text{ and } PL_j^t < r \\ b_j^{t-1} & \text{otherwise} \end{cases} \tag{8}$$

where r indicates a function that returns a random variable uniformly sampled in the range of $[0, 1]$. Eq.(8) actually says that given $\tilde{x}_j^t$ is larger/smaller than b_j^{t-1}, there is a probability PL_j^t/PS_j^t that $\tilde{x}_j^t$ is superior to b_j^{t-1}. Ideally, if the landscape increases/decreases monotonously along the 1-dimensional solution space, PL_j^t/PS_j^t equals to 1.00. Unfortunately, it is often the case that the landscape changes from areas to areas in the solution space. Thus, both PL_j^t and PS_j^t need to be tuned during the optimization process. Assuming that the landscape changes smoothly and the value of $\tilde{x}_j^t$ can be generated close to b_j^{t-1}, both probabilities can be adjusted slightly over iterations according to the 1/5 success rule [48], which is a heuristic rule for tuning parameters locally, as Eq.(9).

$$\begin{aligned} PS_j^{t+1} &= PS_j^t \cdot \exp^{\frac{1}{\sqrt{2}}}[\mathbb{I}_{\tilde{x}_j^t < b_j^{t-1}} \cdot (\mathbb{I}_\Theta - \frac{1}{5})] \\ PL_j^{t+1} &= PL_j^t \cdot \exp^{\frac{1}{\sqrt{2}}}[\mathbb{I}_{\tilde{x}_j^t > b_j^{t-1}} \cdot (\mathbb{I}_\Theta - \frac{1}{5})] \\ \text{where } \Theta &= f([b_1^t, b_2^t, \ldots, b_D^t]) < f([b_1^{t-1}, b_2^{t-1}, \ldots, b_D^{t-1}]). \end{aligned} \tag{9}$$

[1]The equivalent case is highly unlikely and thus can be practically assigned to either inequivalent case.

In Eq.(9), $\mathbb{I}_\Theta$ is an indicator function that returns 1 if event Θ is true and 0 otherwise. Θ denotes the comparison on the qualities between each pair of b_j^{t-1} and b_j^t. It is calculated after the b_j^t in each j-th sub-problem is decided via g_j^t. It says that if the quality of $\tilde{x}_j^t$ is superior to the quality of b_j^{t-1}, and if the value of $\tilde{x}_j^t$ is larger/smaller than the value of b_j^{t-1}, the probability PL_j^t/PS_j^t should be increased with a factor of $\exp^{\frac{1}{\sqrt{2}}}(\frac{4}{5})$. Otherwise, if the quality of $\tilde{x}_j^t$ is inferior to the quality of b_j^{t-1}, the corresponding probability should be reduced with a factor of $\exp^{\frac{1}{\sqrt{2}}}(-\frac{1}{5})$. After tuning the two probabilities with Eq.(9), g_j^t is actually adjusted for one iteration and thus re-denoted as g_j^{t+1}.

Both PS_j^0 and PL_j^0 are initialized as 1.00 at the beginning of the search. This means that the randomly initialized parent b_j^0 is assumed to be always inferior to its offspring $\tilde{x}_j^1$, which is reasonable. During the adjustment, both PS_j^0 and PL_j^0 should be kept always no larger than 1.00, and no smaller than 0.00. However, if either of them equals 0.00, there will be no chance for them to be changed according to Eq.(9). And the search process will stagnate since any $\tilde{x}_j^t$ will be regarded as inferior to b_j^{t-1} and thus b_j^t always equals to b_j^{t-1} according to Eq.(9). In practice, we let them to be no smaller than a very small value, i.e., $\frac{2}{D}$. Given an offspring is on average equally likely to be generated either smaller or larger than the parent, then it is guaranteed that each decision variable has at least $\frac{1}{D}$ probability to be changed in each iteration, which reduces to the well-known uniform mutation operator.

B. THE DETAILS OF NPDC

NPDC follows the divide-and-conquer framework. Specifically, NPDC firstly decomposes the original objective function f into D sub-problems, each of which exclusively contains one decision variable. Then each sub-problem is solved in parallel. At the T-th iteration, the best-ever found partial solution b_j^T to each j-th sub-problem is merged to form the final solution b^T to the original problem f.

In order to keep NPDC compatible with the proposed meta-model, the optimization of each sub-problem should comply with two considerations. First, each sub-problem should be solved in a (1 + 1)-EA paradigm. Second, the offspring should be generated close to its parent, so that the historical comparison results can be helpful to the current comparison between them. Given this, the Gaussian mutation operator, as shown in Eq.(10), is well-suited to be the search operator in NPDC since it generates the offspring closer to the parent with higher probability.

$$\tilde{x}_j^t = b_j^{t-1} + \sigma_j^t \cdot \mathcal{N}(0, 1), \tag{10}$$

where σ_j^t indicates the search step-size, and $\mathcal{N}(0, 1)$ is a Gaussian random variable with zero mean and standard deviation 1. On the other hand, if we think from the perspective of the search effectiveness, the search operator should generate offspring quite different from the parent, so that the search

process can go faster and has higher probability to jump away from premature convergence. Such purpose can be realized by the Cauchy mutation operator [49], as shown in Eq.(11), since it has much 'wider' probability distribution than the Gaussian mutation.

$$\tilde{x}_j^t = b_j^{t-1} + \sigma_j^t \cdot \mathcal{C}(0,1), \tag{11}$$

where $\mathcal{C}(0,1)$ indicates a random variable subject to the standard Cauchy distribution.

In order to make a good balance between the effectiveness of search process and that of the meta-model, NPDC generates an offspring by applying either Eq.(10) or Eq.(11) uniformly at random. Generally, the search step-size σ_j^t in both Eq.(10) and Eq.(11) can be adapted during the search and may also vary over sub-problems. To make it simple, each σ_j^t is initialized as 1.00, and adjusted at each iteration, according to the 1/5 success rule again, as given in

$$\sigma_j^{t+1} = \sigma_j^t \cdot \exp^{\frac{1}{\sqrt{2}}}[\mathbb{I}_{\tilde{x}_j^t \neq b_j^{t-1}} \cdot (\mathbb{I}_{\Theta} - \frac{1}{5})]. \tag{12}$$

Eq.(12) says that, as long as $\tilde{x}_j^t \neq b_j^{t-1}$, σ_j^t will be adjusted. More specifically, if the quality of b_j^t is superior to the quality of $\tilde{x}_j^t$, σ_j^t will be enlarged with a factor of $\exp^{\frac{1}{\sqrt{2}}}(\frac{4}{5})$. Otherwise, σ_j^t will be reduced with a factor of $\exp^{\frac{1}{\sqrt{2}}}(-\frac{1}{5})$. The qualities of $\tilde{x}_j^t$ and b_j^{t-1} are compared using Θ in Eq.(9).

