Complex & Intelligent Systems (2023) 9:4403–4423
https://doi.org/10.1007/s40747-022-00937-w

ORIGINAL ARTICLE

DBCC2: an improved difficulty-based cooperative co-evolution for many-modal optimization

Yingying Qiao[2] · Wenjian Luo[1] · Xin Lin[2] · Peilan Xu[2] · Mike Preuss[3]

Received: 27 February 2022 / Accepted: 5 September 2022 / Published online: 13 January 2023

Abstract
Evolutionary multimodal optimization algorithms aim to provide multiple solutions simultaneously. Many studies have been conducted to design effective evolutionary algorithms for solving multimodal optimization problems. However, optimization problems with many global and acceptable local optima have not received much attention. This type of problem is undoubtedly challenging. In this study, we focus on problems with many optima, the so-called many-modal optimization problems, and this study is an extension of our previous conference work. First, a test suite including additively nonseparable many-modal optimization problems and partially additively separable many-modal optimization problems is designed. Second, an improved difficulty-based cooperative co-evolution algorithm (DBCC2) is proposed, which dynamically estimates the difficulties of subproblems and allocates the computational resources during the search. Experimental results show that DBCC2 has competitive performance.

Keywords Many-modal optimization · Evolutionary multimodal optimization · Cooperative co-evolution · Difficulty-based cooperative co-evolution

Introduction

The multimodal optimization problem (MMOP) is a type of problem that has multiple global optima (sometimes including acceptable local optima on request from the decision maker). Generally, multimodal optimization (MMO) has two goals: (1) to learn the problem landscape and the distribution of the optima and (2) to provide more alternatives for a decision maker to choose from when the current solution is unsatisfactory. Seeking multiple solutions can help the decision maker to change from the existing solution to another one immediately [26]. Under these circumstances, an efficient MMO algorithm is of great importance [7].

Evolutionary algorithms (EAs) and swarm intelligence (SI) are popular algorithms for solving MMOPs. However, both typical EAs and SI have a tendency to converge to a single optimum, which is problematic for MMOPs [51]. Therefore, many widespread niching methods, such as crowding [8,39], speciation [22,47,70], and fitness sharing [14], have been proposed to maintain the diversity of solutions. EAs and SI embedded with niching methods are common efficient algorithms for solving MMOPs. For example, CrowdingDE [59] improves the performance of differential evolution (DE) using a crowding scheme. Species-based DE [23] adopts an efficient niching method for maintaining diversity, where a population is divided into multiple species and DE is performed within each species. However, the radius of the species is a sensitive parameter, which should be set carefully.

This work is supported by National Natural Science Foundation of China (No. 61573327).

✉ Wenjian Luo
luowenjian@hit.edu.cn

Yingying Qiao
bonsur@mail.ustc.edu.cn

Xin Lin
iskcal@mail.ustc.edu.cn

Peilan Xu
xpl@mail.ustc.edu.cn

Mike Preuss
m.preuss@liacs.leidenuniv.nl

1 School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen 518055, Guangdong, China

2 School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, Anhui, China

3 Leiden Institute of Advanced Computer Science (LIACS) at Leiden University, Leiden, The Netherlands

NSDE [54] adopts a mutation strategy based on neighborhood. Therein, the neighborhood of an individual consists of m closest individuals using the Euclidean distance, where m is a manually fixed parameter. In [29], Lin et al. proposed a variant of DE (FBK-DE). FBK-DE adopts three strategies including species formulation, species balance, and keypoint-based mutation to meet the convergence and diversity requirements. In [71], DE based on the adaptive mutation strategy, archive, and Gaussian distribution elitist set selection was used to solve MMOPs. Wang et al. [63] proposed a niching method with adaptive estimation distribution where each individual adaptively determined its niche radius. Niches are co-evolved using the master-slave mechanism, and the local search based on probability is adopted to enhance the accuracy of solutions. In [24], Li proposed a ring topological particle swarm optimization (PSO). In the proposed algorithm, neighbors are determined according to the index distance. Fieldsend [11] proposed a multi-swarm PSO algorithm, where the number of swarms was dynamically changed in the process of searching. In addition to niching methods, multi-objective approaches provide another way to solve MMOPs, by transforming MMOPs into multi-objective optimization problems to satisfy the objectiveness of diversity and convergence of the population [3,6,9].

In real-world applications, many optimization problems require adequate knowledge of the fitness landscape to search for all global optima and local optima. For example, urban search and rescue (USAR) problems [31] is a research topic to search for victims in extreme conditions, such as the drowned city during a very short period. That is, USAR teams must search for every area and rescue victims as soon as possible. Lots of evolutionary algorithms and swarm intelligence are studied, searching for the path to reach the victims as fast as possible. In [12], Firthous and Kumar tested the performance of different optimization algorithms on USAR problems. In [13], Geng *et al.* designed a novel version of particle swarm optimization to solve the task allocation in USAR problems. In addition to minimizing the time, the number of rescued victims should be maximized [2,16]. In the case of a major natural disaster such as an earthquake or a tsunami, a large number of areas are revolved where there may be lots of survivors to be rescued. Obviously, USAR teams cannot blindly traverse the entire affected area, which will greatly waste manpower and material resources. Therefore, how to design algorithms to efficiently and accurately find every area where there is a probability of survivors is the main challenge of USAR problems. The USAR problem searching for lots of victims could be modeled as a many-modal optimization problem.

Compared with multimodal optimization, many-modal optimization problems are a class of problems with a large number of global optima (or acceptable local optima) and pose a great challenge to niching methods. Although MMO has been studied for many years, there is no work on many-modal optimization except in our previous conference paper [36]. In [36], we have defined and made a study of the many-modal optimization problem, and designed an optimization algorithm called difficulty-based cooperative co-evolution (DBCC) to solve it. However, the previous work is preliminary and does not consider complex environments. Hence, as an extension of our previous work [36], more complex many-modal optimization problems are considered in this work. The contributions of this work are given as follows.

(1) We propose a new benchmark for many-modal optimization problems. Among them, some are nonseparable problems, and others are additively separable problems. Each of them has many optima. In addition, test functions are more complicated than in the previous work in [36] owing to higher dimensions.
(2) We propose an improved version of difficulty-based cooperative co-evolution. Difficulty-based cooperative co-evolution (DBCC), proposed in our previous conference work [36], has four steps: problem separation, resource allocation, optimization, and solution reconstruction. In DBCC, difficulty estimation is performed only once and used as a guideline to allocate the computational resources. In this paper, we propose an improved version of DBCC, which is named DBCC2. In DBCC2, the dynamic resource allocation strategy is adopted according to the estimated difficulty.

The remainder of this paper is organized as follows: the next section introduces the related work, including MMOPs, cooperative co-evolution (CC) and typical optimizers. The third section discusses a novel many-modal benchmark of problems. DBCC2 is proposed in the fourth section. Experimental results and analysis are demonstrated in the fifth section. Finally, the last section presents conclusions and the future work.

Related work

Multimodal optimization problems (MMOPs)

MMOPs have been paid much attention in recent years, and several benchmark test suites have been proposed to test the performance of MMO algorithms. In [25], Li et al. have provided 12 test functions, where the range of dimensions is [1, 20]. The function with the largest number of global optima is F7, i.e., Vincent (3D) [25], which has 216 global optima. In [52], Qu et al. have proposed a novel MMO benchmark. Their benchmark contains eight extended simple functions and seven composition functions.

Although some problems of the above-mentioned benchmarks have many global peaks, they are not specifically designed for evaluating algorithms of many-modal optimization. In this study, we design a new many-modal optimization benchmark, in which each problem has more than 100 global optima. Importantly, we consider variable interactions of the problems. Some problems designed are additively partially separable, and others are additively nonseparable.

Cooperative co-evolution

The framework of CC [48] provides an effective divide-and-conquer strategy for complex problems. CC has been widely used in the large-scale optimization [43,62,68]. When problems can be decomposed into several subproblems, solutions of complex problems can be obtained from subproblems using co-evolving methods. The optimizer is important to search solutions of subproblems, and many existing optimizers are used in the CC framework, such as DE [44,66], PSO [4,28], genetic algorithms [48], evolutionary programming [32], and clonal selection algorithms (CSAs) [36]. Besides, two key problems should be solved in the CC framework, i.e., the decomposition method and the computational resource allocation problem, which will be explained in the following subsections.

Decomposition methods

In [67], variables are randomly divided into multiple groups. The variables in one group dynamically change in different cycles to enhance the variable interaction. Although the grouping method is sufficiently simple, full detection of the intrinsic interaction of the variables is neglected. Differential grouping (DG) [43] adopts the differential method to detect the variable interactions. Interactive variables are grouped into one group. The method of detecting the interaction between two variables is shown in Formulas (1) and (2), where Δ_1 and Δ_2 are differential values, v_1 and v_2 are different values of variable x_j, and δ is a perturbation. Two variables x_i and x_j are interactive if Formula (2) holds, where τ is a preset threshold:

$$\Delta_1 = f(\ldots, x_i + \delta, \ldots, v_1, \ldots) - f(\ldots, x_i, \ldots, v_1, \ldots)$$
$$\Delta_2 = f(\ldots, x_i + \delta, \ldots, v_2, \ldots) - f(\ldots, x_i, \ldots, v_2, \ldots). \quad (1)$$
$$|\Delta_1 - \Delta_2| > \tau. \quad (2)$$

There is an issue that DG cannot detect indirect interactions. Global differential grouping (GDG) [40] uses an interaction-related matrix to identify overlapping groups. Another issue of DG is that the threshold τ needs to be predefined and has an effect on the sensitivity of the detection. Thus, DG2 provides a method to estimate an appropriate τ, where the lower and upper bounds of roundoff errors are used [46].

An improved version, recursive differential grouping (RDG), is considered an efficient DG method [58], where the interactions between variables are recursively identified. RDG uses $\mathcal{O}(nlog(n))$ fitness evaluations to decompose the problem, which is better than the aforementioned methods. In RDG, the interactive detection occurs in a test set T of variables and an unprocessed set U. If U is detected to interact with variables in T, U is divided into two equal parts, U_1 and U_2. The detection is performed recursively in U_1 and U_2, and the interactive variables are included in T. In another case, there is no variable interaction with T. Then, T is grouped into a separable set if only one variable exists in T. Otherwise, T is identified as a new independent group. When RDG starts, the first variable in U is moved into the empty T, and the next round of detection starts until there are no variables in U. Besides, other grouping strategies have been proposed based on the interaction identification between variables like [15,57].

Computational resource allocation

Classical CC methods use the round-robin strategy to allocate computational resources to each component [33,49], which is called RBCC. RBCC does not consider the imbalanced nature of components and allocates the same computational resources for each component. Contribution-based cooperative co-evolution (CBCC) [42,45] prefers to allocate more resources to the components that have more contributions to the fitness value, and there are three versions of CBCC, i.e., CBCC1, CBCC2, and CBCC3. In CBCC1, after components are sorted by the contributions, the component with the maximum contribution is optimized in only a one-time slice. However, CBCC2 tends to optimize the subcomponent with the maximum contribution until no improvement. Hence, CBCC1 and CBCC2 have the problems of over-exploration and over-exploitation, respectively [42]. CBCC3 considers both exploration and exploitation, and achieves a more reasonable allocation. In addition to CBCC, Ren et al. [55] proposed a fine-grained contribution-aware CC framework, which allocate computational resources to components based on their contribution expectations. In [20], a CC framework called bandit-based CC was proposed, which allocates computational resources to components from the perspective of the dynamic multi-armed bandit. Moreover, Jia et al. [17] proposed DCCA with the aim of allocating more processors to the components with the larger contribution.

