Contents lists available at ScienceDirect

Applied Soft Computing Journal

journal homepage: www.elsevier.com/locate/asoc

Bandit-based cooperative coevolution for tackling contribution imbalance in large-scale optimization problems

Borhan Kazimipour [a,*], Mohammad Nabi Omidvar [b], A.K. Qin [c], Xiaodong Li [a], Xin Yao [b,d]

[a] *School of Science (Computer Science and Software Engineering), Melbourne, Victoria, 3000, Australia*
[b] *School of Computer Science, University of Birmingham, Birmingham B15 2TT, UK*
[c] *Department of Computer Science and Software Engineering, Swinburne University of Technology, John Street, Hawthorn, Victoria, 3122, Australia*
[d] *Shenzhen Key Laboratory of Computational Intelligence, University Key Laboratory of Evolving Intelligent Systems of Guangdong Province, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China*

HIGHLIGHTS

- Where subproblems do not contribute equally to the overall objective value, allocating the budget uniformly to all subproblems is a waste of valuable resources.
- Multi-armed bandit techniques can be adopted to effectively allocate computational resource within the context of cooperative coevolution.
- Even the simplest implementation of bandit-based credit assignment scheme can significantly outperform its analogous round-robin algorithms.
- The modularity of the proposed framework allows us to create many different instances of the framework.

ARTICLE INFO

Article history:
Received 10 July 2017
Received in revised form 12 November 2018
Accepted 8 December 2018
Available online 18 December 2018

Keywords:
Imbalanced large-scale optimization
Resource allocation
Multi-armed bandits

ABSTRACT

This paper addresses the issue of computational resource allocation within the context of cooperative coevolution. Cooperative coevolution typically works by breaking a problem down into smaller subproblems (or components) and coevolving them in a round-robin fashion, resulting in a uniform resource allocation among its components. Despite its success on a wide range of problems, cooperative coevolution struggles to perform efficiently when its components do not contribute equally to the overall objective value. This is of crucial importance on large-scale optimization problems where such difference are further magnified. To resolve this *imbalance* problem, we extend the standard cooperative coevolution to a new generic framework capable of learning the contribution of each component using multi-armed bandit techniques. The new framework allocates the computational resources to each component proportional to their contributions towards improving the overall objective value. This approach results in a more economical use of the limited computational resources. We study different aspects of the proposed framework in the light of extensive experiments. Our empirical results confirm that even a simple bandit-based credit assignment scheme can significantly improve the performance of cooperative coevolution on large-scale continuous problems, leading to competitive performance as compared to the state-of-the-art algorithms.

© 2018 Elsevier B.V. All rights reserved.

1. Introduction

Optimization tasks are found in numerous applications ranging from modeling biological systems [1] to engineering design problems [2]. With the rapid advances in science and technology, more complex optimization problems are emerging every day. Since the traditional optimization methods usually do not scale well to very high dimensional problems [3], a *divide-and-conquer* strategy can be adopted to alleviate the issue of *curse of dimensionality*. In recent years, many meta-heuristic algorithms have leveraged such decomposition strategies to tackle large-scale problems [4].

Cooperative Coevolutionary (CC) is a classic evolutionary technique which follows the divide-and-conquer approach. The CC framework requires decision variables to be grouped into separate components (or subproblems), and subsequently optimization is carried out on each component one at a time in a round-robin fashion. A default option for CC regarding resource allocation is to allocate an equal portion of the computational budget to each subproblem. A large and growing body of literature is devoted to utilizing effective decomposition techniques with CC algorithms [5,

* Corresponding author.
E-mail addresses: borhan.kazimipour@monash.edu (B. Kazimipour), m.omidvar@cs.bham.ac.uk (M.N. Omidvar), qin@swin.edu.au (A.K. Qin), xiaodong.li@rmit.edu.au (X. Li), x.yao@cs.bham.ac.uk (X. Yao).

https://doi.org/10.1016/j.asoc.2018.12.007
1568-4946/© 2018 Elsevier B.V. All rights reserved.

6] and advancing the schemes for their computational budget allocation [7,8].

Despite showing promising performances on numerous applications [4], CC models do not function efficiently and effectively when the problem is *imbalanced* [7,9]. Imbalanced problems are optimization tasks that have components with unequal contributions to the overall objective value. The CC techniques typically ignore such imbalanced contributions and treat all components equally. As a consequence, the least and most contributing components will be allocated with the same amount of the computational budget. This uniform resource allocation scheme results in an unwanted waste of valuable resources [10].

A few factors make dealing with imbalanced problems challenging. Firstly, potential contributions from the components are usually unknown to users and optimizers, especially in the case of black-box problems [11]. Secondly, the distributions of contributions are not necessarily always linearly correlated to the amounts of resources allocated from the total computational budget. For example, having a component with twice as large as the other one's contribution (in an arbitrary scale) does not necessarily means that it should receive a portion of resources two times bigger than the other component's budget. The CC algorithms must be equipped with an effective resource allocation mechanism to designate an appropriate portion of resources to solve each subproblem according to its estimated contributions.