As a summary, the detailed pseudo-code of NPDC is presented in Algorithm 1. In order to generalize the description to the population-based EAs composed of multiple $(1+1)$-EA paradigms, the denotation of each b_j^t is modified to $b_{i,j}^t$, where $i = 1, 2, \ldots, \lambda$ denotes the i-th $(1+1)$-EA paradigm, or say the i-th individual. Other notations are also modified accordingly. NPDC initializes the optimization process at steps 1-5 and records the best complete solution at step 6. Then for each sub-problem, NPDC employs a sub-problem optimizer consisting with λ $(1+1)$-EA paradigm based search operators. The search operators generate offspring at steps 10-13 with the Gaussian mutation and Cauchy mutation. Other EAs with the $(1+1)$-EA paradigm, e.g., PSO and DE, can easily be incorporated into NPDC by replacing steps 10-13 with their specific search operators. The offspring and their parents are pairwise compared and selected to the next iteration at step 14 using the meta-model. The pre-selected best partial solutions in each sub-problem is merged and evaluated at step 15, where the original objective function f will be called once for a $(1+1)$-EA paradigm at one iteration. In other words, at each iteration of the whole search, NPDC consumes λ function evaluations at step 15. The meta-models and the search operators are adjusted at steps 16-19, based on the qualities of the merged solutions evaluated by f. The best complete solution of each $(1+1)$-EA paradigm is respectively updated at steps 20-24. When the whole optimization process is iterated for $\frac{T}{\lambda}$ times, i.e., NPDC consumes T function evaluations, the best complete solution among all merges of the λ $(1+1)$-EA paradigm is output at step 25.

Algorithm 1 NPDC(f, T, λ, n)

1: Divide f into D exclusive sub-problems.
2: **For** $i = 1$ **to** λ
3: **For** $j = 1$ **to** D
4: $PS_{i,j}^1 = 1.0$; $PL_{i,j}^1 = 1.0$; and $\sigma_{i,j}^t = 1.0$.
5: Initialize $x_{i,j}^1$ uniformly randomly; Let $b_{i,j}^0 = x_{i,j}^1$.
6: $FB_i = f([b_{i,1}^0, b_{i,2}^0, \ldots, b_{i,D}^0])$.
7: **For** $t = 1$ **to** $\frac{T}{\lambda}$
8: **For** $i = 1$ **to** λ
9: **For** $j = 1$ **to** D
10: **If** $r > 0.5$
11: $\tilde{x}_j^t = b_j^{t-1} + \sigma_j^t \cdot \mathcal{N}(0,1)$.
12: **Else**
13: $\tilde{x}_j^t = b_j^{t-1} + \sigma_j^t \cdot \mathcal{C}(0,1)$.
14: $b_j^t = g_j^t(b_j^{t-1}, \tilde{x}_j^t)$.
15: $\widetilde{FB}_i = f([b_{i,1}^t, b_{i,2}^t, \ldots, b_{i,D}^t])$.
16: **For** $j = 1$ **to** D
17: $PS_j^{t+1} = PS_j^t \cdot \exp^{\frac{1}{\sqrt{2}}}[\mathbb{I}_{\tilde{x}_j^t < b_j^{t-1}} \cdot (\mathbb{I}_{\widetilde{FB}_i < FB_i} - \frac{1}{5})]$.
18: $PL_j^{t+1} = PL_j^t \cdot \exp^{\frac{1}{\sqrt{2}}}[\mathbb{I}_{\tilde{x}_j^t > b_j^{t-1}} \cdot (\mathbb{I}_{\widetilde{FB}_i < FB_i} - \frac{1}{5})]$.
19: $\sigma_j^{t+1} = \sigma_j^t \cdot \exp^{\frac{1}{\sqrt{2}}}[\mathbb{I}_{\tilde{x}_j^t \neq b_j^{t-1}} \cdot (\mathbb{I}_{\widetilde{FB}_i < FB_i} - \frac{1}{5})]$.
20: **If** $\widetilde{FB}_i < FB_i$
21: $FB_i = \widetilde{FB}_i$.
22: **Else**
23: **For** $j = 1$ **to** D
24: $b_{i,j}^t = b_{i,j}^{t-1}$.
25: **Output** $min_{1 \leq i \leq \lambda}(FB_i)$.

C. THE PARALLELISM OF NPDC

As can be seen from Algorithm 1, NPDC is parallelizable on two levels. First, NPDC is fully parallelizable on the individual level, since each of the λ $(1+1)$-EA paradigms acts fully independently with others. Second, NPDC is parallelizable on the decision variables level. Specifically, only step 6 and step 15 require merging $b_{i,j}^t$ from each j-th sub-problem, while other steps can be processed independently. As discussed earlier, each $b_{i,j}^t$ required by step 6 and step 15 will not introduce any temporal interdependencies among sub-problems, since NPDC can pre-select $b_{i,j}^t$ by the meta-model of each sub-problem itself at step 14. Therefore, as long as the workload for solving each sub-problem is in balance, which is highly likely, each $b_{i,j}^t$ can be collected simultaneously from the sub-problems for function evaluations at step 6 and step 15.

Based on that, to implement NPDC in parallel, one can simply allocate the computational tasks of each sub-problem to a machine. If the number of available machines is smaller than D, one can either allocate an equal number of sub-problems to each machine or extend one iteration of Algorithm 1 to multi-round. When those slave machines have finished their jobs of sub-problem optimizers for one iteration, the data of $b_{i,j}^t$ will be transmitted to a master machine

for executing step 6 and step 15. Then the information of the corresponding function evaluations will be transmitted from the master machine back to each slave machine. In this regard, the only bottleneck of parallelizing NPDC is the computational efficiency of the calls of f at step 6 and step 15, which cannot be parallelized unless the explicit formula of f can be known and mathematically decomposable.