There have been also CC frameworks developed for other types of complex problems. For example, Xu et al. [65] proposed a constraint-objective CC framework for large-scale constrained optimization, which allocates computational resources to components with larger contributions or larger corrections based on the constraint violation of the optimal solution. Jia et al. [18] proposed a CC framework for the overlapping problems, which included a contribution-based decomposition method and a contribution-based resource allocation method. Liu et al. [30] proposed a variable importance-based resource allocation mechanism for large-scale multi-objective optimization problems, which prioritizes the allocation of more computational resources to the group of variables with higher importance.

Nevertheless, the above-mentioned resource allocation strategies do not consider that components have different search difficulties. In general, the more difficult components need to obtain more resources to find solutions than the easier ones.

Optimizer

In this part, three optimizers are introduced, which are used in our experiments.

Differential evolution

DE and its variants are widespread techniques for solving MMOPs. Here, a variant of DE for MMO in [10] is introduced, which is used in experiments. In [10], the information of the neighborhood is considered in the mutation strategy. As shown in Formulas (3)∼(4), two methods are adopted to generate a mutant z_i of x_i, which both use the nearest individual according to the Euclidean distance:

$$z_i = x_i^{nn} + F_1(x_{r1} - x_{r2}), \tag{3}$$
$$z_i = x_i^{nn} + F_1(x_{r1} - x_{r2}) + F_2(x_{r3} - x_{r4}), \tag{4}$$

where z_i denotes the mutant individual of x_i. x_i^{nn} is the individual nearest to x_i. F_1 and F_2 are the scaling factors to control the step of mutation. x_{r1}, x_{r2}, x_{r3}, and x_{r4} are randomly selected different individuals, which are also different from the i-th individual x_i. After the mutant is generated, the crossover operation is performed according to the following formula:

$$u_i^j = \begin{cases} z_i^j & \text{if } \ rand_i^j < CR \ \text{ or } \ j = j_{rand} \\ x_i^j & \text{otherwise,} \end{cases} \tag{5}$$

where CR is used to control the selected probability from the mutant, $rand_i^j \in [0, 1]$ is a random value from the uniform distribution. In addition, j_{rand} is the randomly chosen dimensional index. Hence, it ensures that at least one dimension comes from z_i^j if all random numbers are greater than CR.

Algorithm 1 shows the working mechanism at one generation. In line 3, the function $rand()$ generates a random number in the range of [0, 1] which follows the uniform distribution.

Algorithm 1 Evolution in a generation using DE

```
Input: Current population $P_{cur}$
Output: Evolved population $P_{evo}$

1: for each individual $x_i \in P_{cur}$ do
2:    Find the nearest individual $x_i^{nn}$ in the $P_{cur}$;
3:    if $rand() < 0.5$ then
4:       Perform the mutation by Formula (3);
5:    else
6:       Perform the mutation by Formula (4);
7:    end if
8:    Perform the crossover by Formula (5);
9:    Evaluate fitness of offspring $u_i$;
10:   Select the best one from $x_i$ and $u_i$ and put it into the $P_{evo}$;
11: end for
```

Particle swarm optimization

PSO is a representative algorithm of SI, which has received much attention in MMO [24,53]. PSO usually has two categories, $lbest$PSO and $gbest$PSO according to the source of social information. Particles take the global best individual as their information source of social cognition in $gbest$PSO, whereas $lbest$PSO uses the information of the local best individual. In this study, we choose $lbest$PSO as our optimizer because of better diversity.

In $lbest$PSO with a constriction factor [56], the j-th dimension of the velocity v_i of the i-th particle is updated as follows:

$$v_i^j(t+1) = \mathcal{K} * (v_i^j(t) + c_1 * rand_i^j * (pbest_i^j(t) - x_i^j(t)) + c_2 * rand_i^j * (lbest_i^j(t) - x_i^j(t))), \tag{6}$$

where $\mathcal{K}$ is a constriction factor, c_1, c_2 are learning factors, $pbest_i$ denotes the historical best position of the i-th particle, and $lbest_i$ is the historical best position of the particles in the neighborhood. The j-th dimension of the solution x_i in the $(t+1)$-th iteration is updated according to the following formula:

$$x_i^j(t+1) = x_i^j(t) + v_i^j(t+1). \tag{7}$$

Algorithm 2 shows the iteration of individuals in the same species, where f is the fitness of x. $pbest$ and $pbestCost$ are historical best locations and their fitness, respectively.

Clonal selection algorithm

CSA simulates the behaviors of cloning and selecting in the immune system [34,61], which clones and mutates antibodies and conducts the selection operation among the current individual and its offspring.

In [35], two methods of mutation are adopted in CSA: DE-like mutation and conventional Gaussian mutation. The step size of Gaussian mutation is set according to the distance

Algorithm 2 Particles perform an iteration in the same species

```
Input: $x$, $f$, $v$, $pbest$, $pbestCost$
Output: $x$, $f$, $v$, $pbest$, $pbestCost$
1: Find the best position $lbest$ according to $pbest$;
2: for each individual $x_i \in x$ do
3:     Update the velocity $v_i$ by Formula (6);
4:     Update the position $x_i$ by Formula (7);
5:     Calculate the fitness $f_i$ of $x_i$;
6: end for
7: for each individual $x_i \in x$ do
8:     if $x_i$ is better than $pbest_i$ then
9:         $pbestCost_i = f_i$;
10:        $pbest_i = x_i$;
11:    end if
12: end for
```

between the two farthest individuals in the species. DE-like mutation produces a mutant a_i of x_i, as shown in the following formula:

$$a_i = 0.5(best_i - x_i) + x_i, \tag{8}$$

where $best_i$ denotes the best individual in the same species. Algorithm 3 presents the process of reproducing the next generation antibodies in the same species. In line 2, the set $Pool_i$ stores copies of the i-th individual. In line 7, the first copy x_{c1} adopts DE-like mutation with a probability of 0.5. In lines 13 and 14, each antibody is evaluated, and the best individual is chosen from x_i and its variants including the Gaussian variants and the DE-like variant. Note that the fitness is called affinity in CSA.

Algorithm 3 Evolution in a generation using the CSA

```
Input: Current population $P_{cur}$
Output: Evolved population $P_{evo}$
1: for each individual $x_i \in P_{cur}$ do
2:     Clone $n_c$ times and include into set $Pool_i$;
3:     if $x_i$ has the highest affinity then
4:         All copies in $Pool_i$ use the Gaussian mutation;
5:     else
6:         if $rand() < 0.5$ then
7:             Take the first copy $x_{c1}$ from $Pool_i$;
8:             $x_{c1}$ performs the DE-like mutation;
9:             $Pool_i = Pool_i \backslash \{x_{c1}\}$;
10:        end if
11:        Copies in $Pool_i$ perform the Gaussian mutation;
12:    end if
13:    Evaluate the affinities of the antibodies;
14:    Include the best one from $x_i$ and variants into $P_{evo}$;
15: end for
```

Many-modal optimization problems

Many-modal optimization problems are a type of multi-modal problems which have many global optima (sometimes including acceptable local optima). Herein, we design a benchmark that could be used to test the performance of many-modal optimization algorithms. Similar to the problem construction method in [27], this paper adopts different subcomponents to construct many-modal problems. In particular, the problems designed in this paper take the difficulty differences between subcomponents into consideration. All test problems are maximization problems and have more than 100 global optima.

Structure of the proposed functions

The many-modal optimization problems designed can be divided into the following two categories.

(1) Partially additively separable problems: such many-modal optimization problems are constructed by the following formula:

$$F(X)=\sum_{i=1}^{N} g_i(X_i),\ \bigcup_{i=1}^{N} X_i = X,\ \ X_i \cap X_j = \emptyset\ (i \neq j), \tag{9}$$

where N is the number of subproblems and g_i is a non-separable function.

(2) Additively nonseparable problems: decision variables are transformed according to the following formula:

$$Y = XM, \tag{10}$$

where M is the rotation matrix. Specifically, suppose the i-th dimension in X is x_i, and the ith dimension y_i in Y is obtained by Formula (11), where D denotes the dimension of the problem. Then, $F(Y)$ is constructed using Formula (9), and the variables y_i replace the x_i in the same dimensions. Here, $X = \{x_1, x_2, ..., x_D\} = \bigcup_{i=1}^{N} X_i$.

$$y_i = m_{i1}x_1 + m_{i2}x_2 + \cdots + m_{iD}x_D. \tag{11}$$

The search range of all test functions is set to $[-100, 100]^D$, and the total fitness evaluations are set to $10^5 \cdot D$. The test suite is shown in Table 1, where MF1 and MF2 are nonseparable functions, and others are partially additively separable functions. In Table 1, r_{mm} is used to eliminate the redundant solutions close to the same peak. The way to construct the subproblems in Formula (9) will be introduced in the following part.

Table 1 Details of the benchmark of many-modal optimization problems

Function	Subproblems	Properties	*nopt*	Dimension	r_{mm}
MF1	CF1(2D), CF3(2D), CF5(2D), CF7(2D), CF9(2D)	Nonseparable	360	10	0.5
MF2	CF2(2D), CF4(2D), CF6(2D), CF8(2D), CF10(2D)	Nonseparable	480	10	0.5
MF3	CF3(2D), CF4(2D), CF6(2D), CF6(2D), CF7(2D)	Partially separable	108	10	0.5
MF4	CF3(2D), CF4(2D), CF5(2D), CF6(2D), CF8(2D)	Partially separable	144	10	0.5
MF5	CF1(3D), CF2(3D), CF3(2D), CF5(2D)	Partially separable	150	10	0.5
MF6	CF5(2D), CF6(3D), CF6(2D), CF8(3D)	Partially separable	108	10	0.5
MF7	CF6(2D), CF7(3D), CF8(2D), CF9(3D)	Partially separable	144	10	0.5
MF8	CF1(3D), CF5(2D), CF6(2D), CF7(3D)	Partially separable	135	10	0.5
MF9	CF2(2D), CF3(3D), CF5(2D), CF8(3D)	Partially separable	120	10	0.5
MF10	CF2(2D), CF5(3D), CF5(2D), CF9(3D)	Partially separable	180	10	0.5
MF11	CF1(3D), CF4(2D), CF8(2D), CF9(3D)	Partially separable	160	10	0.5
MF12	CF1(2D), CF2(3D), CF8(2D), CF10(3D)	Partially separable	400	10	0.5
MF13	CF2(2D), CF8(3D), CF9(2D), CF10(3D)	Partially separable	320	10	0.5
MF14	CF1(4D), CF2(2D), CF3(2D), CF8(2D)	Partially separable	200	10	0.5
MF15	CF2(4D), CF8(2D), CF9(2D), CF10(2D)	Partially separable	320	10	0.5

Composition functions

The subproblems in Formula (9) are all designed using the composition functions with at least two global optima. We used a method similar to [25] to formulate these composition functions. The following formula shows the details of the construction of a composite function:

$$CF(\mathbf{x}) = \sum_{i=1}^{n} \omega_i \hat{f}_i \left(\frac{\mathbf{x} - \mathbf{o}}{\lambda_i} M_i + bias_i \right) + f_{bias}, \tag{12}$$

where $\mathbf{o}$ denotes the optimal solution, n is the number of basic functions and $n \geq 2$, M_i represents the rotation matrix of the i-th component, $bias_i$ is the bias value of the i-th component which is set to 0, and f_{bias} controls the global peak height of the composite function. Without loss of generality, f_{bias} is set to 0. λ_i is implemented to compress or stretch the i-th component. Similar to [25], $\hat{f}_i$ estimates the normalization value of f_i, which is calculated by the C multiplying a ratio between the fitness at the current location fraction and the absolute fitness of the fixed position. Here, the fixed position $\mathbf{x} = [100, 100, \ldots, 100]$ and $C = 1000$. Herein, f_i is the fitness obtained by the i-th basic function.