Thirdly, a competent device is needed to maintain a right balance between the budget spent on the contribution learning (i.e., exploring different components several times) and optimization (i.e., exploiting the most contributing component found so far). Because of the uncertainty in the estimated contributions and also the fact that their distributions may change dramatically, these estimations should be updated frequently. For example, a component that is found to contribute a lot at the beginning stage of the search may contribute less at a later stage. Furthermore, contribution learning costs valuable resources that could be spent on the optimization of the most rewarding components. Therefore, preserving the balance between contribution learning and component optimization is vital in such a dynamic environment.

Recently, a few techniques have been proposed to enable the traditional CC to handle the imbalance issue [7,8,10,12,13]. These techniques, which we refer to as *contribution-aware* CCs, keep track of the components' contributions to the improvement of the overall objective value. These values are used to estimate the likelihood of receiving future improvement by optimizing each subproblem. Then, the algorithms allocate the remaining computational resources based on the predicted contributions. As a result, the highly contributing components are more likely to receive a bigger portion of the available resources.

In spite of the improvements that contribution-aware techniques demonstrate over the classic CC framework, they still suffer from some shortcomings. Firstly, these schemes adopted *ad hoc* heuristics to learn the contributions and allocate the resources (e.g., [7,12]). As a result, they seldom adapt to the dynamics of search process and their parameters are difficult to specify. Secondly, these techniques are often combinations of several other algorithms (e.g., [8,14]). This extra complexity brings lots of parameters into the framework such that the effects and contributions of each part to the overall outcome are difficult to identify. Finally, their performance is only examined on a limited number of test functions. For a better statistical analysis, a comprehensive set of large-scale imbalanced problems which covers more cases is required [9].

In this paper, we model the resource allocation problem as a dynamic multi-armed bandit problem [15]. We propose a modular framework to deal with the imbalanced contribution issue in the context of large-scale black-box optimization. Adopting the theoretically-sound bandit methods into this architecture, which we refer to as Bandit-Based Cooperative Coevolution (BBCC), has two merits. Firstly, we can leverage on the extensive literature on bandit problems and solvers to evaluate the proposed framework, develop several variants of it, and better understand its behavior. As a result, the theoretical and practical significance of BBCC will be much easier to appreciate than those previous *ad hoc* techniques. Secondly, the modularity of BBCC will provide users the flexibility to choose any appropriate decomposition strategies, reward functions, credit assignment formulations, bandit solvers, and optimization methods to plug into the framework. This feature will further facilitate sensible comparative studies on the topic of resource allocation.

This paper contributes to the field in several ways: Firstly, we formulate the problem of budget allocation in CC framework (when dealing with problems that composed of heterogeneous subfunctions) as a dynamic multi-armed-bandit problem. Secondly, we provide a generic, flexible, modular, and yet effective framework capable of solving the formulated problem. We show that the previously proposed contribution-aware CCs can be reformulated as particular instances of the BBCC framework. Thirdly, we conduct a comprehensive series of case studies to investigate the performance of an instance of the proposed framework in six different scenarios (covers 30 large-scale modular problems). Fourthly, we compare a basic implementation of the framework with three families of contribution-aware CCs (which include12 technique in total) as well as the winners of the past competitions on large-scale optimization. Finally, we conduct a brief sensitivity analysis on two generic parameters to show the stability of the performance of the framework in 16 different settings.

The rest of the paper is organized as follows: Section 2 presents background information about the CC framework, the issue of imbalanced contributions, and existing contribution-aware algorithms. Section 3 introduces the proposed framework and a few of its instances in detail. Section 4 provides a range of case studies, sensitivity analyses and comparative studies to demonstrate the capacities of BBCC. And finally, Section 5 concludes the paper and highlights some potential topics for future work.

2. Background

This section briefly reviews the CC framework, formally defines the imbalance contribution problem and summarizes the contribution-aware techniques that were previously proposed.

2.1. Cooperative Coevolution (CC) framework

To reduce the adverse influence of dimensionality on the performance of Evolutionary Algorithms (EAs), two major directions have been followed in the literature. One approach aims to enhance the traditional EAs by the aid of different techniques such as advanced initialization [16], intelligent sampling [17], memetic algorithms [18], adaptation [19] and hybridization [20]. The other approach attempts to decompose the problems into smaller pieces so that existing algorithms can handle them separately. The CC algorithms fall into the second category [21,22].

Typically, the original large-scale problem can be divided into smaller pieces called subproblems or components. This task can be done manually by a field expert or automatically using a decomposition (a.k.a. variable grouping) algorithm. In either case, a D-dimensional problem f is split into K subproblems:

$$f(\mathbf{x}) = \sum_{k=1}^{K} f_k(\mathbf{x}_{\mathcal{D}_k}), \tag{1}$$

Algorithm 1: CC(f, D, N)

```
1:  $t \leftarrow 0$                                          ▷ Epoch counter
2:  $\mathbf{X} \leftarrow$ initiate($N$, $D$)                ▷ Initialization
3:  $\mathbf{F}[:, t] \leftarrow$ evaluate($f$, $\mathbf{X}$)  ▷ Evaluation
4:  $\mathbf{v} \leftarrow$ initContextVector($\mathbf{X}$, $\mathbf{F}$)  ▷ Initial Context Vector
5:  $\mathcal{D} \leftarrow$ initGrouping($f$)                ▷ Static grouping
6:  $K \leftarrow$ size($\mathcal{D}$)                        ▷ Number of components
7:  repeat
8:      $t \leftarrow t + 1$                                  ▷ Increase epoch counter
9:      $k \leftarrow (t \mod K) + 1$                         ▷ Select next component
10:     $\mathbf{X}[:, \mathcal{D}[k]] \leftarrow$ optimize($f$, $\mathbf{X}[:, \mathcal{D}[k]]$, $\mathbf{v}$)  ▷ One epoch optimization
11:     $\mathbf{F}[:, t] \leftarrow$ evaluate($f$, $\mathbf{X}$)  ▷ Re-evaluate
12:     $\mathbf{v} \leftarrow$ updateContextVector($\mathbf{X}$, $\mathbf{F}$, $\mathbf{v}$)  ▷ Update Context Vector
13:     $\mathcal{D} \leftarrow$ updateGrouping($f$, $\mathcal{D}$)  ▷ Dynamic grouping
14: until termination() = True                             ▷ Termination
15: return $\mathbf{X}[\arg\min(\mathbf{F}[:, t]), :]$       ▷ Return final solution
```

where $\mathbf{x}$, $\mathbf{x}_{\mathcal{D}_k}$ and $\mathcal{D}_k$ are a complete (*i.e. D*-dimensional) solution, the kth subsolution, and the set of dimension indices form the kth component, respectively.

The task of decomposition may be performed only once at the beginning (i.e., static variable grouping) or several times during the course of optimization (i.e., dynamic grouping). It has been shown that some advanced decomposition techniques can accurately detect interactions between the variables and group them using a fraction of computational budget [23].

The effective number of components (i.e., K) varies from one problem to another. Optimization problems are generally categorized into three major groups based on the optimal value of K: *fully separable* problems when there is no interaction between any two variables ($K = D$), *partially separable* problems when the variables can be clustered into distinct groups of two or more variables ($1 < K < D$), and *fully non-separable* problems when no isolated cluster of variables can be found ($K = 1$).

When decomposition is done, each component f_k is treated as a separate optimization problem and tackled by an arbitrary EA (a.k.a. component optimizer). Since the exact formula or simulator of each single subproblem might not be available, the fitness of each potential solution is evaluated using some information from other components. For example, a so-called *context vector* may be constructed by merging the potential subsolutions of all subproblems. Therefore, to evaluate a potential subsolution $\mathbf{x}_{\mathcal{D}_k}$, it must be plugged into the context vector and the whole vector is evaluated by the original objective function.

Traditionally, each subproblem is optimized for one epoch at a time which consists of one or more optimization iteration. After each epoch, the context vector is updated. When all subproblems are optimized for one epoch, one optimization cycle is completed. This cycle is repeated until the termination criteria are met. At the end, the best subsolution for each subproblem is selected and merged with the other subsolutions to form a D-dimensional solution to the main problem.

Algorithm 1 shows the main steps of the round-robin CC framework. Here, N and T are the population size and the maximum number of epochs. Therefore, $\mathbf{X}$ and $\mathbf{F}$ are $N \times D$ and $N \times T$ tensors, respectively. For the sake of simplicity, we omit some of the control parameters (e.g., EA parameters in line 10) that the functions in Algorithm 1 can take.

The population initialization, problem decomposition and context vector creation all are performed in lines 1–4. The main loop starts at line 7. In line 9, the next component is selected in a round-robin fashion and optimized in line 10 for one epoch. Then, the population is re-evaluated and the context vector is updated (typically by merging the best subsolutions found so far). When a dynamic decomposition algorithm is adopted, the variable grouping should be updated in line 13. If the termination criteria are not satisfied yet, the main loop continues for another cycle. Otherwise, the final solution which is a single D-dimensional vector is returned. To form the final solution, we usually combine all best subsolutions (one per subproblem).

2.2. Imbalance contribution

It is very likely that the components of an optimization problem have unequal contributions to the overall objective value improvement [7]. In such cases, solving a subset of components may result in a considerably larger objective value improvement than the other subproblems given a fixed computational budget. Since the computational resources are usually limited in practice, it is more economical to discover the most contributing components and invest more resources in solving them, rather than treating all subproblems equally regardless of their contributions.

Broadly speaking, the main causes of contribution imbalance can be any or a combination of the followings: (1) differences in subproblems' fitness landscape, (2) unequal dimensionality and (3) non-uniform coefficients. More formally, an additive partially separable problem is considered as an imbalanced problem if it can be formulated as:

$$f(\mathbf{x}) = \sum_{k=1}^{K} c_k^{(t)} \cdot f_k(\mathbf{x}_{\mathcal{D}_k}) \tag{2}$$

and $\exists i,\ j \in \{1, \ldots, K\}$ where $i \neq j$ and $c_i^{(t)} \neq c_j^{(t)}$ or $f_i \neq f_j$ or $|\mathcal{D}_i| \neq |\mathcal{D}_j|$. In Eq. (2), $c_k^{(t)}$, f_k and $|\mathcal{D}_k|$ represent the coefficient at time t, fitness function, and dimensionality of the kth component, respectively. In the most recent large-scale benchmarking test suite [11], a combination of non-uniform dimension sizes and coefficients are introduced to challenge decomposition techniques.

2.3. Contribution-aware CC techniques

Recently, a number of online budget allocation techniques have been proposed to incorporate the estimated contributions of heterogeneous components into consideration. These algorithms can be categorized as hyper-heuristic methodologies (see [24,25]) as they generally operate on the space of the optimizers' segments rather than directly on the search space of the given large-scale optimization problem. As a result, the main challenge to be addressed here is not the dimensionality of the original search space but the underlying dynamism in the evolution of solution population and maintaining the "Exploration–Exploitation" balance in the budget assignment process.