Specifically, the parallelism of NPDC can be quantified in terms of the speed-up, which is a commonly used indicator for measuring parallel efficiency [50]. It measures the ratio of T_1 over T_N, i.e., the elapsed computational time of a task executed on 1 and N processors. Let us denote the computational time costed by the T function evaluations as T_{FE}, then given the above discussions we have $T_1 - T_{FE} = N \cdot (T_N - T_{FE})$. According to Amdahl's law, the speed-up of NPDC is ideally given as Eq.(13).

$$\frac{T_1}{T_N} = \frac{T_1}{T_{FE} + (T_N - T_{FE})} = \frac{T_1}{T_{FE} + \frac{1}{N} \cdot (T_1 - T_{FE})} = \frac{N}{1 + \frac{T_{FE}}{T_1} \cdot (N - 1)}. \tag{13}$$

If $\frac{T_{FE}}{T_1} \to 0$, the speed-up ratio converges to N, which means NPDC enjoys a linear speed-up that the total computational time can be reduced by N times if N processors are used. On the other hand, if $\frac{T_{FE}}{T_1} \to 1$, the speed-up ratio converges to 1, which means that parallelization does not make any sense to NPDC. Also note that, although we cannot distribute the computational task of each function evaluation to multiple processors, the burden of T_{FE} can still be alleviated by distributing multiple function evaluations to multiple processors if a population consists of multiple individuals [51].

V. EMPIRICAL STUDIES

This paper raises four questions. First, how differently the existing serial DC-based works behave from their parallel counterparts, in terms of the final solution quality? Second, does NPDC perform better than the existing DC-based works, in terms of the final solution quality? Third, how is the parallelism of NPDC in practice? Lastly, how does the proposed meta-model impact on the performance of NPDC. In the empirical studies, four groups of comparisons are conducted to answer these questions, respectively.

A. EXPERIMENT PROTOCOL

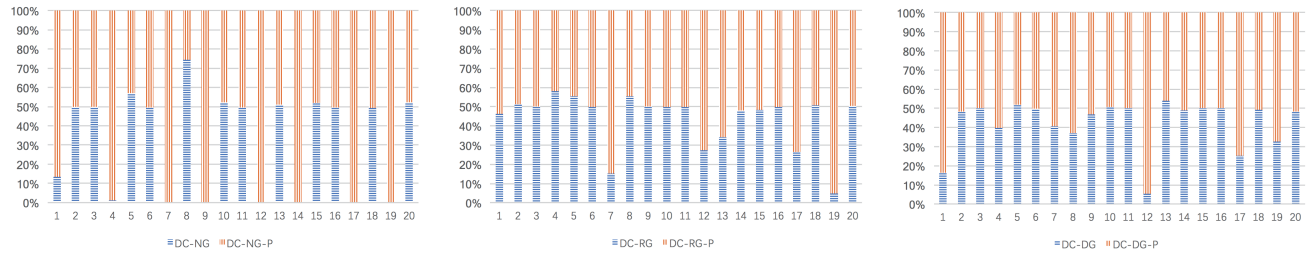
The CEC'2010 large-scale optimization benchmark covers different degrees of separability and multi-modality, which are the main difficulties for large-scale optimization problems [52], and thus has been commonly used to verify the performance of large-scale optimization algorithms [12], [28], [53]. Given this, it is also adopted as the test suite in the empirical studies.

All the 20 problems in the test suite involve 1000 decision variables, i.e., $D = 1000$. The time budget, i.e., the total number of function evaluations of each run, is set to 3e6, which is widely used in the literature [12], [28], [53]. Each run of an algorithm terminates when the time budget runs out. The function error is used to measure the quality of the final solution, i.e., the difference between the objective function value of the final output solution and that of the optimal solution to the problem (which are known for all tested problems as 0.0). In a word, the smaller the function error is, the better the quality of the final solution will be. All the compared algorithms are repeated on each problem for 20 runs. The function errors of the corresponding solutions are recorded and averaged over 20 runs to represent the performance of algorithms on a problem. The two-sided Wilcoxon rank-sum test at a 0.05 significance level is also conducted to see whether the performances of pairwise compared algorithms are statistically significantly different. All experiments are conducted on a workstation with 2 CPUs specified with Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10 GHz (in total 16 physical cores), 128 GB memory size, 256 GB SSD, and Ubuntu 16.04 LTS.

B. ALGORITHM SETTINGS

For the compared algorithms, three representatives of the existing DC-based EAs are tested, together with their parallelized counterparts. We denote these three DC-based EAs as DC-NG, DC-RG and DC-DG, and their parallelized counterparts as DC-NG-P, DC-RG-P and DC-DG-P. To be specific, DC-NG is short for divide-and-conquer with natural grouping, which naively decomposes the D-dimensional problem into D 1-dimensional sub-problems. DC-RG follows the random grouping strategy proposed in [11] that decomposes the decision variables into 10 sub-problems in an on-line manner, each of which exclusively consists of 100 decision variables. The on-line random grouping happens every iteration as [26] proved that performing random grouping frequently could minimize the interdependencies among sub-problems. DC-DG decomposes the problem using the improved differential grouping strategy, i.e., DG2 [53], which actively analyzes the interdependencies among decision variables so that the decomposition can be made much more accurate. The only difference between the above three compared algorithms and their corresponding parallelized counterparts is that the former builds the objective function for sub-problems using Eq.(2), while the latter builds that using Eq.(3). The decomposition phases of the above algorithms are parameterless.

To make the comparisons fair enough, all the compared algorithms employ the same sub-problem optimizer with NPDC. That is, in each sub-problem, an offspring is generated based on a parent using Eq.(10) and Eq.(11). The search step-size is adjusted with Eq.(12). For DC-NG and DC-NG-P, as the sub-problems are all 1-dimensional, each partial solution is actually a scalar, which is the same case with NPDC. For the rest compared algorithms, each partial solution is a vector, while the search step-size is still a scalar for each sub-problem. When applying Eq.(10) and Eq.(11) to a vector partial solution, the vector is actually generated one dimension at a time using the same search step-size, until all dimensions in the sub-problem have been gone through.



(a) Comparisons of DC-NG and DC-NG-P (b) Comparisons of DC-RG and DC-RG-P (c) Comparisons of DC-DG and DC-DG-P

FIGURE 2. The pairwise comparisons of the existing DC-based EAs and their parallel counterparts on 20 tested problems, in terms of the final solution quality. For each bar, it shows the sum of the final averaged solution qualities obtained by an existing DC-based EA and its parallel counterpart, and it is normalized to 100%. The contribution of either algorithm made to the whole bar is depicted in orange or blue. As the minimization problem is considered, the better algorithm between the pairwise comparison will be with short bar, and the larger the divergence is, the more differently they perform.