The parameter ω_i determines the weight of the i-th basic function according to the following formulas:

$$\omega_i = \exp\left(- \frac{\sum_{i=1}^{D} \left(x_j - o_{ij} \right)^2}{2D\sigma_i^2} \right), \tag{13}$$

$$\omega_i = \begin{cases} \omega_i & \omega_i = \max(\omega_i) \\ \omega_i (1 - max(\omega_i)^{10}) & otherwise. \end{cases} \tag{14}$$

The o_{ij} is the j-th dimension of the optimum of the i-th component, and D is the dimension of the composition function to be constructed. It is worth noting that σ_i controls the coverage range of the i-th basic function. ω_i is normalized after being obtained according to Formulas (13) and (14).

In addition, all composite functions are transformed into maximization problems. The basic functions for each composition function are given in Table 2, and all composition functions are scalable.

Improved difficulty-based cooperative co-evolution

It has been shown that CC works well for complex optimization problems by decomposing them into several subproblems [38]. CC provides the basic guideline for us to implement the divide-and-conquer methods for many-modal optimization problems. The subproblems could have different difficulties in searching, but this fact received little attention in the existing CC frameworks. Hence, we have proposed a new framework of DBCC in [36], and propose its improved version (DBCC2) in this section. The framework includes the following steps namely: problem separation, resource allocation, optimization, and solution reconstruction.

Framework process of DBCC2

Algorithm 4 shows the framework process of DBCC2, where different optimizers can be embedded into this framework. In line 2, the problem is separated into several subproblems

Table 2 Composite functions (CF_1-CF_{10}) used in the benchmark

No.	Subproblems	Properties
CF_1	$f1$: Griewank function	$n = 5$
	$f2$: Rastrigin function	$\lambda = \{1, 10, 20, 1, 1\}$, $\sigma = \{10, 20, 60, 20, 10\}$
	$f3$: Weierstrass function	Global peaks: 5
	$f4$: Sphere function	
	$f5$: Sum Squares function	
CF_2	$f1$: Ackley function	$n = 5$
	$f2$: Modified periodic function *	$\lambda = \{10, 10, 10, 10, 20\}$, $\sigma = \{30, 20, 10, 10, 40\}$
	$f3$: Sphere function	Global peaks: 5
	$f4$: Salomon function	
	$f5$: Weierstrass function	
CF_3	$f1$-$f2$: Ackley function	$n = 2$
		$\lambda = \{200, 200\}$, $\sigma = \{10, 20\}$
		Global peaks: 2
CF_4	$f1$: Sphere function	$n = 2$
	$f2$: Ackley function	$\lambda = \{10, 40\}$, $\sigma = \{100, 100\}$
		Global peaks: 2
CF_5	$f1$-$f2$: Sphere function	$n = 3$
	$f3$: Griewank function	$\lambda = \{1, 1, 10\}$, $\sigma = \{15, 15, 15\}$
		Global peaks: 3
CF_6	$f1$: Sphere function	$n = 3$
	$f2$: Modified Periodic function*	$\lambda = \{1, 10, 60\}$, $\sigma = \{30, 20, 40\}$
	$f3$: Weierstrass function	Global peaks: 3
CF_7	$f1$: Griewank function	$n = 3$
	$f2$: Rastrigin function	$\lambda = \{1, 10, 10\}$, $\sigma = \{60, 80, 80\}$
	$f3$: Modified Periodic function*	Global peaks: 3
CF_8	$f1$: Griewank function	$n = 3$
	$f2$: Rastrigin function	$\lambda = \{10, 20, 10, 30\}$, $\sigma = \{60, 20, 20, 50\}$
	$f3$: Sphere function	Global peaks: 4
CF_9	$f1$-$f2$: Griewank function	$n = 4$
	$f3$: Weierstrass function	$\lambda = \{1/4, 1/4, 80, 30\}$, $\sigma = \{50, 60, 70, 80\}$
	$f4$: Ackley function	Global peaks: 4
CF_{10}	$f1$: Ackley function	$n = 4$
	$f2$: Modified Periodic function *	$\lambda = \{10, 10, 50, 30\}$, $\sigma = \{100, 20, 40, 80\}$
	$f3$-$f4$: Weierstrass function	Global peaks: 4

* Modified Periodic function: $f(x_1 \ldots x_n) = \sum_{i=1}^{n} sin^2(x_i) - 0.1e^{-(\sum_{i=1}^{n} x_i^2)} + 0.1$

according to the grouping strategy, and set X_i contains the variable indexes of subproblem SP_i. The details of problem separation are referred to in Sect. Problem separation.

The population initialization steps of the subproblems are in lines 3–7. The problems are black-box optimization problems, where the structure of subproblems can not be obtained directly. Hence, the variables whose indices are not in X_i are set to fixed values. In line 5, the lower bound of search space is set for these variables, and the initial fitness of the population in SP_i can be obtained according to line 5. The other supplemental information for the optimizer is updated in line 6. For example, in PSO, the velocity, best historical position, and best historical fitness of particles are updated in this line.

DBCC2 includes two steps to optimize the subproblems. First, the subproblems are optimized for *base* generations in line 8, which makes sure that the basic optimization is performed for each subproblem. Next, the difficulty of the subproblem is calculated, and the computational resources are allocated according to the estimated difficulty. In lines 13–25, the remaining resources are dynamically allocated. At the end of the cycle, the current difficulties of the subproblems are re-estimated, which guides the next allocation of computational resources. The number of cycles c is updated

Algorithm 4 DBCC2

```
1: Randomly initialize a population $pop$;
2: Separate the problem into $SP_1, SP_2, \ldots, SP_n$, where the corresponding variable indexes are $X_1, X_2, \ldots, X_n$; // Problem Separation (in Section Problem separation)
3: for $i = 1 : n$ do
4:     $subP_i.pop = pop[X_i]$;
5:     $subP_i.fit = f(subP_i.pop, X_i, lbounds)$;
6:     Update the supplemental information of $subP_i$;
7: end for
8: Optimize the $base$ generations for each subproblem;
9: Estimate the difficulties of the subproblems by (18); // in Section Estimated difficulty of subproblems
10: Obtain the remaining generations $remainGen$ of the problem;
11: $maxGen = remainGen$;
12: $c = 1$;
13: while $remainGen > n$ do
14:     Allocate resources according to Formula (20); // in Section Computational resource allocation
15:     for $i = 1 : n$ do
16:         $cur_{gen} = 1$;
17:         while $cur_{gen} \leq gen_i$ do
18:             $subP_i = optimizer(subP_i)$; // Algorithm 5 in Section Optimizer using BI-NBC
19:             $cur_{gen} = cur_{gen} + 1$;
20:         end while
21:     end for
22:     Re-estimate the difficulties of subproblems by (18);
23:     $c = c + 1$;
24:     Get the remaining generations $remainGen$;
25: end while
26: Choose the best solution set $B_i$ from $subP_i$; // in Section Solution reconstruction
27: Return the Cartesian set $F = B_1 \times B_2 \times \ldots \times B_n$ as the final solutions;
```

in line 23. The resource allocation can be referred to in section Resource allocation.

When the computational resources are exhausted, each subproblem gets a solution set $subP_i$, and the selection strategy shown in section Solution reconstruction is implemented to choose a suitable subset B_i. The final solutions are formed based on the Cartesian set, which is shown in line 27.

In the following contents, the components adopted in DBCC2 are illustrated in detail, including the problem separation, resource allocation, optimizers and solution reconstruction.

Problem separation

The first step of DBCC2 is to detect the structure of the problem and decompose the problem into subproblems. In the existing methods of problem separation, RDG in [58] has a good performance to detect interactive variables. In this study, we adopt RDG with minor modifications for problem separation. Each separable variable is included in an independent group. In addition, the choice of detecting points is the same as in DG2 [46]. The estimation of the upper bound of the roundoff error is also similar to DG2, which uses the absolute fitness values of the detecting points to estimate the roundoff error.

Resource allocation

Resource allocation is critical for CC, and various strategies have been proposed. Different from the existing work, we suggest that the difficulty of subproblems can be used to guide resource allocation, while the difficulty of subproblems could be estimated by the fitness-distance correlation.

The fitness-distance correlation [19] can obtain prior knowledge about the landscape using statistical correlation of the sample points. Specifically, the correlation is calculated according to the following formula:

$$r = \frac{\sum_{t \in P} (f_t - \overline{f_P})(d_t - \overline{d_P})}{\sqrt{\sum_{t \in P} (f_t - \overline{f_P})^2}\sqrt{\sum_{t \in P} (d_t - \overline{d_P})^2}}, \tag{15}$$

where P denotes the collection of the sampling points, f_t is the fitness of point t, d_t is the distance between point t and the best point, $\overline{d_P}$ and $\overline{f_P}$ are the mean values for all points in P. Ideally, there is strong correlation between fitness and distance in a linear function [41].

However, in a landscape with multiple peaks, the correlation can be weakened dramatically owing to the distribution of points at the other peaks. To handle such cases, we proposed a novel method to estimate the difficulty of subproblems as follows.

Estimated difficulty of subproblems

In the previous conference work in [36], we proposed a scheme of resource allocation based on the estimated difficulty. However, the estimation of the difficulty of subproblems is performed only once at the beginning of the algorithm, and the change of difficulty for the optimizer during the evolutionary search is not considered. In fact, the difficulties of subproblems for the algorithm are different from the initial difficulties. Furthermore, the species with poor performance are less likely to locate the peaks with high quality, which should be ignored in estimating the status of the entire population. Especially, in the later search process, the species with better performance should receive more attention.

Hence, based on the considerations above, an improved version that dynamically estimates the difficulty is proposed here.

First, in each cycle of the estimation, for subproblem f^i, each selected species S^i for difficulty estimation are required to satisfy the following two conditions.

1. S^i should have at least two individuals.
2. The fitness of the seed of S^i, i.e., $f(S^i.seed)$, should satisfy the following formula:

$$f(S^i.seed) \geq f^i_{min} + (f^i_{max} - f^i_{min})(1 - e^{-c}), \tag{16}$$

where f^i_{min} and f^i_{max} are the minimum and maximum fitness values, respectively, of all the individuals in the population for the i-th subproblem, and c denotes the current evolutionary cycle.