The so-called Contribution-Based CC (CBCC) technique is the first published work that explicitly addressed the imbalanced component contributions in the context of large-scale optimization [7]. These algorithms adopt simple heuristics to find the most contributing component and optimize it more often than the others. The framework consists of two phases (*exploration* and *exploitation*) that are executed iteratively. The exploration phase is similar to one cycle of a round-robin CC where all components are optimized for a single epoch and the fitness values of the best solutions before and after that epoch are recorded. The differences between these values are used as an indicator of a component's contribution to improving the overall objective value. In the next exploitation phase, the component with the highest accumulated contribution receives more computational resources.

The exploitation phase of CBCC1 consists of one optimization epoch of the most contributing component [7]. In contrast, CBCC2 continuously optimizes this component until there is no more improvement [7]. These algorithms switch between exploration and exploitation phases until the termination criteria are met. In CBCC3 [12], the algorithm only performs the exploration phase with a probability that can be tuned by a user defined parameter p_t. In contrast with the earlier versions, CBCC3 does not accumulate the contributions from the very beginning to the end. Instead, it only takes the most recent improvement as the contribution indicator.

The CC with Adaptive Optimizer Iterations (CCAOI) is another contribution-aware technique [10]. It projects contributions to a value between zero and one using a formula inspired by the Gini's index of inequality. At the start of each cycle, CCAOI spends a fraction of the computational budget to optimize the subproblems and record their performances. At the end of each cycle, it updates the budget allocation for the next cycle and repeats the procedure. CCAOI's budget allocator normalizes the recent fitness improvements based on the calculated Gini's index and then assigns the resources proportional to the normalized contributions. The CCAOI has not been compared with other contribution-aware techniques on any well-known benchmark suite. Therefore, it remains unclear to what extent CCAOI improves upon CBCCs.

The Multilevel Optimization Framework Based on Variables Effect (MOFBVE) is one of the most recent attempts to address the imbalance problem [8]. The MOFBVE adopts a sensitivity analysis tool to measure the contributions of components at the early stage of the optimization. It ranks the components and groups them automatically based on their ranks using a conventional K-Means clustering algorithm. Then, the components of the groups with the highest average scores are selected to receive more computational resources than the others. In contrast with the other computational-aware techniques, MOFBVE does not update its budget plan during the optimization process.

The New CC Framework (CCFR) is one of the most recent attempts to tackle the imbalance problem [13]. Similar to CBCCs, the CCFR consists of exploration and exploitation phases, although in the exploration phase it uses a slightly different formula for contribution estimation. In exploitation phase, it selects the most rewarding component to optimize for one more epoch. When the estimated contributions of all components converge to the same level, CCFR moves to exploration phase. It also adopts a stagnation detection tool that sets the contribution of a component to zero when it detects that component is stagnated.

Among the contribution-aware techniques, CBCCs are the simplest models in terms of the number of extra parameters and computational complexity. For example, CBCC1 has no additional parameter and CBCC2 and CBCC3 have only one tuning parameter. These algorithms are easy to understand, implement and tune [12]. MOFBVE, on the other hand, is the most complex algorithm since it comprises a few modules (e.g., sensitivity analysis algorithms and clustering method). These elements force computational overhead to the algorithm and also make it difficult to track the contribution of each element to the overall performance of the algorithm.

The advantage of modular techniques like MOFBVE over the others is that their modules are well-known algorithms which are studied in many different scenarios [8]. Therefore, we already know the strengths and weaknesses of each part. However, CBCCs and CCAIO, are more *ad hoc* techniques and not designed based on sound theories. Indeed, they are working tools but there exist no theoretical proof or extensive experimental studies to support the proposed heuristics. An ideal framework is an algorithm that is as simple as CBCCs while at the same time it is built based on a more established theory such as bandit-based credit assignment schema. We elaborate these concepts and related techniques in the next section.

3. Bandit-based cooperative coevolution

In this paper, we model the computational budget allocation of the CC framework as a dynamic credit assignment problem. In a typical credit assignment problem, two or more actions with unequal benefits are competing against each other. The objective of the solver is to find a sequence of actions that maximizes the accumulated rewards in a given time horizon.