In order to clearly see how the different ways of defining objective functions for sub-problems will influence the performance of the tested algorithms, we let $\lambda = 1$ for all tested algorithms. As mentioned earlier, the CEC'2010 test suite involves two characteristics, i.e., multi-modality and separability. Between them, the different degrees of separability will largely influence the evaluation of partial solutions, which highly correlates with defining f_j^t. Oppositely, the multi-modality mainly concerns how the sub-problem optimizers can explore the solution space more effectively, which has nothing to do with defining f_j^t. Thus, it would be better not to enhance the exploration ability of the tested algorithms. Otherwise, it is difficult to tell whether a good qualitied solution is produced by effective objective functions for sub-problems or by an effective sub-problem optimizer. By letting $\lambda = 1$, each sub-problem optimizer is merely an individual-based hill-climber, whose exploration ability has been minimized, and the impacts of defining objective functions for sub-problems on the final solution quality can be emphasized.

C. INVESTIGATIONS ON THE PARALLELISM OF EXISTING DC-BASED EAs

In this part, the above-mentioned three representatives of existing DC-based EAs, i.e., DC-NG, DC-RG, and DC-DG, are compared with their parallel counterparts, i.e., DC-NG-P, DC-RG-P, and DC-DG-P, respectively. The comparisons aim to show that, to parallelize existing DC-based EAs, the final solution quality may be degenerated. The empirical results on all 20 tested problems are shown in Figs.2(a)-(c), respectively for each pair of algorithms. In each figure, there are 20 bars representing the comparisons of each existing DC-based EAs against its parallel counterparts on 20 different tested problems. For each bar, the final solution qualities of a pair of two algorithms averaged over 20 problems are shown, where the blue bar represents the performance of the existing DC-based EA and the orange bar represents its parallel counterpart. As all the 20 problems are minimization problems, the shorter the bar is, the better the corresponding algorithm performs.

There are two observations can be made from Figs.2(a)-(c). First, for the three pairs of algorithms, the existing DC-based EAs all perform generally better than their parallel counterparts. On some problems, the superiority can be very significantly (see #12 in all three figures for example), while on others it is only slightly better (see #6 in Fig.2(c)). However, note that, sometimes 1% better could be dominant, which heavily depends on the actual scale of fitness values. Second, the more accurate the decomposition is, the closer the performances between the existing algorithm and its parallel counterpart is. This is because the interdependencies among sub-problems will amplify the inconsistency of the objective functions between the sub-problems and the original problem [34]. That is, the more interdependent two sub-problems are, their objective functions are more likely to be inconsistent with f. Nevertheless, even the DG2 decomposition strategy has empirically shown the perfect decomposition, i.e., no interdependencies among sub-problems, on most problems in CEC'2010 test suite [53], the divergences between DC-DG and DC-DG-P on some problems are still significant. This phenomenon again stresses that the existing DC-based EAs are difficult to parallelize, unless some compromises on the final solution qualities are made.

D. INVESTIGATIONS ON THE EFFECTIVENESS OF NPDC

Indeed, NPDC is proposed for its feature of being naturally parallelizable, while its effectiveness of reducing the search iterations or producing high-quality final solution is also important. Because if NPDC performs even worse than the parallelized compared algorithms, one should directly use them to solve problems rather than NPDC. In order to verify the effectiveness of NPDC, it is compared with DC-NG, DC-RG, and DC-DG on all 20 problems for 20 runs. The averaged final solution of each algorithm is shown in Table 1, together with the standard deviation. For each problem, the best averaged solution quality is marked in gray for emphasis. The statistical test is given in the last row of Table 1, where 'w', 'd', and 'l' indicate the number of problems that NPDC performs statistically better, the same, and worse than the compared algorithm. Furthermore, the best intermediate solution

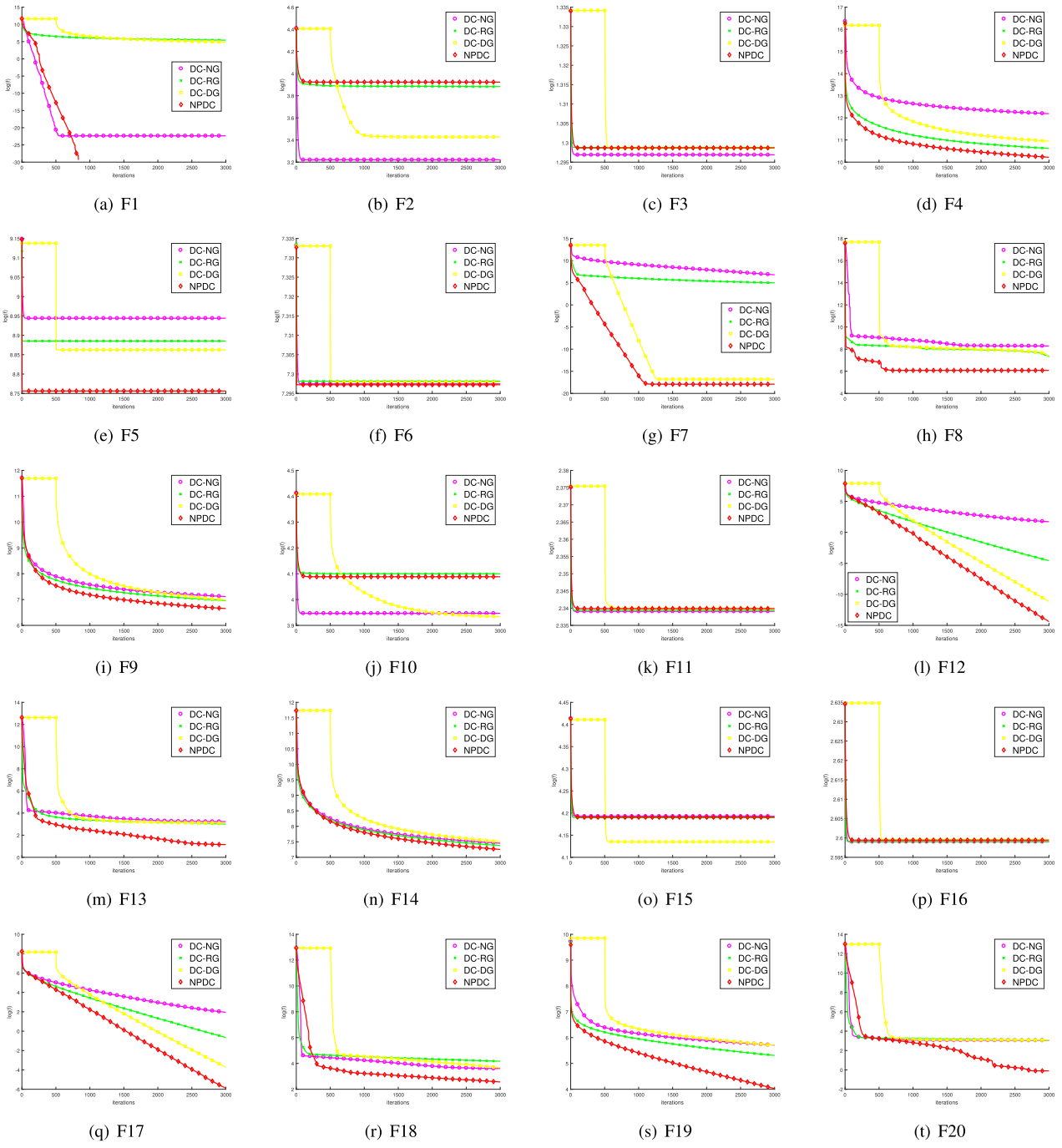
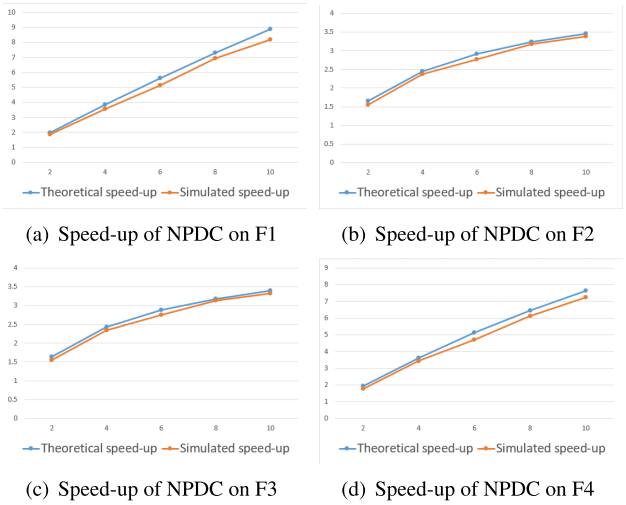