Second, without loss of generality, suppose the species $S^i_1, \cdots, S^i_K$ are selected; $r^i_1, \cdots, r^i_K$ are calculated according to Formula (15). Next, the weakest fitness-distance correlation in the selected species could be obtained by the following formula:

$$\mu_i = min\{\left|r^i_1\right|, \left|r^i_2\right|, \ldots, \left|r^i_K\right|\}. \tag{17}$$

Third, the difficulty of the i-th subproblem, d_i, is not linearly dependent on μ_i. For example, $\mu_i = 1$ is permanent when all picked species have two individuals. For another example, the value of μ_i close to 0 does not always indicate that the problem is very difficult, especially when individuals of the species gather to a peak. Therefore, we implement the following formula as a nonlinear mapping between μ_i and d_i, where ρ is a manual parameter:

$$d_i = \rho(1 - \mu_i)e^{-\rho(1-\mu_i)}. \tag{18}$$

Computational resource allocation

Computational resources are divided into two parts, i.e., basic resources and flexible resources. In the previous resource allocation scheme in [36], basic resources are averagely allocated to each subproblem, and the remaining flexible computational resources are allocated at once according to the difficulties of the subproblems.

Herein, basic resources are also averagely allocated to each subproblem. However, the remaining flexible resources are allocated dynamically for each evolutionary cycle. In each evolutionary cycle, the difficulty of each subproblem is estimated according to the current status of species. Suppose subproblems $SP_1, SP_2, \ldots, SP_n$ have the estimated difficulties $d_1, d_2, \ldots, d_n$ according to Formula (18). The relative difficulty of the i-th subproblem is normalized according to the following formula:

$$p_i = \frac{d_i}{\sum_{i=1}^{n} d_i}, \qquad i = 1, 2, \ldots, n. \tag{19}$$

In addition, p is fixed at 1 if the problem cannot be decomposed by the decomposition method.

Finally, the number of generations (i.e., computational resources) allocated to each subproblem in this cycle is calculated by the following formula:

$$gen_i = \lfloor p_i * (min\{\lfloor maxGen * \alpha \rfloor, remainGen\}) \rfloor, \tag{20}$$

where gen_i is the number of generations allocated to the i-th subproblem in the current cycle. $maxGen$ is the maximum generations for dynamical allocation, i.e., flexible resources. $remainGen$ denotes the remaining generations, which is updated at the end of each cycle. Parameter α controls the times of rounds in the dynamic resource allocation step.

Optimizer using BI-NBC

The NBC [50,51] is an efficient clustering method, which has attracted much attention [5,29,35]. The core idea is to construct a search tree, and each edge links the individual and its nearest better individual. Then, the edges of the tree are cut if the weights are larger than the distance threshold. Thus, multiple sub-trees are created, and individuals in a sub-tree form a population. NBC has a good performance as it uses the information about the fitness values and distances reasonably. However, the accuracy of clustering is easily affected by outliers [37].

In this study, NBC based on the better individuals is adopted, which is called BI-NBC [36]. In BI-NBC, a control strategy is adopted that only better individuals are divided into different species with NBC. Specifically, individuals with higher quality are chosen before clustering. After that, the outliers are included in their nearest cluster.

Here, we use a simple method to identify better individuals based on the fitness. The i-th individual is chosen if its fitness, f_i, is greater than the cutoff value f_c, and f_c is calculated according to the following formula:

$$f_c = \frac{\sum_{i=1}^{NP} f_i}{NP} - \epsilon, \tag{21}$$

where ϵ is a small value, which is set to 10^{-10}. Since floating point operations have rounding errors, ϵ ensures that the set of better individuals is not empty. The pseudo-code of BI-NBC is shown in Algorithm 5. In line 3, the temporary set TP contains the individuals except for the outliers. In line 6, the Euclidean distance is used.

After the population is divided into species, the optimizer is performed in each species. The pseudo-code of the optimizer with BI-NBC is shown in Algorithm 6, which is similar to our previous conference work in [36]. In Algorithm 6, S^i_j is the j-th species of the i-th subproblem.

Algorithm 5 BI-NBC [36]

```
Input: Population $P$
Output: Species $S$

1: Calculate the cutoff value, $f_c$, according to Formula (21);
2: Pick out the set of outliers $O = \{O_1, \ldots, O_m\}$;
3: $TP = P \backslash \{O\}$;
4: $TS = \text{NBC}(TP)$; //Divided by NBC
5: for $i = 1$ to $m$ do
6:     Calculate the distance between $O_i$ and seeds of $TS$;
7:     Find the nearest species $TS_j$ of $O_i$;
8:     $S_j = TS_j \cup \{O_i\}$;
9: end for
```

Algorithm 6 Optimizer with BI-NBC

```
Input: Current population of the $i$-th subproblem $subP_i$
Output: Next population $subP_i$

1: $S^i = \text{BI-NBC}(subP_i)$; // call for Algorithm 5
2: $K = length(S)$; //$K$ is the number of species
3: for $j = 1 : K$ do
4:     $S^i_j = optimizer(S^i_j)$;
5: end for
6: $subP_i = S^i_1 \cup S^i_2 \cup \cdots \cup S^i_K$;
```

Solution reconstruction

The framework of CC generates a set of solutions $subP_i$ for the i-th subproblem, and $|subP_i|$ denotes the size of $subP_i$. However, if all solutions participate in constructing the final solutions, the number of solutions will be $|subP_1| * |subP_2| * \ldots * |subP_n|$, which increases exponentially with n (the number of subproblems). On the other hand, the diversity will be lost if only the best solution of the subproblem is chosen to construct the final solution, which is not suitable for many-modal optimization problems. Therefore, it is necessary to reduce the number of solutions used in constructing the final solutions, maintain diversity, and find a suitable subset B_i from set $subP_i$. Then the number of the final solutions can reduce to $|B_1| * |B_2| * \ldots * |B_n|$.

The way to select B_i is the same as [36]. First, all solutions in $subP_i$ are arranged in descending order of fitness. The individual with the best quality b_i is removed from $subP_i$, and is added into B_i. Next, we determine if the remaining solutions of $subP_i$ are included in the set B_i or not. The solution x_k will be added to the B_i if two conditions are satisfied.

Condition 1: $|f(b_i) - f(x_k)| < \xi$, where ξ is set to 0.1.
Condition 2: the distance between x_k and all existing individuals in B_i is greater than the distance threshold γ, which is set to 0.1.

The selection loop continues until the end of $subP_i$.

Experiments

Metrics

We utilize the same method as in [25] to measure the number of the found peaks, and the error between the fitness values of the found peaks and the real global optima are not greater than the manual threshold. Here, the threshold is fixed at 1.0E-04. The distance threshold to identify the different found global peaks is greater than the radius, r_{mm}, which is shown in Table 1 and set to 0.5. The peak ratio (PR) is used to measure the overall performance of the algorithm in solving the problem. The PR can be mathematically described as the following formula:

$$PR = \frac{\sum_{Run=1}^{TR} FP_{Run}}{nopt * TR}, \tag{22}$$

where TR is the total number of runs, FP_{Run} is the found peaks in a run, and $nopt$ shown in Table 2 is the number of global optima of the test function.

In addition, we use the standard deviation (Std) of PR as another metric, which counts the standard deviation of the ratio of the found global optima.

Experimental setting

In the experiments, the four CC frameworks are used including DBCC2, DBCC, CBCC3, and RBCC. DBCC is our previous framework. RBCC allocates the same computational resources to each subproblem using the round-robin method, and CBCC3 allocates the resources based on the contributions.

In each CC framework, there are three choices of the optimizers given in section Optimizer, i.e., DE, PSO, and CSA. The size of the population (NP) in all algorithms is fixed to 500. Each algorithm is run independently 50 times.

Parameter settings are shown in Table 3. In DBCC2 and DBCC, the basic generations for each subproblem are set to 100. At the step of dynamic allocation in DBCC2, α is set to 0.1 and ρ is set to 5.0. The default parameter values are used in CBCC3. The operation of exploration is performed with a probability P_t which is set to 0.05. The contribution of the subproblem is calculated after $iter_{gap}$ generations, which is set to 100.

In DE, two mutant strategies are used with a probability of 0.5 each. The crossover rate (CR) of recombination is set to 0.9. The constriction factor $\mathcal{K}$ and c_1, c_2 in PSO are all set to the default values. Each antibody copies twice in the CSA. The scaling factor φ in BI-NBC is fixed at 2.0.

Table 3 Parameter setting

Algorithms	Parameters		
DBCC2	$base = 100$	$\alpha = 0.1$	$\rho = 5.0$
DBCC	$base = 100$	–	–
CBCC3	$P_t = 0.05$	$iter_{gap} = 100$	–
RBCC	–	–	–
DE	$F_1 = 0.5$	$F_2 = 0.5$	$CR = 0.9$
PSO	$\mathcal{K} = 0.729$	$c_1 = 2.05$	$c_2 = 2.05$
CSA	$nc = 2$	–	–
BI-NBC	$\varphi = 2.0$	$\epsilon = 10^{-10}$	–

Experimental results

The results of DE, PSO, and CSA in the different frameworks are shown in Tables 4, 5, 6. In these tables, BPR denotes the number of the best results of PR obtained using the algorithms, which are used to measure the comprehensive performance of the algorithms on the test functions. It can be seen that DBCC2 has higher BPR values than other CC frameworks under comparison.

As for nonseparable problems (MF1 and MF2), all the algorithms have poor performances. Specifically, DE in the different CC frameworks can find a few solutions, whereas PSO and CSA can not find solutions.

In the separable problems (MF3-MF15), DBCC2 performs better than the other three frameworks. Table 4 shows that DBCC2-DE and DBCC-DE have significant advantages in solving the following separable problems: MF3, MF4, MF9, MF10, and MF14. Moreover, DBCC2-DE performs better than DBCC-DE in solving 8 separable problems among all 13 separable problems. In Table 5, DBCC2-PSO

Table 4 Results of DE at an accuracy level of 1.0E-04

Function		DBCC2-DE	DBCC-DE	CBCC3-DE	RBCC-DE
MF1	PR	0.005	0.005	0.005	0.005
	Std	0.002	0.002	0.002	0.002
MF2	PR	0.005	0.005	0.005	0.005
	Std	0.001	0.001	0.001	0.001
MF3	PR	**0.786**	0.600	0.512	0.518
	Std	0.240	0.183	0.160	0.122
MF4	PR	**0.905**	0.873	0.685	0.717
	Std	0.235	0.166	0.239	0.117
MF5	PR	**0.469**	0.462	0.349	0.398
	Std	0.133	0.087	0.145	0.091
MF6	PR	0.302	**0.305**	0.187	0.289
	Std	0.099	0.122	0.099	0.104
MF7	PR	0.543	**0.564**	0.384	0.524
	Std	0.157	0.153	0.142	0.148
MF8	PR	0.440	**0.524**	0.355	0.496
	Std	0.165	0.156	0.146	0.120
MF9	PR	0.798	**0.854**	0.345	0.599
	Std	0.155	0.131	0.158	0.112
MF10	PR	**0.675**	0.582	0.369	0.483
	Std	0.117	0.123	0.109	0.076
MF11	PR	**0.594**	0.583	0.534	0.588
	Std	0.030	0.068	0.105	0.041
MF12	PR	**0.151**	0.120	0.109	0.118
	Std	0.059	0.025	0.049	0.024
MF13	PR	0.124	**0.129**	0.087	0.113
	Std	0.067	0.036	0.033	0.029
MF14	PR	**0.695**	0.641	0.499	0.507
	Std	0.179	0.130	0.184	0.106
MF15	PR	0.132	0.126	**0.172**	0.126
	Std	0.087	0.048	0.087	0.050
BPR		**7**	5	1	0