Algorithm 2: BBCC(f, D, N)

```
1:  $t \leftarrow 0$                                          ▷ Epoch counter
2:  $\mathbf{X} \leftarrow$ initiate($N, D$)                   ▷ Initialization
3:  $\mathbf{F}[:, t] \leftarrow$ evaluate($f, \mathbf{X}$)     ▷ Evaluation
4:  $\mathbf{v} \leftarrow$ initContextVector($\mathbf{X}, \mathbf{F}$)   ▷ Initial Context Vector
5:  $\mathcal{D} \leftarrow$ initComponentPool($f$)             ▷ Equation (3)
6:  $K \leftarrow$ size($\mathcal{D}$)                         ▷ Number of components
7:  $\boldsymbol{\mu}[:, t] \leftarrow$ initContribution($K, \infty$)   ▷ Initial contribution
8:  repeat
9:      $t \leftarrow t + 1$                                   ▷ Increase epoch counter
10:     $k \leftarrow$ componentSelector($\boldsymbol{\mu}$)    ▷ Equation (20)
11:     $\mathbf{X}[:, \mathcal{D}[k]] \leftarrow$ optimize($f, \mathbf{X}[:, \mathcal{D}[k]], \mathbf{v}$)   ▷ One epoch optimization
12:     $\mathbf{F}[:, t] \leftarrow$ evaluate($f, \mathbf{X}$)  ▷ Re-evaluate
13:     $\mathbf{v} \leftarrow$ updateContextVector($\mathbf{X}, \mathbf{F}, \mathbf{v}$)   ▷ Update Context Vector
14:     $\delta[k, t] \leftarrow$ improvementMeasure($\mathbf{F}$)   ▷ Equation (5)
15:     $\boldsymbol{\mu}[k, t] \leftarrow$ contributionEstimator($\boldsymbol{\delta}$)   ▷ Equation (9)
16:     $\{\mathcal{D}\} \leftarrow$ updateGrouping($f, \mathcal{D}$)   ▷ Equation (4)
17: until termination() = True                              ▷ Termination
18: return $\mathbf{X}$[arg min($\mathbf{F}[:, t]$), :]        ▷ Return final solution
```

A large number of credit assignment problems have been addressed by Multi-Armed Bandit (MAB) techniques [26]. A heuristic MAB solver samples the reward distributions by exploring the action space to estimate the expected long-term profit from execution of each action. At the same time, the solver should exploit the knowledge and fire the most rewarding action as much as possible to achieve the maximum accumulated reward. Therefore, maintaining a balance between action exploration (i.e., examining different actions) and exploitation (i.e., taking the best action repeatedly) is absolutely vital. Preserving this balance is especially critical in non-stationary problems where the distributions of the rewards change over time [27–29].

MAB techniques have been extensively used in the EA domain for adaptive operator selection [27,30,31], strategy selection in differential evolution [31,32], and component size adaptation in the CC framework [5]. In all of these instances, four essential building blocks are provided: the *action set* or the collection of all available options (e.g., EA operators), *reward function* which measures the immediate benefit of taking a particular action, *credit function* which translates the immediate rewards to long-term utility of taking an action, and *strategy* which selects the next actions based on their estimated credits. In what follows, we explain how the resource allocation problem in CC can be formulated as an MAB problem, and how this formulation is implemented in our Bandit-Based CC (BBCC) framework.

3.1. The BBCC framework

Analogous to MAB agents, the BBCC framework consists of four fundamental building blocks: the *component pool* or the set of all subproblems, *improvement measure* which reflects the immediate fluctuations in the objective value after optimizing a particular component, *contribution estimator* which predicts the contribution of optimizing a component to improving the quality of final solution, and *component selector* which selects the next component to be optimized based on the estimated contributions (see Fig. 1).

A simplified pseudocode of the BBCC framework is presented in Algorithm 2. As lines 1–3 state, BBCC starts with initializing the population and building the context vector. Then, the component pool $\mathcal{D}$ is formed (see Section 3.2). The objective of BBCC is to select and optimize one of the K components at a time such that the overall fitness improvement is maximized.

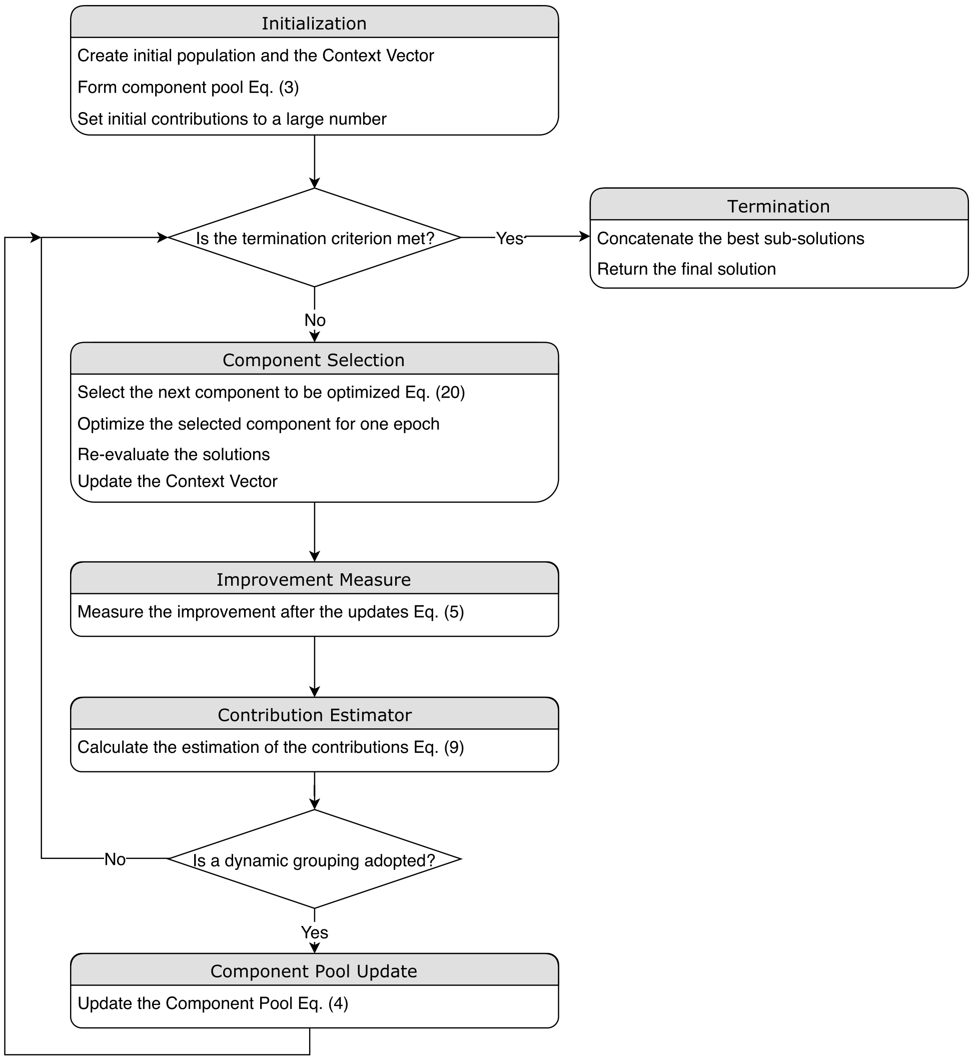
Initialization
Create initial population and the Context Vector
Form component pool Eq. (3)
Set initial contributions to a large number
Is the termination criterion met?
Yes
Termination
Concatenate the best sub-solutions
Return the final solution
No
Component Selection
Select the next component to be optimized Eq. (20)
Optimize the selected component for one epoch
Re-evaluate the solutions
Update the Context Vector
Improvement Measure
Measure the improvement after the updates Eq. (5)
Contribution Estimator
Calculate the estimation of the contributions Eq. (9)
No
Is a dynamic grouping adopted?
Yes
Component Pool Update
Update the Component Pool Eq. (4)