FIGURE 3. The convergence rate of NPDC and three existing DC-based EAs. The Y-axis denotes the $\log_{10} f$, which means the order of magnitude of the quality of final solution. The X-axis indicates the function evaluations consumed during the optimization.

qualities during the optimization of each algorithm are also recorded and averaged, which are depicted in Figs.3(a)-(t) to show the convergence rate of the compared algorithms. In these figures, the X-axis represents the number of function evaluations consumed by the algorithms, while the Y-axis describes the averaged solution quality at a $\log_{10}$ level. Hence, the Y-axis of the figures actually denotes the orders of magnitude of the final solution quality. On this basis, the lower the curve is, the faster the corresponding algorithm converges.

A quick conclusion that can be drawn from Table 1 and Figs.3(a)-(t) is that, NPDC successfully outperforms the three compared algorithms on the majority of 20 tested problems. Most importantly, even DC-DG has perfectly decomposed the sub-problems, it is still statistically inferior to NPDC, which just naively decomposes the problem into 1-dimensional



(a) Speed-up of NPDC on F1 (b) Speed-up of NPDC on F2

(c) Speed-up of NPDC on F3 (d) Speed-up of NPDC on F4

FIGURE 4. The comparisons between theoretical speed-up and simulated speed-up of NPDC. The Y-axis denotes the speed-up of NPDC. The X-axis indicates the cores used for such run.

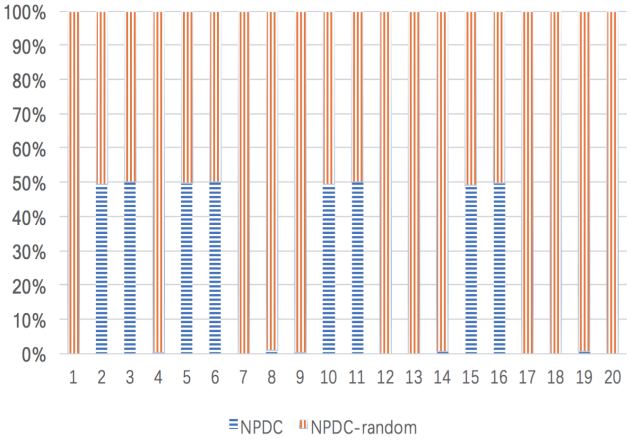


FIGURE 5. The pairwise comparisons between NPDC and NPDC-random on 20 problems, in terms of the final solution quality.

sub-problems. This actually benefits from two aspects. First, NPDC evaluates the partial solutions consistent with the original problem, where a good partial solution to each sub-problem is definitely a good component to f. Second, existing DC-based EAs consume M function evaluations in one iteration, each of which is used to evaluate the offspring partial solution in the i-th sub-problem using f_i^t, $i = 1, 2, \ldots, M$. On the other hand, NPDC only consumes 1 function evaluation in one iteration, which gains M times more iterations for optimization. Thus, NPDC is shown to be a competitive DC-based EA for large-scale optimization problems that can output very good-qualitied solution.

E. INVESTIGATIONS ON THE PARALLELISM OF NPDC

To measure the parallelism of NPDC, we run NPDC on the 20 problems with 1, 2, 4, 6, 8 and 10 CPU cores,

TABLE 1. The average and standard deviation of function errors on the 1000-dimensional CEC'2010 large-scale global optimization benchmark.