Table 5 Results of PSO at an accuracy level of 1.0E-04

Function		DBCC2-PSO	DBCC-PSO	CBCC3-PSO	RBCC-PSO
MF1	PR	0.000	0.000	0.000	0.000
	Std	0.000	0.000	0.000	0.000
MF2	PR	0.000	0.000	0.000	0.000
	Std	0.000	0.000	0.000	0.000
MF3	PR	**0.829**	0.778	0.649	0.787
	Std	0.196	0.194	0.244	0.184
MF4	PR	**0.913**	0.893	0.823	0.900
	Std	0.148	0.157	0.192	0.154
MF5	PR	0.577	0.575	**0.578**	0.549
	Std	0.118	0.097	0.120	0.117
MF6	PR	**0.411**	0.394	0.374	0.381
	Std	0.170	0.163	0.161	0.171
MF7	PR	**0.612**	0.600	0.538	0.585
	Std	0.150	0.140	0.144	0.165
MF8	PR	0.564	**0.600**	0.524	0.590
	Std	0.178	0.169	0.188	0.171
MF9	PR	**0.800**	0.781	0.796	0.766
	Std	0.107	0.098	0.103	0.102
MF10	PR	**0.445**	0.435	0.439	0.422
	Std	0.074	0.076	0.073	0.055
MF11	PR	**0.577**	0.576	0.574	0.558
	Std	0.073	0.063	0.069	0.068
MF12	PR	0.318	0.346	0.344	**0.350**
	Std	0.121	0.089	0.088	0.087
MF13	PR	0.345	0.348	0.347	**0.355**
	Std	0.088	0.072	0.081	0.087
MF14	PR	0.610	0.606	0.608	**0.611**
	Std	0.090	0.076	0.091	0.084
MF15	PR	0.297	0.294	0.290	**0.300**
	Std	0.040	0.042	0.048	0.037
BPR		**7**	1	1	4

performs better than the other algorithms in solving 7 separable problems. Compared to the other CC frameworks using the CSA, DBCC2-CSA has 8 better results among all separable functions.

In addition, the choice of the optimizers for the CC framework has a great influence on the numbers of the found global optima. It can be seen that CSA performs worse than DE and PSO for most test functions.

Comparisons with other MMO algorithms

Several state-of-art MMO algorithms are chosen for the comparisons, including FBK-DE [29], EMO-MMO [6], and MOMMOP [69]. In FBK-DE, the size of the population is set to the suggested value. In EMO-MMO, the population size is fixed at 500 and the cutting ratio in the peak detection is set to 0.1. In MOMMOP, the size of the population is set to 500, and other settings are set to the default values.

Table 7 shows the detailed results of PR and Std for all test functions. In the separable problems, DBCC2-DE and DBCC2-PSO have obvious advantages over the other algorithms. In addition, Table 8 shows the average ranks based on PR by the above algorithms as well as DBCC-DE, CBCC3-DE, and RBCC-DE in the Friedman test, which is obtained by the software tool KEEL 3.0 [1,60]. It can be seen that DBCC2-DE has the best performance and MOMMOP is inefficient in the test suite.

Furthermore, the boxplots of the found global optima in partial problems (MF3, MF7, MF10, and MF13) are shown in Fig. 1. DBCC2-DE, DBCC2-PSO, and DBCC2-CSA have higher median values in solving all four problems. Besides, among FBK-DE, EMO-MMO, and MOMMOP, EMO-MMO

Table 6 Results of CSA at an accuracy level of 1.0E-04

Function		DBCC2-CSA	DBCC-CSA	CBCC3-CSA	RBCC-CSA
MF1	PR	0.000	0.000	0.000	0.000
	Std	0.000	0.000	0.000	0.000
MF2	PR	0.000	0.000	0.000	0.000
	Std	0.000	0.000	0.000	0.000
MF3	PR	**0.462**	**0.462**	0.453	0.444
	Std	0.061	0.076	0.077	0.063
MF4	PR	**0.687**	0.682	0.669	0.640
	Std	0.080	0.110	0.057	0.132
MF5	PR	0.213	0.203	0.206	**0.216**
	Std	0.071	0.070	0.071	0.068
MF6	PR	**0.168**	0.162	0.154	0.166
	Std	0.020	0.022	0.026	0.016
MF7	PR	**0.333**	0.332	0.326	0.314
	Std	0.049	0.033	0.042	0.050
MF8	PR	**0.207**	0.197	0.205	0.188
	Std	0.057	0.055	0.059	0.056
MF9	PR	0.451	**0.465**	0.409	0.431
	Std	0.092	0.080	0.103	0.096
MF10	PR	**0.301**	0.291	0.298	0.295
	Std	0.050	0.053	0.053	0.045
MF11	PR	0.303	0.319	0.308	**0.322**
	Std	0.101	0.093	0.108	0.116
MF12	PR	0.082	**0.091**	0.085	0.085
	Std	0.018	0.024	0.020	0.020
MF13	PR	**0.088**	0.086	0.075	0.084
	Std	0.011	0.013	0.020	0.013
MF14	PR	**0.160**	0.149	0.158	0.145
	Std	0.065	0.050	0.053	0.051
MF15	PR	0.113	0.146	0.137	**0.161**
	Std	0.065	0.049	0.061	0.053
BPR		**8**	3	0	3

has the highest median value, and FBK-DE performs more robustly than EMO-MMO.

Convergence analysis

In this section, we compare the differences in the behavior of DBCC2-DE, DBCC2-PSO, and DBCC2-CSA by convergence analysis. In global optimization, the radius of the population is generally used to analyze the convergence trend of the population [21,64]. However, considering that the population is divided into multiple species by the niching strategy in multimodal optimization, the mean of the radius of each species is used to analyze the convergence trend of the population in this paper.

Figure 2 shows the trend of the mean of the species radius of DBCC2-DE, DBCC2-PSO, and DBCC2-CSA on functions MF3, MF7, MF10, and MF15. It can be seen that the convergence of PSO is the strongest and the convergence of CSA is the weakest. Also, experimental results in Table 7 show that DBCC2-PSO is the best among all the compared algorithms, while DBCC2-CSA performs worse than DBCC2-PSO and DBCC2-DE. Therefore, experimental results could support that the convergence of the optimization operator and the algorithm performance are correlated. However, the fast convergence tends to deteriorate the algorithm performance by converging the species to the local optima. As shown in Fig. 2c, DBCC2-PSO converges rapidly after the 2nd cycle such that it is unable to explore new regions in the subsequent evolutionary process. Therefore, for MF10, the performance of DBCC2-PSO is weaker than that of DBCC2-DE.

Table 7 Results of the algorithm comparison at an accuracy level of 1.0E-04

Function		DBCC2-DE	DBCC2-PSO	DBCC2-CSA	FBK-DE	EMO-MMO	MOMMOP
MF1	PR	**0.005**	0.000	0.000	0.000	0.000	0.002
	Std	0.002	0.000	0.000	0.001	0.000	0.001
MF2	PR	**0.005**	0.000	0.000	0.000	0.000	0.000
	Std	0.001	0.000	0.000	0.001	0.000	0.000
MF3	PR	0.786	**0.829**	0.462	0.095	0.343	0.000
	Std	0.240	0.196	0.061	0.018	0.083	0.000
MF4	PR	0.905	**0.913**	0.687	0.073	0.462	0.000
	Std	0.235	0.148	0.080	0.015	0.066	0.000
MF5	PR	0.469	**0.577**	0.213	0.105	0.198	0.013
	Std	0.133	0.118	0.071	0.014	0.053	0.015
MF6	PR	0.302	**0.411**	0.168	0.029	0.142	0.001
	Std	0.099	0.170	0.020	0.011	0.056	0.008
MF7	PR	0.543	**0.612**	0.333	0.075	0.170	0.009
	Std	0.157	0.150	0.049	0.013	0.095	0.019
MF8	PR	0.440	**0.564**	0.207	0.058	0.132	0.006
	Std	0.165	0.178	0.057	0.011	0.075	0.015
MF9	PR	0.798	**0.800**	0.451	0.107	0.372	0.000
	Std	0.155	0.107	0.092	0.019	0.097	0.000
MF10	PR	**0.675**	0.445	0.301	0.090	0.235	0.000
	Std	0.117	0.074	0.050	0.014	0.059	0.000
MF11	PR	**0.594**	0.577	0.303	0.102	0.227	0.009
	Std	0.030	0.073	0.101	0.016	0.050	0.017
MF12	PR	0.151	**0.318**	0.082	0.037	0.069	0.005
	Std	0.059	0.121	0.018	0.006	0.019	0.004
MF13	PR	0.124	**0.345**	0.088	0.047	0.079	0.001
	Std	0.067	0.088	0.011	0.006	0.025	0.003
MF14	PR	**0.695**	0.610	0.160	0.101	0.152	0.007
	Std	0.179	0.090	0.065	0.014	0.069	0.011
MF15	PR	0.132	**0.297**	0.113	0.033	0.087	0.001
	Std	0.087	0.040	0.065	0.006	0.029	0.002

Table 8 Average rankings of the algorithms (Friedman) according to the values of PR

Algorithm	Ranking	Algorithm	Ranking	Algorithm	Ranking	Algorithm	Ranking
DBCC2-DE	2.3333	DBCC2-PSO	2.4333	DBCC-DE	2.5	RBCC-DE	3.5667
CBCC3-DE	4.7333	DBCC2-CSA	5.9667	EMO-MMO	6.9667	FBK-DE	7.9
MOMMOP	8.6	–	–	–	–	–	–

Analysis of mechanism

In this part, the strategies adopted in DBCC2 and the parameter settings are discussed detailed. That is, problem separation, resource allocation strategy, and two parameters α, ρ are tested to illustrate the influence of the performance.