Fig. 1. The flowchart of the BBCC framework.

In line 6, BBCC initiates the matrix $\boldsymbol{\mu}$ which stores the estimated contributions. When there is no prior knowledge about the contributions of the components, it is recommended to initialize $\boldsymbol{\mu}$ with an effectively large value to ensure all components are examined at least once.

The main loop at lines 8–17 starts by selecting a subproblem using a component selection algorithm (see Section 3.5). The selected component is then optimized using an arbitrary optimizer for one epoch or δ_t iterations. The main challenge here is to maintain a good balance between exploring all components to improve/update the estimations and optimizing the most contributing component as many times as possible.

In line 14, BBCC assesses the effectiveness of the optimization of the recently selected component using an improvement measure (see Section 3.3). Next, BBCC updates the contribution estimations based on the recent measured improvement (i.e., $\boldsymbol{\delta}$) and other factors, e.g., the number of times the component is optimized so far or the magnitude of the remaining budget (see Section 3.4). This information is recorded to be used by the component selector at the next round.

All steps inside the loop are repeated until a termination criterion is met (line 17). At this point, the solution with the highest fitness value (or minimum objective value in minimization tasks) is returned as the ultimate solution (line 18). To preserve simplicity, the control parameters e.g., δ_f in Eq. (8) or ϵ in Eq. (21) are excluded.

In the following we discuss the BBCC building blocks in more details.

3.2. Component pool

The component pool is conceptually similar to the action set in the MAB taxonomy. Assume a D-dimensional optimization problem is decomposed into K distinct components, either manually or automatically:

$$\mathcal{D} = g(f) \tag{3}$$

where the component pool $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$ is essentially the superset of all subproblems of f that formed by the decomposition technique g. Here, $\mathcal{D}_k$ indicates the indices of the decision variables that form the kth component. Notably, any decomposition technique can be acquired in Eq. (3) to initiate the component pool. For example, if a dynamic decomposition technique is adopted, the component pool should be updated every few iterations:

$$\mathcal{D}^{(t)} = g(f, \mathcal{D}^{(t-1)}) \tag{4}$$

3.3. Improvement measure

In the absence of *a priori* knowledge about component contributions, BBCC needs a tool to measure the goodness of each action.

These metrics are related to the concept of immediate or practical reward in the MAB literature. In our particular problem, the goodness of optimizing a component depends on its contribution to improving the objective value of the final solution. Therefore, we define the improvement measure as a function of the fitness fluctuations when a component is being optimized. In the most general form, we define

$$\delta_k^{(t)} = \Delta(\mathbf{F}^{(t)}), \tag{5}$$

where $\mathbf{F}^{(t)}$ is an $N \times t$ matrix that contains the complete history of the fitness values of all candidate solutions till epoch t. Note that in Eq. (5), we implicitly assume that kth component is optimized at epoch $t-1$. Since only one component is optimized at a time, the $\delta_i^{(t)}$ for all $i \neq k$ is zero.

Usually, we do not need to record all the fitness values during the optimization to measure the improvement of a component. For example, we can assess the gained improvement of minimizing the kth component at epoch t only based on the fitness value of the best solutions before and after the optimization of that component:

$$\delta_k^{(t)} = \overset{*}{f}^{(t-1)} - \overset{*}{f}^{(t)}, \tag{6}$$

where $\overset{*}{f}^{(t-1)}$ and $\overset{*}{f}^{(t)}$ are the objective values of the best solution before and after optimizing the kth component for one epoch. In maximization tasks, the order of the terms in the above definition should be reversed. Note that each single epoch consists of one or more optimization iterations (i.e., EA generations). We discuss the role of epoch length δ_t in Section 3.6.3.