Algo.		DC-NG	DC-RG	DC-DG	NPDC
F_1	Mean	4.30e-23	2.65e+05	7.65e+04	0.00e+00
	Std	4.72e-23	5.56e+04	1.92e+04	0.00e+00
F_2	Mean	1.67e+03	7.63e+03	2.67e+03	8.38e+03
	Std	1.61e+02	6.20e+02	4.57e+02	3.69e+02
F_3	Mean	1.98e+01	1.99e+01	1.99e+01	1.99e+01
	Std	1.91e-02	1.53e-02	1.65e-02	1.29e-02
F_4	Mean	1.55e+12	4.16e+10	8.82e+10	1.66e+10
	Std	7.30e+11	1.56e+10	2.46e+10	9.79e+09
F_5	Mean	8.79e+08	7.68e+08	7.28e+08	5.71e+08
	Std	1.60e+08	1.05e+08	1.14e+08	1.54e+08
F_6	Mean	1.98e+07	1.98e+07	1.98e+07	1.98e+07
	Std	9.39e+04	6.17e+04	8.96e+04	7.16e+04
F_7	Mean	6.32e+06	9.64e+04	1.64e-17	1.22e-18
	Std	5.05e+06	1.86e+04	9.42e-18	1.98e-19
F_8	Mean	1.95e+08	2.01e+07	2.00e+07	1.20e+06
	Std	2.65e+08	3.25e+07	4.70e+07	1.87e+06
F_9	Mean	1.30e+07	9.17e+06	9.72e+06	4.48e+06
	Std	1.30e+06	7.99e+05	8.35e+05	5.12e+05
F_{10}	Mean	8.84e+03	1.26e+04	8.60e+03	1.23e+04
	Std	4.49e+02	6.12e+02	4.80e+02	4.11e+02
F_{11}	Mean	2.18e+02	2.18e+02	2.19e+02	2.19e+02
	Std	2.43e-01	2.72e-01	2.28e-01	2.49e-01
F_{12}	Mean	5.23e+01	2.73e-05	7.14e-12	3.89e-15
	Std	3.91e+01	2.35e-05	2.75e-12	2.78e-15
F_{13}	Mean	1.69e+03	1.01e+03	1.23e+03	1.40e+01
	Std	7.69e+02	8.58e+02	6.93e+02	1.38e+01
F_{14}	Mean	2.84e+07	2.41e+07	3.19e+07	1.79e+07
	Std	1.50e+06	1.79e+06	2.75e+06	1.35e+06
F_{15}	Mean	1.56e+04	1.55e+04	1.36e+04	1.55e+04
	Std	4.66e+02	4.61e+02	7.00e+02	4.54e+02
F_{16}	Mean	3.97e+02	3.97e+02	3.98e+02	3.98e+02
	Std	2.57e-01	3.21e-01	4.29e-01	3.94e-01
F_{17}	Mean	8.90e+01	2.19e-01	1.87e-04	1.24e-06
	Std	6.97e+01	4.57e-02	3.35e-05	7.56e-07
F_{18}	Mean	4.02e+03	1.49e+04	4.85e+03	3.67e+02
	Std	1.16e+03	6.65e+03	2.63e+03	2.20e+02
F_{19}	Mean	5.12e+05	2.04e+05	5.05e+05	1.07e+04
	Std	2.98e+04	9.35e+03	2.63e+04	1.25e+03
F_{20}	Mean	1.13e+03	1.20e+03	1.19e+03	7.97e-01
	Std	1.93e+02	1.55e+02	1.07e+02	1.64e+00
w-d-l		13-2-5	13-4-3	13-4-3	-

respectively. Three kinds of data are recorded for each run, i.e., the simulated computational time $\widehat{T_1}$ of NPDC on single-core, the simulated computational time $\widehat{T_N}$ of NPDC on N-cores, and the simulated computational time $\widehat{T_{FE}}$ for function evaluations. According to Eq.(13), we can calculate two kinds of speed-up ratios. The first is the theoretical speed-up ratio that all the components of NPDC, except the function evaluations, are assumed to be fully parallelized, calculated as $\frac{N}{1+\frac{\widehat{T_{FE}}}{\widehat{T_1}}\cdot(N-1)}$. The second is the simulated speed-up ratio that directly compares the simulated computational time of NPDC on single-core and N-cores, calculated as $\frac{\widehat{T_1}}{\widehat{T_N}}$. In general, the closer the latter to the former, the better the parallelism of NPDC will be. We only depict the results on the first 4 tested problems in Figs.4 (a)-(d) as illustrations.

It can be clearly seen that, the simulated speed-up of NPDC remains very close to its theoretical speed-up on the 4 problems. Actually, this also happens to the rest problems, which is omitted here due to the limitation of the pages. The results verify that NPDC is naturally parallelizable. We also noticed that for the 6-cores case, the gap between those two speed-up becomes a little bit larger. The reason might be that, for the tested problems, their 1000 decision variables cannot be equally assigned to 6 cores, where not all the current best partial solution can be collected simultaneously from each sub-problem for function evaluations. If we look at the Y-axis of the four figures, both speed-up ratios drop

down dramatically on problems 2 and 3. This is because the computational costs for function evaluations of these two problems are relatively high, which are above 20% of the total computational time as we counted (61.3s out of 284.4s). This corresponds to the discussions in section IV.C that the function evaluation phase is the main bottleneck for the parallel efficiency of NPDC. Unfortunately, such bottleneck exists for all DC-based EAs due to the black-box optimization nature, and thus cannot be addressed systematically.

F. INVESTIGATIONS ON THE META-MODEL OF NPDC

The meta-model plays an important role to the performance of NPDC. In order to show how the accuracy of the meta-model can benefit NPDC, we specially designed a new compared algorithm, named NPDC-random. In NPDC-random, the two parameters of the meta-model, i.e., PL_j^t and PS_j^t, are fixed to 0.5 during the whole optimization. This means that, given an offspring and its parent, the meta-model in NPDC-random simply gives a random guess on which one is better. Except for this, the rest sub-routines of NPDC-random are kept the same to NPDC. Both algorithms are compared on the 20 problems and the averaged final solution qualities over 20 runs are shown in Fig.5. On problems 1, 4, 7, 8, 9, 12, 13, 14, 17, 18, 19, 20, NPDC outperforms significantly than NPDC-random. For the rest problems, NPDC only shows slightly advantages over NPDC-random. The reason might be that the landscapes of those problem are not smooth at all, where the assumption of the proposed meta-model does not hold anymore and the meta-model becomes less effective.

VI. CONCLUSION AND DISCUSSIONS

This paper investigated the parallelism of DC-based EAs on large-scale optimization black-box problems. We first discussed that the existing DC-based EAs could not be directly parallelized. Otherwise, their effectiveness in terms of final solution quality could be severely degenerated. The reason behind this phenomenon was also analyzed in details. To be specific, one most important thing for DC-based EAs is to build the objective functions for sub-problems. In existing works, the objective functions for sub-problems were inconsistent with the objective function for the original problem. In other words, a partial solution that is good to a sub-problem may not be a component of a good complete solution to the original problem. We revealed an ideal way that could build objective functions for sub-problems consistent with the original problem. Then the ways for defining objective functions used by existing DC-based EAs were compared to the ideal way. It shows that the parallelizable way of existing DC-based EAs can be probabilistically much worse than the non-parallelizable way. This analysis was also supported by empirical studies. We then designed a novel way based on the ideal way, which is naturally parallelizable and effective. The resultant algorithm, i.e., NPDC, was detailed. Its effectiveness in terms of the final solution quality was empirically verified on the CEC'2010 large-scale optimization benchmark, against several representatives of existing DC-based EAs. We also empirically showed that NPDC can be efficiently parallelized. Besides, the limitations of both NPDC and its meta-model are also discussed in this paper.