Influence of problem separation

Problem separation is firstly executed in DBCC2 to decompose the many-modal optimization problems into subproblems, which can reduce the difficulty of finding a huge amount of optimal solutions. Here, the problem separation mechanism is tested. The algorithm without problem separation is denoted as DBCC2-xx-NPS, where xx is the optimizer adopted in DBCC2. For example, DBCC2-DE-NPS repre-

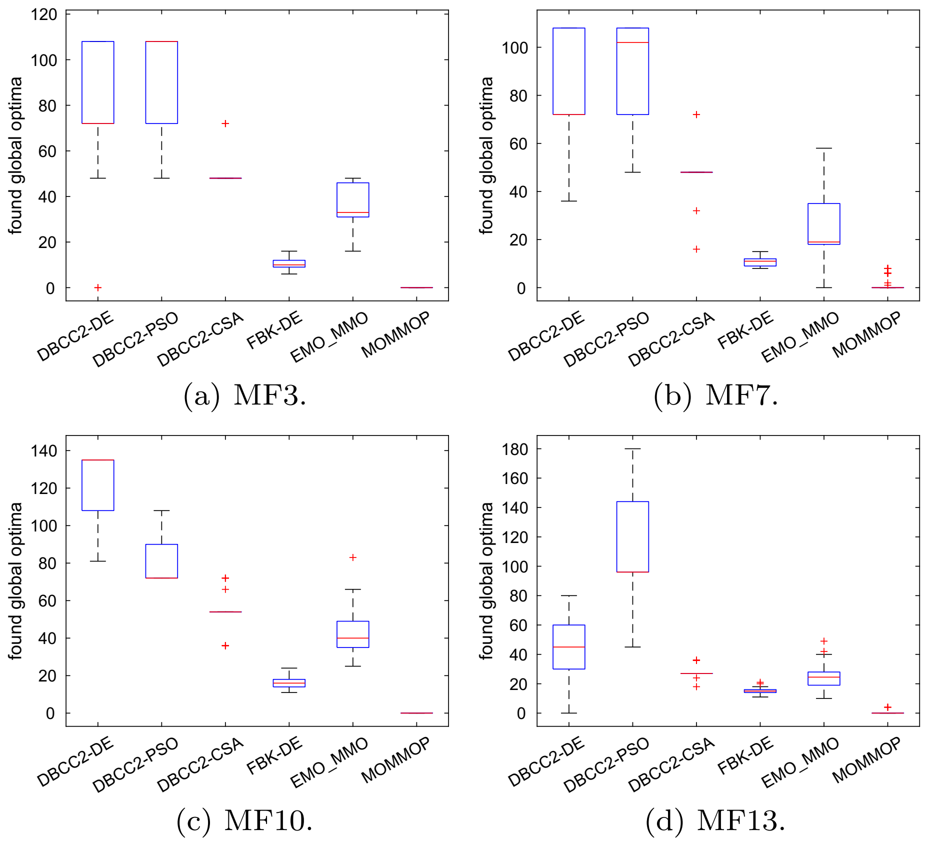

Fig. 1 Results of the found global optima on functions MF3, MF7, MF10, and MF13

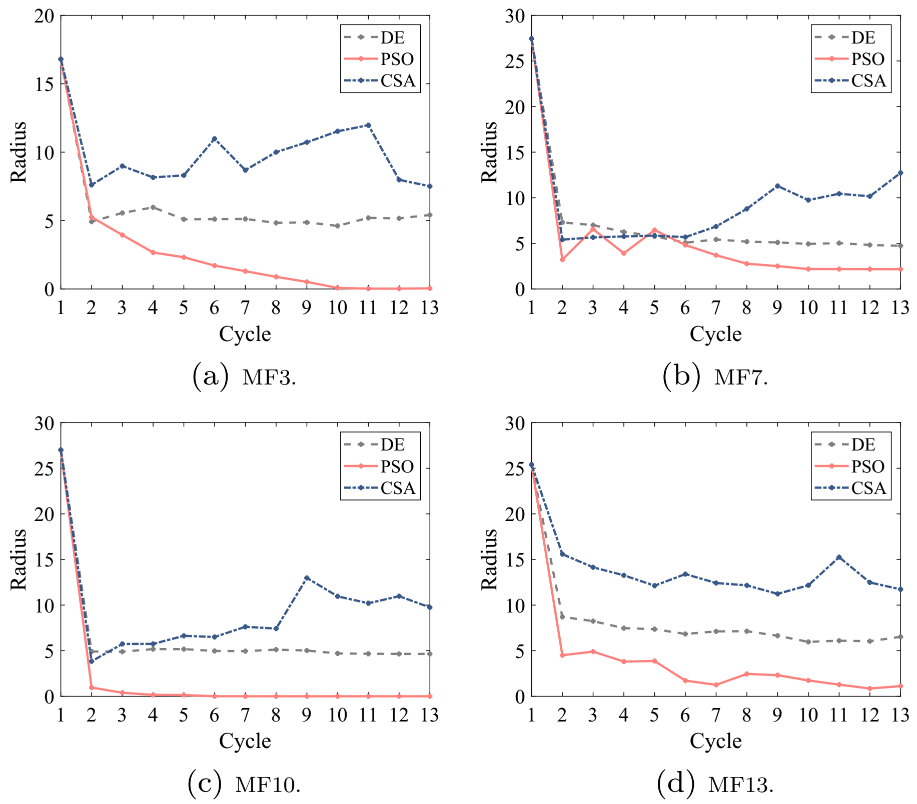

Fig. 2 Convergence trend of DBCC2 combining DE, PSO, and CSA on functions MF3, MF7, MF10, and MF13

Table 9 Results of DBCC2 with and without problem separation

Function		DBCC2-DE	DBCC2-DE-NPS	DBCC2-PSO	DBCC2-PSO-NPS	DBCC2-CSA	DBCC2-CSA-NPS
MF1	PR	**0.005**	**0.005**	0.000	0.000	0.000	0.000
	Std	0.002	0.002	0.000	0.000	0.000	0.000
MF2	PR	**0.005**	**0.005**	0.000	0.000	0.000	0.000
	Std	0.001	0.002	0.000	0.000	0.000	0.000
MF3	PR	**0.786**	0.054	**0.829**	0.040	**0.462**	0.000
	Std	0.240	0.007	0.196	0.010	0.061	0.000
MF4	PR	**0.905**	0.042	**0.913**	0.025	**0.687**	0.000
	Std	0.235	0.008	0.148	0.010	0.080	0.000
MF5	PR	**0.469**	0.046	**0.577**	0.024	**0.213**	0.000
	Std	0.133	0.008	0.118	0.011	0.071	0.000
MF6	PR	**0.302**	0.001	**0.411**	0.013	**0.168**	0.000
	Std	0.099	0.003	0.170	0.008	0.020	0.002
MF7	PR	**0.543**	0.030	**0.612**	0.018	**0.333**	0.000
	Std	0.157	0.009	0.150	0.008	0.049	0.000
MF8	PR	**0.440**	0.030	**0.564**	0.015	**0.207**	0.000
	Std	0.165	0.007	0.178	0.006	0.057	0.001
MF9	PR	**0.798**	0.058	**0.800**	0.034	**0.451**	0.000
	Std	0.155	0.008	0.107	0.016	0.092	0.000
MF10	PR	**0.675**	0.053	**0.445**	0.025	**0.301**	0.000
	Std	0.117	0.006	0.074	0.007	0.050	0.000
MF11	PR	**0.594**	0.042	**0.577**	0.021	**0.303**	0.000
	Std	0.030	0.008	0.073	0.006	0.101	0.000
MF12	PR	**0.151**	0.018	**0.318**	0.009	**0.082**	0.000
	Std	0.059	0.003	0.121	0.004	0.018	0.000
MF13	PR	**0.124**	0.022	**0.345**	0.012	**0.088**	0.000
	Std	0.067	0.004	0.088	0.005	0.011	0.000
MF14	PR	**0.695**	0.045	**0.610**	0.024	**0.160**	0.000
	Std	0.179	0.006	0.090	0.009	0.065	0.000
MF15	PR	**0.132**	0.013	**0.297**	0.008	**0.113**	0.000
	Std	0.087	0.003	0.040	0.004	0.065	0.000
NBR		**15**	2	**13**	0	**13**	0

sents the proposed DBCC2 with DE optimizer and without the problem separation mechanism.

Table 9 shows the mean and standard deviation of PR results of DBCC2 and its version without problem separation. The experiment results show that problem separation performs powerfully in the many-modal optimization problems, especially in the partially separable benchmarks. In detail, in the comparison with the DE optimizer, DBCC2-DE performs much better than DBCC2-DE-NPS. On MF4, DBCC2-DE almost finds all the optima (i.e., 0.905) but DBCC2-DE-NPS obtains less than 10% optimal solutions. This huge gap also exists in the comparisons of PSO and CSA optimizers. It is noted that without problem separation, DBCC2-CSA-NPS hardly finds any optimal peaks.

Overall, the problem separation mechanism is useful in solving many-modal optimization problems with the separable property. For DE optimizer, DBCC2-DE gets the best performance on all benchmarks and the version without problem separation performs the best only on MF1 and MF2. For the PSO and CSA optimizers, DBCC2 obtains more powerful performance on all separable benchmarks in the corresponding comparisons.

Influence of resource allocation

The dynamic resource allocation strategy is proposed in DBCC2, and it can dynamically allocate the computing resources to the subproblems with different difficulties. Here, the performance of the resource allocation strategy is tested.

Table 10 Results of DBCC2 with and without resource allocation strategy

Function		DBCC2-DE	DBCC2-DE-NRA	DBCC2-PSO	DBCC2-PSO-NRA	DBCC2-CSA	DBCC2-CSA-NRA
MF1	PR	**0.005**	**0.005**	0.000	0.000	0.000	0.000
	Std	0.002	0.002	0.000	0.000	0.000	0.000
MF2	PR	**0.005**	**0.005**	0.000	0.000	0.000	0.000
	Std	0.001	0.002	0.000	0.000	0.000	0.000
MF3	PR	**0.786**	0.624	**0.829**	**0.829**	**0.462**	0.449
	Std	0.240	0.185	0.196	0.196	0.061	0.071
MF4	PR	**0.905**	0.768	**0.913**	**0.913**	**0.687**	0.680
	Std	0.235	0.154	0.148	0.147	0.080	0.066
MF5	PR	**0.469**	0.398	0.577	**0.582**	**0.213**	0.210
	Std	0.133	0.008	0.118	0.120	0.071	0.073
MF6	PR	0.302	**0.317**	**0.411**	**0.411**	**0.168**	0.167
	Std	0.099	0.138	0.170	0.170	0.020	0.014
MF7	PR	0.543	**0.587**	**0.612**	**0.612**	**0.333**	0.322
	Std	0.157	0.151	0.150	0.150	0.049	0.059
MF8	PR	0.440	**0.552**	0.564	**0.567**	**0.207**	0.197
	Std	0.165	0.161	0.178	0.176	0.057	0.057
MF9	PR	**0.798**	0.669	**0.800**	**0.800**	**0.451**	0.449
	Std	0.155	0.149	0.107	0.107	0.092	0.073
MF10	PR	**0.675**	0.549	**0.445**	**0.445**	0.301	**0.308**
	Std	0.117	0.099	0.074	0.074	0.050	0.040
MF11	PR	**0.594**	0.593	0.577	**0.580**	0.303	**0.304**
	Std	0.030	0.060	0.073	0.071	0.101	0.104
MF12	PR	**0.151**	0.116	0.318	**0.350**	**0.082**	**0.082**
	Std	0.059	0.029	0.121	0.086	0.018	0.020
MF13	PR	**0.124**	0.119	0.345	**0.351**	**0.088**	0.086
	Std	0.067	0.030	0.088	0.079	0.011	0.011
MF14	PR	**0.695**	0.505	**0.610**	**0.610**	**0.160**	0.144
	Std	0.179	0.100	0.090	0.090	0.065	0.046
MF15	PR	**0.132**	0.117	0.297	**0.299**	0.113	**0.134**
	Std	0.087	0.044	0.040	0.039	0.065	0.043
BPR		**12**	5	7	**13**	**10**	4

We denote the proposed algorithms without the resource allocation strategy as DBCC2-xx-NRA, where xx represents the selected optimizer from DE, PSO and CSA.