Generally speaking, the magnitudes of fitness values vary significantly as optimization progresses. For example, some δ_k may take larger values at the early epochs and smaller values towards the end. In such cases, using the simple difference may result in over-emphasizing the earlier trials rather than the recent observed improvements. To lessen this effect, one can simply *normalize* the fitness improvement as:

$$\delta_k^{(t)} = \frac{\overset{*}{f}^{(t-1)} - \overset{*}{f}^{(t)}}{\overset{*}{f}^{(t-1)} + \varepsilon}, \tag{7}$$

where ε is a small constant that is added to avoid division by zero. Assuming $\overset{*}{f}^{(t)}$ is always positive, Eq. (7) guarantees that $\forall t\ \forall k,\ 0 \leq \delta_k(t) < 1$.

The *quantization* is another approach to deal with the significant variations in objective values during optimization. For example, it is possible to binarize the fitness improvement as:

$$\delta_k^{(t)} = \begin{cases} 1 & \text{if } \overset{*}{f}^{(t-1)} - \overset{*}{f}^{(t)} > \delta_f \\ 0 & \text{otherwise} \end{cases}, \tag{8}$$

where δ_f is a tuning parameter that can take absolute (e.g., 10^{-8}) or relative (e.g., $\overset{*}{f}^{(t)}/100$) values to control the sensitivity of the measurement on the marginal improvements.

The generality of BBCC framework allows us to adopt any of the aforementioned improvement measures, as well as many others. Practitioners can choose the most effective form of improvement measure according to properties such as the sensitivity to small improvements (e.g., Eq. (6)) or robustness in noisy environments (e.g., Eq. (8)).

3.4. Contribution estimator

The contribution estimation in the BBCC framework is similar to the utility or value function in Reinforcement Learning [33]. The objective of this module is to translate the measured improvements (as immediate rewards) to estimated contributions (as long-term utility). Therefore, the general form of the contribution estimator is:

$$\mu_k^{(t)} = M(\boldsymbol{\delta}^{(t)}), \tag{9}$$

where $\boldsymbol{\delta}^{(t)}$ denotes the measured improvements of all components till epoch t.

In general, this task can be done in three ways: *contribution value* estimation, *contribution rank* estimation, or a combination of both. The first approach generalizes the past measured improvements to forecast the contribution values in the next optimization epochs. These projected values will be used to prioritize the optimization of each component. In contrast to the value estimation, the rank-based approach directly sorts the components according to their past performance without any explicit calculation of contribution magnitudes. The hybrid techniques, as their name suggests, are combinations of the other two categories. These approaches are discussed below.

3.4.1. Contribution value estimation

In a stationary situation, the expected contribution of the kth component can be approximated as the mean of the past fitness improvements:

$$\mu_k^{(t)} = \frac{1}{n_k}\sum_{i=1}^{t}\delta_k^{(i)} = \frac{1}{|\boldsymbol{\lambda}_k^{(t)}|}\sum_{i\in\boldsymbol{\lambda}_k^{(t)}}\delta_k^{(i)}, \tag{10}$$

where $\boldsymbol{\lambda}_k^{(t)}$ denotes a set of length n_k that contains all epoch indices that the kth component has been optimized until epoch t. As mentioned earlier, for all the other epochs (i.e., $i \notin \boldsymbol{\lambda}_k^{(t)}$) the improvements are assumed to be zero. Based on Central Limit Theorem, as t tends toward infinity, $\mu_k^{(t)}$ approaches its true value if all δ_k are drown from a fixed distribution.

Since the distribution of δ_k may change multiple times during optimization, the stationary assumption in Eq. (10) becomes invalid. For example, it is very likely that an optimizer will be trapped into a local optimum or stagnate during the search process. In any of these cases, the magnitude of the recent δ_k drops dramatically. Therefore, Eq. (10) becomes inefficient as it takes a long time to adapt to the dynamics, especially for large n_k.

This issue can be addressed in two ways: *passive* or *active*. In the passive approach, we assume that the improvement distributions are stationary at least in a bounded time window. By holding this assumption, we can substitute $\boldsymbol{\lambda}_k^{(t)}$ by $\overset{L}{\boldsymbol{\lambda}}_k^{(t)} = \langle \lambda_1, \ldots, \lambda_L \rangle$ which only contains the epoch indices of the last L non-zero improvements of kth component prior to epoch t. Therefore:

$$\mu_k^{(t)} = \frac{1}{L}\sum_{i=1}^{L}\delta_k^{(\lambda_i)}. \tag{11}$$

The window length L in Eq. (11) can be kept constant or tuned adaptively. To have a smoother sliding window, we could adopt a weighted averaging method:

$$\mu_k^{(t)} = \frac{1}{L}\sum_{i=1}^{L} W_i \cdot \delta_k^{(\lambda_i)}, \tag{12}$$

where $\forall i,\ W_i \in [0, 1]$ and $\sum_{i=1}^{L} W_i = 1$. Usually the larger coefficients are used for the most recent observations. A similar approach is to adopt a forgetting factor to decrease the memory requirements:

$$\mu_k^{(t)} = \alpha \cdot \delta_k^{(t)} + (1-\alpha)\cdot \mu_k^{(t-1)}, \tag{13}$$

where $\alpha \in (0, 1]$ controls the length of the memory horizon such that larger values for α will result in a shorter memory.