It should be noted that the framework of NPDC looks similar to our previous work [10] that also divides the original problem into D sub-problems and employs an meta-model in each sub-problem. However, this work is actually very different from the work [10] for two reasons. First, these two works have different motivations and contributions. For [10], it aims to solve the difficulty of accurately complementing a partial solution, which is in turn addressed as an expensive optimization problem and the meta-model is thus used for cheaper computational costs. However, for this work, it aims to address the parallelism difficulty of existing DC-based EAs, where the meta-model is used to break the dining phylosophers dilemma. From this perspective, these two works are proposed for completely different problems but share the same framework. Second, from the perspective of implementations under the framework, these two algorithms are still very different. In [10], each sub-problem employs a $(1 + \lambda)$-EA paradigm where one parent generates λ offspring. Among them, a half of the λ offspring are generated with the Gaussian mutation operator, and half of them are generated with the Cauchy mutation operator. At the end of each iteration, the parent will be compared with each offspring one-by-one for survival. However, in this work, each sub-problem only employs a $(1 + 1)$-EA, making the algorithm much simpler and more compatible with other well-established EAs, e.g., PSO and DE. On this basis, these two works are inherently different from each other, and thus we did not discuss the details of [10] in the previous sections.

In NPDC, the sub-problem optimizer can be a population-based EA following the $(1 + 1)$-EA paradigm. Specially, in this paper, we let it be λ independent individual-based hill-climbers. In the future, it is expected that the NPDC could be incorporated with other well-established $(1 + 1)$-EA paradigm based EAs, e.g., DE and PSO, to enhance its ability of solving complex multi-modal optimization problems, while maintaining the feature of natural parallelism to a good extent. Besides, NPDC currently employs the Gaussian mutation and Cauchy mutation operators in the sub-problem optimizers, making it ill-equipped to deal with combinatorial optimization problems. This issue will also be investigated for future work.

REFERENCES

[1] A. Azadegan and J. Jayaram, "Resiliency in supply chain systems: A triadic framework using family resilience model," in *Supply Chain Risk Management*. Singapore: Springer, 2018, pp. 269–288.
[2] F. Mei, Q. Cao, H. Jiang, and L. Tian, "LSM-tree managed storage for large-scale key-value store," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 2, pp. 400–414, Feb. 2019.
[3] Y. Cui, S. Peng, Y. Lu, X. Zhu, B. Wang, C. Wu, and X. Liao, "mSNP: A massively parallel algorithm for large-scale SNP detection," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 11, pp. 2557–2567, Nov. 2018.

[4] S. Santander-Jiménez and M. A. Vega-Rodríguez, "Parallel multiobjective metaheuristics for inferring phylogenies on multicore clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 6, pp. 1678–1692, Jun. 2015.

[5] 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.

[6] A. Gulati, A. Holler, M. Ji, G. Shanmuganathan, C. Waldspurger, and X. Zhu, "Vmware distributed resource management: Design, implementation, and lessons learned," *VMware Tech. J.*, vol. 1, no. 1, pp. 45–64, 2012.

[7] Z.-H. Zhan, X.-F. Liu, H. Zhang, Z. Yu, J. Weng, Y. Li, T. Gu, and J. Zhang, "Cloudde: A heterogeneous differential evolution algorithm and its distributed cloud version," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 704–716, Mar. 2017.

[8] Y.-J. Gong, W.-N. Chen, Z.-H. Zhan, J. Zhang, Y. Li, Q. Zhang, and J.-J. Li, "Distributed evolutionary algorithms and their models: A survey of the state-of-the-art," *Appl. Soft Comput.*, vol. 34, pp. 286–300, Sep. 2015.

[9] 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.

[10] P. Yang, K. Tang, and X. Yao, "Turning high-dimensional optimization into computationally expensive optimization," *IEEE Trans. Evol. Comput.*, vol. 22, no. 1, pp. 143–156, Feb. 2018.

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

[12] 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.

[13] 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, pp. 63:1–63:33, Jul. 2015.

[14] Z. Zhang, C. Li, Y. Tao, R. Yang, H. Tang, and J. Xu, "Fuxi: A fault-tolerant resource management and job scheduling system at Internet scale," *Proc. VLDB Endowment*, vol. 7, no. 13, pp. 1393–1404, 2014.

[15] D. Molina, M. Lozano, and F. Herrera, "MA-SW-chains: Memetic algorithm based on local search chains for large scale continuous global optimization," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2010, pp. 1–8.

[16] A. LaTorre, S. Muelas, and J.-M. Peña, "Large scale global optimization: Experimental results with MOS-based hybrid algorithms," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2013, pp. 2742–2749.

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

[18] Y. Mei, M. N. Omidvar, X. Li, and X. Yao, "A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization," *ACM Trans. Math. Softw.*, vol. 42, no. 2, pp. 13:1–13:24, 2016.

[19] Y. Liu, X. Yao, Q. Zhao, and T. Higuchi, "Scaling up fast evolutionary programming with cooperative coevolution," in *Proc. Congr. Evol. Comput.*, vol. 2, May 2001, pp. 1101–1108.

[20] K. Tang, J. Wang, X. Li, and X. Yao, "A scalable approach to capacitated arc routing problems based on hierarchical decomposition," *IEEE Trans. Cybern.*, vol. 47, no. 11, pp. 3928–3940, Nov. 2017.

[21] A. Kabán, J. Bootkrajang, and R. J. Durrant, "Toward large-scale continuous EDA: A random matrix theory perspective," *Evol. Comput.*, vol. 24, no. 2, pp. 255–291, 2016.

[22] H. Qian and Y. Yu, "Scaling simultaneous optimistic optimization for high-dimensional non-convex functions with low effective dimensions," in *Proc. 13th AAAI Conf. Artif. Intell.* Phoenix, AZ USA: AAAI, Feb. 2016, pp. 2000–2006.

[23] E. Alba and J. M. Troya, "A survey of parallel distributed genetic algorithms," *Complexity*, vol. 4, no. 4, pp. 31–52, 1999.

[24] 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.

[25] 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.

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

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

[28] W. Chen, T. Weise, Z. Yang, and K. Tang, "Large-scale global optimization using cooperative coevolution with variable interaction learning," in *Proc. Int. Conf. Parallel Problem Solving Nature*. Kraków, Poland: Springer, Sep. 2010, pp. 300–309.

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

[30] X. Lu, S. Menzel, K. Tang, and X. Yao, "The performance effects of interaction frequency in parallel cooperative coevolution," in *Proc. Asia–Pacific Conf. Simulated Evol. Learn.* Dunedin, New Zealand: Springer, Dec. 2014, pp. 82–93.

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

[32] P. Yang, K. Tang, J. A. Lozano, and X. Cao, "Path planning for single unmanned aerial vehicle by separately evolving waypoints," *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1130–1146, Oct. 2015.

[33] T. Jansen and R. P. Wiegand, "The cooperative coevolutionary (1+1) EA," *Evol. Comput.*, vol. 12, no. 4, pp. 405–434, 2004.