Table 10 shows the mean and standard deviation of PR results of DBCC2 with and without the resource allocation strategy, respectively. It can be seen that the resource allocation strategy performs better in the algorithms with DE and CSA optimizers, but gets a poor performance with PSO optimizer. For example, on MF10, DBCC2-DE obtains 0.675 PR results, while DBCC2-DE-NRA finds a 0.549 ratio of the optimal peaks. On MF14, DBCC2-CSA achieves 0.160 results, performing better than DBCC2-CSA-NRA, which gets 0.144 PR results.

In summary, the resource allocation strategy is useful in DBCC2 framework with DE and CSA optimizers. In the current experiment, DBCC2-DE obtains the best results on 12 benchmark problems, while the version without the resource allocation strategy obtains the best results on 5 problems. As for the CSA optimizer, DBCC2-CSA obtains the best results on 10 benchmarks than DBCC2-CSA-NRA, which obtains the best results only on 4 problems. However, the reason that DBCC2-PSO-NRA is better DBCC2-PSO is worthy of further studying.

Influence of α

In DBCC2, α controls the cycles in the step of dynamic resource allocation. The values of α are chosen from the set $\{0.05, 0.1, 0.2, 0.3, 0.5\}$. The smaller the value of α, the more the cycles of DBCC2. Here, DE is used as the opti-

Table 11 Results of DBCC2-DE taking different values of α

Function		α = 0.05	α = 0.1	α = 0.2	α = 0.3	α = 0.5
MF1	PR	0.005	0.005	0.005	0.005	0.005
	Std	0.002	0.002	0.002	0.002	0.002
MF2	PR	0.005	0.005	0.005	0.005	0.005
	Std	0.001	0.001	0.001	0.001	0.001
MF3	PR	0.729	**0.786**	0.653	0.659	0.604
	Std	0.271	0.240	0.252	0.221	0.220
MF4	PR	0.890	**0.905**	0.822	0.832	0.767
	Std	0.232	0.235	0.244	0.219	0.205
MF5	PR	0.461	**0.469**	0.428	0.429	0.402
	Std	0.172	0.133	0.112	0.125	0.118
MF6	PR	0.273	0.302	0.298	**0.314**	0.274
	Std	0.088	0.099	0.113	0.154	0.114
MF7	PR	0.517	**0.543**	0.542	**0.543**	0.476
	Std	0.156	0.157	0.175	0.165	0.185
MF8	PR	0.435	0.440	0.464	**0.491**	0.440
	Std	0.180	0.165	0.154	0.191	0.184
MF9	PR	0.679	**0.798**	0.763	0.735	0.742
	Std	0.262	0.155	0.199	0.202	0.179
MF10	PR	0.671	**0.675**	0.668	0.660	0.642
	Std	0.141	0.117	0.113	0.125	0.122
MF11	PR	0.592	**0.594**	0.578	0.588	0.582
	Std	0.051	0.030	0.055	0.041	0.049
MF12	PR	0.091	**0.151**	0.134	0.130	0.120
	Std	0.081	0.059	0.049	0.028	0.032
MF13	PR	0.036	**0.124**	0.141	0.145	0.134
	Std	0.066	0.067	0.049	0.034	0.037
MF14	PR	0.653	**0.695**	0.661	0.592	0.568
	Std	0.182	0.179	0.187	0.190	0.146
MF15	PR	0.102	0.132	0.147	0.138	**0.153**
	Std	0.076	0.087	0.058	0.063	0.055
BPR		0	**10**	0	3	1

Table 12 Results of DBCC2-DE taking different values of ρ

Function		ρ = 1.0	ρ = 2.0	ρ = 3.0	ρ = 5.0	ρ = 10.0
MF1	PR	0.005	0.005	0.005	0.005	0.005
	Std	0.002	0.002	0.002	0.002	0.002
MF2	PR	0.005	0.005	0.005	0.005	0.005
	Std	0.001	0.001	0.001	0.001	0.001
MF3	PR	0.663	0.734	**0.786**	**0.786**	0.620
	Std	0.265	0.287	0.270	0.240	0.278
MF4	PR	0.765	0.833	0.895	**0.905**	0.815
	Std	0.292	0.271	0.241	0.235	0.257
MF5	PR	0.474	0.486	**0.496**	0.469	0.387
	Std	0.170	0.158	0.159	0.133	0.156
MF6	PR	0.267	0.286	**0.306**	0.302	0.270
	Std	0.096	0.093	0.130	0.099	0.114
MF7	PR	0.425	0.509	**0.608**	0.543	0.487
	Std	0.128	0.175	0.138	0.157	0.170
MF8	PR	0.484	0.494	**0.527**	0.440	0.356
	Std	0.144	0.165	0.128	0.165	0.153
MF9	PR	0.870	0.880	**0.903**	0.798	0.476
	Std	0.177	0.177	0.125	0.155	0.311
MF10	PR	0.585	0.687	**0.718**	0.675	0.561
	Std	0.133	0.113	0.079	0.117	0.167
MF11	PR	0.591	0.580	**0.595**	0.594	0.576
	Std	0.075	0.069	0.047	0.030	0.068
MF12	PR	0.063	0.096	0.131	**0.151**	0.108
	Std	0.072	0.073	0.059	0.059	0.049
MF13	PR	0.030	0.092	**0.149**	0.124	0.092
	Std	0.058	0.089	0.064	0.067	0.056
MF14	PR	0.604	0.635	**0.701**	0.695	0.567
	Std	0.201	0.187	0.172	0.179	0.234
MF15	PR	0.044	0.075	0.096	0.132	**0.135**
	Std	0.058	0.063	0.075	0.087	0.058
BPR		0	0	10	3	1

mizer of DBCC2 to illustrate the influence. Table 11 shows the results for different values of α. Overall, DBCC2-DE has the best performance when α is equal to 0.1.

The algorithm requires more rounds to allocate the resources when α is set to 0.05. However, it can be seen that the performance is worse than that of DBCC2-DE with $\alpha = 0.1$. The experimental result indicates that more rounds of difficulty estimation do not always produce better performance.

The performance of DBCC2-DE with $\alpha = 0.5$ is worse than that with $\alpha = 0.1$. The former has only one better result in solving the separable problems, which shows that a small number of the difficulty estimation rounds cannot reflect the population status in time.

Influence of ρ

The parameter ρ has an impact on the difficulty estimation. Here, the value of ρ is taken from the set {1.0, 2.0, 3.0, 5.0, 10.0}, and the results are shown in Table 12. The parameter ρ plays a key role in the mapping from the fitness-distance correlation to the estimated difficulty. As discussed in Section Improved difficulty-based cooperative co-evolution, the fitness-distance correlation does not reflect the difficulty linearly. In Formula (18), ρ intrinsically determines which value of μ will lead to the maximum value of the estimated difficulty. For example, $\rho = 2.0$ indicates that $\mu = 1/2$ in Formula (18) results in the maximum estimated difficulty.

The experimental results show that there are significant advantages when ρ is set to 3.0, where the maximum value of the estimated difficulty is achieved with $\mu = 2/3$. There are 10 better results among the 13 separable problems.

Furthermore, in a majority of the problems, DBCC2-DE tends to achieve better performance when ρ is increased from 1.0 to 3.0 and performs worse performance when ρ is decreased from 3.0 to 10.0. For example, in MF5, the value of PR increases from 0.474 to 0.496 when the value of ρ is increased from 1.0 to 3.0. The value of PR decreases to 0.387 when ρ is set to 10.0.

Finally, for all separable problems, except for MF15, DBCC2-DE achieves better performance when ρ is set to 3.0 or 5.0. As for MF1 and MF2, the algorithms have the same results for different values of ρ as they are nonseparable.

Conclusion

In this paper, many-modal optimization problems are investigated. A new benchmark and an improved DBCC (DBCC2), which dynamically estimates the difficulty of subproblems in the search process, are proposed. DBCC2 shows better performance as compared to other typical MMO algorithms. However, much remains to be done in the future. Optimizers, resource allocation strategies, and CC frameworks of many-modal optimization problems still deserve further study.

Acknowledgements The authors would like to thank Cheng et al. for providing the source code of EMO-MMO [6]. Our source code is available at https://github.com/bonsur-ustc/DBCC2.

References

1. Alcalá-Fdez J, Fernández A, Luengo J, Derrac J, García S, Sánchez L, Herrera F (2011) KEEL data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. Multiple-Value Logic Soft Comput 17:255–287
2. Alotaibi ET, Alqefari SS, Koubaa A (2019) LSAR: Multi-UAV collaboration for search and rescue missions. IEEE Access 7:55817–55832
3. Basak A, Das S, Tan KC (2012) Multimodal optimization using a biobjective differential evolution algorithm enhanced with mean distance-based selection. IEEE Trans Evol Comput 17:666–685
4. Van den Bergh F, Engelbrecht AP (2004) A cooperative approach to particle swarm optimization. IEEE Trans Evol Comput 8:225–239
5. Bu C, Luo W, Zhu T, Yi R, Yang B (2019) A species and memory enhanced differential evolution for optimal power flow under double-sided uncertainties. IEEE Transactions on Sustainable Computing 5:403–415
6. Cheng R, Li M, Li K, Yao X (2017) Evolutionary multiobjective optimization-based multimodal optimization: fitness landscape approximation and peak detection. IEEE Trans Evol Comput 22:692–706
7. Das S, Maity S, Qu BY, Suganthan PN (2011) Real-parameter evolutionary multimodal optimization-a survey of the state-of-the-art. Swarm Evol Comput 1:71–88
8. De Jong KA (1975) Analysis of the behavior of a class of genetic adaptive systems. Ph.D. dissertation, Univ. Michigan, Ann Arbor
9. Deb K, Saha A (2012) Multimodal optimization using a bi-objective evolutionary algorithm. Evol Comput 20:27–62
10. Epitropakis MG, Li X, Burke EK (2013) A dynamic archive niching differential evolution algorithm for multimodal optimization, in: Proceedings of 2013 IEEE Conference on Evolutionary Computation, IEEE. pp. 79–86
11. Fieldsend JE (2014) Running up those hills: Multi-modal search with the niching migratory multi-swarm optimiser, in: Proceedings of 2014 IEEE Congress on Evolutionary Computation, IEEE. pp. 2593–2600
12. Firthous MAA, Kumar R (2020) Multiple oriented robots for search and rescue operations, in: Proceedings of IOP Conference Series: Materials Science and Engineering, IOP Publishing. p. 032023
13. Geng N, Meng Q, Gong D, Chung PW (2018) How good are distributed allocation algorithms for solving urban search and rescue problems? a comparative study with centralized algorithms. IEEE Trans Autom Sci Eng 16:478–485
14. Goldberg DE, Richardson J, et al., (1987) Genetic algorithms with sharing for multimodal function optimization, in: Proceedings of the Second International Conference on Genetic Algorithms, Hillsdale, NJ: Lawrence Erlbaum. pp. 41–49
15. Hu X, He F, Chen W, Zhang J (2017) Cooperation coevolution with fast interdependency identification for large scale optimization. Inf Sci 381:142–160
16. Jacoff A, Messina E, Evans J, et al., (2001) A standard test course for urban search and rescue robots. NIST special publication SP , 253–259
17. Jia YH, Chen WN, Gu T, Zhang H, Yuan HQ, Kwong S, Zhang J (2018) Distributed cooperative co-evolution with adaptive computing resource allocation for large scale optimization. IEEE Trans Evol Comput 23:188–202
18. Jia YH, Mei Y, Zhang M (2020) Contribution-based cooperative co-evolution for nonseparable large-scale problems with overlapping subcomponents. IEEE Transactions on Cybernetics. https://doi.org/10.1109/TCYB.2020.3025577
19. Jones T, Forrest S (1995) Fitness distance correlation as a measure of problem difficulty for genetic algorithms, in: Proceedings of the 6th International Conference on Genetic Algorithms, pp. 184–192
20. Kazimipour B, Omidvar MN, Qin AK, Li X, Yao X (2019) Bandit-based cooperative coevolution for tackling contribution imbalance in large-scale optimization problems. Appl Soft Comput 76:265–281
21. Li C, Yang S (2009) A clustering particle swarm optimizer for dynamic optimization, in: Proceedings of the 2009 IEEE Congress on Evolutionary Computation (CEC), IEEE. pp. 439–446
22. Li JP, Balazs ME, Parks GT, Clarkson PJ (2002) A species conserving genetic algorithm for multimodal function optimization. Evol Comput 10:207–234
23. Li X (2005) Efficient differential evolution using speciation for multimodal function optimization, in: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation, ACM. pp. 873–880