[34] L. Panait, "Theoretical convergence guarantees for cooperative coevolutionary algorithms," *Evol. Comput.*, vol. 18, no. 4, pp. 581–615, 2010.

[35] L. Panait, S. Luke, and J. F. Harrison, "Archive-based cooperative coevolutionary algorithms," in *Proc. 8th Annu. Conf. Genetic Evol. Comput.*, Seattle, WA, USA, Jul. 2006, pp. 345–352.

[36] L. Panait and S. Luke, "Time-dependent collaboration schemes for cooperative coevolutionary algorithms," in *Proc. AAAI Fall Symp. Coevol. Coadaptive Syst.*, Richmond, VA, USA, Nov. 2005, pp. 18–25.

[37] R.-L. Tang, Z. Wu, and Y.-J. Fang, "Adaptive multi-context cooperatively coevolving particle swarm optimization for large-scale problems," *Soft Comput.*, vol. 21, no. 16, pp. 4735–4754, 2014.

[38] D. Xiao and A.-H. Tan, "Self-organizing neural architectures and cooperative learning in a multiagent environment," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 37, no. 6, pp. 1567–1580, Dec. 2007.

[39] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Proc. Int. Conf. Parallel Problem Solving Nature*. Jerusalem, Israel: Springer, Oct. 1994, pp. 249–257.

[40] Y.-H. Jia, W.-N. Chen, T. Gu, H. Zhang, H.-Q. Yuan, S. Kwong, and J. Zhang, "Distributed cooperative co-evolution with adaptive computing resource allocation for large scale optimization," *IEEE Trans. Evol. Comput.*, vol. 23, no. 2, pp. 188–202, Apr. 2019.

[41] C. Qian, C. Bian, W. Jiang, and K. Tang, "Running time analysis of the (1+1)-EA for onemax and leadingones under bit-wise noise," in *Proc. Genetic Evol. Comput. Conf.*, 2017, pp. 1399–1406.

[42] K. Tang, P. Yang, and X. Yao, "Negatively correlated search," *IEEE J. Sel. Area. Commun.*, vol. 34, no. 3, pp. 542–550, Mar. 2016.

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

[44] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proc. IEEE Int. Conf. Evol. Comput., IEEE World Congr. Comput. Intell.*, Anchorage, AK, USA, May 1998, pp. 69–73.

[45] P. Yang, K. Tang, and X. Lu, "Improving estimation of distribution algorithm on multimodal problems by detecting promising areas," *IEEE Trans. Cybern.*, vol. 45, no. 8, pp. 1438–1449, Aug. 2015.

[46] B. Liu, Q. Zhang, and G. G. E. Gielen, "A Gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems," *IEEE Trans. Evol. Comput.*, vol. 18, no. 2, pp. 180–192, Apr. 2014.

[47] Y. Jin, "Surrogate-assisted evolutionary computation: Recent advances and future challenges," *Swarm Evol. Comput.*, vol. 1, no. 2, pp. 61–70, Jun. 2011.

[48] N. Hansen, D. V. Arnold, and A. Auger, "Evolution strategies," in *Springer Handbook of Computational Intelligence*. Berlin, Germany: Springer, 2015, pp. 871–898.

[49] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 82–102, Jul. 1999.

[50] S. Vargas-Pérez and F. Saeed, "A hybrid MPI-openMP strategy to speedup the compression of big next-generation sequencing datasets," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2760–2769, Oct. 2017.

[51] M. Dubreuil, C. Gagne, 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.

[52] K. Tang, X. Li, P. N. Suganthan, Z. Yang, and T. Weise, "Benchmark functions for the CEC'2010 special session and competition on large scale global optimization," Nature Inspired Comput. Appl. Lab., Nanyang Technol. Univ., USTC, Hefei, China, Tech. Rep., 2009.
[53] M. N. Omidvar, M. Yang, Y. Mei, X. Li, and X. Yao, "DG2: A faster and more accurate differential grouping for large-scale black-box optimization," *IEEE Trans. Evol. Comput.*, vol. 21, no. 6, pp. 929–942, Dec. 2017.

PENG YANG (S'14–M'18) received the B.Eng. and Ph.D. degrees in computer science and technology from the University of Science and Technology of China (USTC), Hefei, China, in 2012 and 2017, respectively. From 2017 to 2018, he was with Huawei Company Ltd., as a Senior Engineer. He is currently a Research Assistant Professor with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China. His research interests include distributed computational intelligence and smart logistics. He has been a Regular Reviewer of several world-class journals and a Program Committee Member of a set of top international conferences.

KE TANG (S'05–M'07–SM'13) received the B.Eng. degree from the Huazhong University of Science and Technology, Wuhan, China, in 2002, and the Ph.D. degree from Nanyang Technological University, Singapore, in 2007.

From 2007 to 2017, he was with the School of Computer Science and Technology, University of Science and Technology of China (USTC), Hefei, China, as an Associate Professor, from 2007 to 2011, and a Professor, from 2011 to 2017. He is currently a Professor with the Department of Computer Science and Engineering, Southern University of Science and Technology (SUSTech), Shenzhen, China. His major research interests include machine learning, evolutionary computation, and their real-world applications. He has published more than 60 journal articles and more than 70 conference articles. According to Google Scholar, his publications have received more than 7000 citations and the H-index is 37 (as of August 2019). He was a member of editorial boards for a few other journals. He was a recipient of the Royal Society Newton Advanced Fellowship, in 2015, and the 2018 IEEE Computational Intelligence Society Outstanding Early Career Award. He is also an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION.

XIN YAO (F'03) is currently the Chair Professor of computer science with the Southern University of Science and Technology, Shenzhen, China, and a Professor of computer science with the University of Birmingham, U.K. He is also a Distinguished Lecturer of the IEEE Computational Intelligence Society (CIS). He has been involved in multi-objective optimization, since 2003, when he published a well-cited EMO'03 article on many objective optimization. His major research interests include evolutionary computation, ensemble learning, and their applications in software engineering. He was a recipient of the 2001 IEEE Donald G. Fink Prize Paper Award, in 2010 and 2016, the 2017 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Awards, the 2010 BT Gordon Radley Award for Best Author of Innovation (Finalist), the 2011 IEEE TRANSACTIONS ON NEURAL NETWORKS Outstanding Paper Award, and many other best paper awards. He was a recipient of the prestigious Royal Society Wolfson Research Merit Award, in 2012, and the IEEE CIS Evolutionary Computation Pioneer Award, in 2013. He was the President of the IEEE CIS, from 2014 to 2015, and the Editor-in-Chief of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, from 2003 to 2008.