24. Li X (2009) Niching without niching parameters: particle swarm optimization using a ring topology. IEEE Trans Evol Comput 14:150–169
25. Li X, Engelbrecht A, Epitropakis MG (2013) Benchmark functions for CEC'2013 special session and competition on niching methods for multimodal function optimization. RMIT University, Evolutionary Computation and Machine Learning Group, Australia, Tech, Rep
26. Li X, Epitropakis MG, Deb K, Engelbrecht A (2017) Seeking multiple solutions: an updated survey on niching methods and their applications. IEEE Trans Evol Comput 21:518–538
27. Li X, Tang K, Omidvar MN, Yang Z, Qin K, China H (2013b) Benchmark functions for the CEC'2013 special session and competition on large-scale global optimization. School of Computer Science and Information Technology, RMIT University, Melbourne, Australia and School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China and National University of Defense Technology, Changsha, China, Technical Report
28. Li X, Yao X (2011) Cooperatively coevolving particle swarms for large scale optimization. IEEE Trans Evol Comput 16:210–224
29. Lin X, Luo W, Xu P (2021) Differential evolution for multimodal optimization with species by nearest-better clustering. IEEE Transactions on Cybernetics 51:970–983
30. Liu S, Lin Q, Tian Y, Tan KC (2021) A variable importance-based differential evolution for large-scale multiobjective optimization. IEEE Transactions on Cybernetics. https://doi.org/10.1109/TCYB.2021.3098186
31. Liu Y, Nejat G (2013) Robotic urban search and rescue: A survey from the control perspective. Journal of Intelligent & Robotic Systems 72:147–165
32. Liu Y, Yao X, Zhao Q, Higuchi T (2001) Scaling up fast evolutionary programming with cooperative coevolution, in: Proceedings of 2001 IEEE Congress on Evolutionary Computation, IEEE. pp. 1101–1108
33. Liu Y, Yao X, Zhao Q, Higuchi T (2002) Scaling up fast evolutionary programming with cooperative coevolution, in: Proceedings of the 2001 Congress on Evolutionary Computation, IEEE. pp. 1101–1108
34. Luo W, Lin X (2017) Recent advances in clonal selection algorithms and applications, in: Proceedings of 2017 IEEE Symposium Series on Computational Intelligence, IEEE. pp. 1–8
35. Luo W, Lin X, Zhu T, Xu P (2019) A clonal selection algorithm for dynamic multimodal function optimization. Swarm Evol Comput 50:100459
36. Luo W, Qiao Y, Lin X, Xu P, Preuss M (2019b) Many-modal optimization by difficulty-based cooperative co-evolution, in: Proceedings of 2019 IEEE Symposium Series on Computational Intelligence, IEEE. pp. 1907–1914
37. Luo W, Sun J, Bu C, Yi R (2018) Identifying species for particle swarm optimization under dynamic environments, in: Proceedings of 2018 IEEE Symposium Series on Computational Intelligence, IEEE. pp. 1921–1928
38. Ma X, Li X, Zhang Q, Tang K, Liang Z, Xie W, Zhu Z (2019) A survey on cooperative co-evolutionary algorithms. IEEE Trans Evol Comput 23:421–441
39. Mahfoud SW (1992) Crowding and preselection revisited, in: Parallel Problem Solving from Nature-PPSN, pp. 27–36
40. Mei Y, Omidvar MN, Li X, Yao X (2016) A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization. ACM Transactions on Mathematical Software 42:13
41. Naudts B, Kallel L (2000) A comparison of predictive measures of problem difficulty in evolutionary algorithms. IEEE Trans Evol Comput 4:1–15
42. Omidvar MN, Kazimipour B, Li X, Yao X (2016) CBCC3 - A contribution-based cooperative co-evolutionary algorithm with improved exploration/exploitation balance, in: Proceedings of 2016 IEEE Congress on Evolutionary Computation, IEEE. pp. 3541–3548
43. Omidvar MN, Li X, Mei Y, Yao X (2013) Cooperative co-evolution with differential grouping for large scale optimization. IEEE Trans Evol Comput 18:378–393
44. Omidvar MN, Li X, Yang Z, Yao X (2010) Cooperative co-evolution for large scale optimization through more frequent random grouping, in: Proceedings of 2010 IEEE Congress on Evolutionary Computation, IEEE. pp. 1–8
45. Omidvar MN, Li X, Yao X (2011) Smart use of computational resources based on contribution for cooperative co-evolutionary algorithms, in: Proceedings of the 13th Annual Conference on Genetic and evolutionary computation, ACM. pp. 1115–1122
46. Omidvar MN, Yang M, Mei Y, Li X, Yao X (2017) DG2: A faster and more accurate differential grouping for large-scale black-box optimization. IEEE Trans Evol Comput 21:929–942
47. Pétrowski A (1996) A clearing procedure as a niching method for genetic algorithms, in: Proceedings of 1996 IEEE International Conference on Evolutionary Computation, IEEE. pp. 798–803
48. Potter MA, De Jong KA (1994) A cooperative coevolutionary approach to function optimization, in: Proceedings of 1994 International Conference on Parallel Problem Solving from Nature, Springer. pp. 249–257
49. Potter MA, Jong KAD (1994) A cooperative coevolutionary approach to function optimization, in: Proceedings of Parallel Problem Solving from Nature- PPSN III, Springer. pp. 249–257
50. Preuss M (2010) Niching the CMA-ES via nearest-better clustering, in: Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation, ACM. pp. 1711–1718
51. Preuss M, Schönemann L, Emmerich M (2005) Counteracting genetic drift and disruptive recombination in (μ^{+}, λ)-EA on multimodal fitness landscapes, in: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation, ACM. pp. 865–872
52. Qu B, Liang J, Wang Z, Chen Q, Suganthan PN (2016) Novel benchmark functions for continuous multimodal optimization with comparative results. Swarm Evol Comput 26:23–34
53. Qu BY, Suganthan PN, Das S (2013) A distance-based locally informed particle swarm model for multimodal optimization. IEEE Trans Evol Comput 17:387–402
54. Qu BY, Suganthan PN, Liang JJ (2012) Differential evolution with neighborhood mutation for multimodal optimization. IEEE Trans Evol Comput 16:601–614
55. Ren Z, Liang Y, Zhang A, Yang Y, Feng Z, Wang L (2018) Boosting cooperative coevolution for large scale optimization with a fine-grained computation resource allocation strategy. IEEE Transactions on Cybernetics 49:4180–4193
56. Shi Y, Eberhart R (1998) A modified particle swarm optimizer, in: Proceedings of 1998 IEEE Conference on Evolutionary Computation, IEEE. pp. 69–73
57. Sun Y, Kirley M, Halgamuge SK (2015) Extended differential grouping for large scale global optimization with direct and indirect variable interactions, in: Proceedings of 2015 Annual Conference on Genetic and Evolutionary Computation, ACM. pp. 313–320
58. Sun Y, Kirley M, Halgamuge SK (2017) A recursive decomposition method for large scale continuous optimization. IEEE Trans Evol Comput 22:647–661
59. Thomsen R (2004) Multimodal optimization using crowding-based differential evolution, in: Proceeding of 2004 IEEE International Conference on Evolutionary Computation, IEEE. pp. 1382–1389
60. Triguero I, González S, Moyano JM, García López S, Alcalá Fernández J, Luengo Martín J, Fernández Hilario A, Díaz J, Sánchez L, Herrera F et al (2017) KEEL 3.0: an open source software for multi-stage analysis in data mining. International Journal of Computational Intelligence Systems 10:1238–1249

61. Ulutas BH, Kulturel-Konak S (2011) A review of clonal selection algorithm and its applications. Artif Intell Rev 36:117–138
62. Wang ZJ, Zhan ZH, Yu WJ, Lin Y, Zhang J, Gu TL, Zhang J (2020) Dynamic group learning distributed particle swarm optimization for large-scale optimization and its application in cloud workflow scheduling. IEEE Transactions on Cybernetics 50:2715–2729
63. Wang ZJ, Zhou YR, Zhang J (2020) Adaptive estimation distribution distributed differential evolution for multimodal optimization problems. IEEE Transactions on Cybernetics. https://doi.org/10.1109/TCYB.2020.3038694
64. Xu P, Luo W, Lin X, Cheng S, Shi Y (2021) BSO20: efficient brain storm optimization for real-parameter numerical optimization. Complex & Intelligent Systems 7:2415–2436
65. Xu P, Luo W, Lin X, Zhang J, Qiao Y, Wang X (2021) Constraint-objective cooperative coevolution for large-scale constrained optimization. ACM Transactions on Evolutionary Learning and Optimization 1:1–26
66. Yang Z, Tang K, Yao X (2008) Large scale evolutionary optimization using cooperative coevolution. Information Sciences 178:2985–2999
67. Yang Z, Tang K, Yao X (2008) Large scale evolutionary optimization using cooperative coevolution. Information Sciences 178:2985–2999
68. Yazdani D, Omidvar MN, Branke J, Nguyen TT, Yao X (2019) Scaling up dynamic optimization problems: A divide-and-conquer approach. IEEE Transactions on Evolutionary Computation
69. Yong W, Hanxiong L, Yen GG, Wu S (2015) MOMMOP: Multi-objective optimization for locating multiple optimal solutions of multimodal optimization problems. IEEE Transactions on Cybernetics 45:830–843
70. Zhao H, Zhan Z, Lin Y, Chen X, Luo X, Zhang J, Kwong S, Zhang J (2020) Local binary pattern based adaptive differential evolution for multimodal optimization problems. IEEE Transactions on Cybernetics 50:3343–3357
71. Zhao H, Zhan Z, Zhang J (2020b). Adaptive guidance-based differential evolution with iterative feedback archive strategy for multimodal optimization problems, in: Proceeding of 2020 IEEE Conference on Evolutionary Computation, IEEE. pp. 1–8