In the active approach, we assume that the improvement distributions are stationary unless we find an evidence that proves otherwise. In other words, an active estimator calculates contributions based on all observed improvements, while at the same time compares the current statistics of the improvement with the previous records. Once a significant change has been detected, we clear all records and build the model from scratch. This approach is very helpful when a dynamic decomposition algorithm is adopted [34]. In the EA literature, some statistical change detection tests such as Page–Hinkley have been used in similar situations [31].

3.4.2. *Rank-based estimations*

There are multiple ways to rank components based on their previous improvements. Let $\overset{L}{\sigma}{}_k^{(t)}$ denote the sum of the last L improvements gained by optimizing the kth component:

$$\overset{L}{\sigma}{}_k^{(t)} = \sum_{i=1}^{L} \delta_k^{(\lambda_i)}, \tag{14}$$

and $\gamma_k^{(t)}$ is its position after we sort all components based on the summations in a descending order. Now, the score of kth component can be defined as:

$$\mu_k^{(t)} = \frac{\alpha^{\gamma_k^{(t)}}}{\sum_{i=1}^{K} \alpha^{\gamma_i^{(t)}}}, \tag{15}$$

where $\alpha \in (0, 1)$ is a decay factor. Eq. (15) resembles the famous *normalized exponential function* which have been widely used in statistical analysis and machine learning representing categorical distributions. Since the calculated scores μ_k are in the range $(0, 1)$ that add up to 1, they can be interpreted as the probabilities of contributing rather than the contribution values.

Another possible rank-based definition that can be adopted is the well-known *sum-of-ranks* that works as follows [27,31]. Let the last L improvements of all components be sorted in descending order and $\overset{L}{\boldsymbol{\gamma}}{}^{(t)} = \langle \gamma_1, \ldots, \gamma_L \rangle$ contains all the ranks since $t - L$ till t. Then, the sum-of-ranks score is defined as:

$$\mu_k^{(t)} = \frac{\sum_{i=1}^{L} w_{k,i} \cdot \alpha^{\gamma_i}}{\sum_{j=1}^{K} \sum_{i=1}^{L} w_{j,i} \cdot \alpha^{\gamma_i}}, \tag{16}$$

$$\text{where } w_{k,i} = \begin{cases} L - \gamma_i & \text{if } \gamma_i \text{ is associated with } k\text{th component} \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

Eq. (17), guarantees that components with better ranks (or smaller γ_k) are multiplied by larger coefficients. Note that there is a pronounced difference between Eqs. (15) and (16): In the former equation we only consider the most recent ranking of the components, whereas in the latter a weighted sum of the last L ranks is used.

3.4.3. *Hybrid estimators*

By combining the aforementioned approaches one can have the best of two worlds. The main disadvantage of contribution value estimation is that it can overestimate the contribution of some components if the improvement range is too wide. On the other hand, a very small difference in recorded improvement of two or more components has the potential to change their orders and dramatically affect the performance of rank-based techniques. Therefore, incorporating the magnitude of the observed improvements into the rank-based estimator can help us to differentiate components while minimizing the risk of over/underestimations.

The simplest hybridization is to define a function based on the improvement values and plug it into the definition of a rank-based estimator. For example, one can modify Eq. (15) to:

$$\mu_k^{(t)} = \frac{\overset{L}{\sigma}{}_k^{(t)} \cdot \alpha^{\gamma_k^{(t)}}}{\sum_{i=1}^{K} \overset{L}{\sigma}{}_i^{(t)} \cdot \alpha^{\gamma_i^{(t)}}}, \tag{18}$$

where $\overset{L}{\sigma}{}_k^{(t)}$ is the sum of the last L improvements of kth component (defined in Eq. (14)) and $\gamma_k^{(t)}$ is its rank among all components (used in Eq. (15)). It is worth noting that any of the aforementioned contribution estimation methods can be adopted in BBCC. In addition, it is possible to make a combination of them to maintain better accuracy in very noisy or dynamic situations.

3.5. *Component selector*

The main goal of the component selector, which is conceptually close to strategy in MAB literature, is to select a component to be optimized at the next epoch based on the estimated contributions. Note that the selection algorithm should maintain a good balance between exploitation (i.e., allocating more resources to the most contributing components) and exploration (i.e., trying different components to improve/update the estimations).

Several techniques have been developed in the MAB literature for sustaining the exploration–exploitation balance [15,31, 35]. With no exception, any of these algorithms can be adopted in BBCC as a component selector. We can define a stochastic selector (e.g., *semi-uniform* and *probability-based* selectors) as:

$$p_k^{(t+1)} = S(\boldsymbol{\mu}^{(t)}), \tag{19}$$

where $p_k^{(t+1)}$ is the probability of the kth component being selected for the next epoch and $\boldsymbol{\mu}^{(t)}$ denotes the estimated contributions of all components till epoch t. The deterministic selectors (e.g., *interval-based* selectors) can be defined as:

$$k^{(t+1)} = S(\boldsymbol{\mu}^{(t)}), \tag{20}$$

where $k^{(t+1)}$ denotes the component that is selected to be optimized in the next epoch. In the following, we briefly review some representative approaches from each category.

3.5.1. *Semi-uniform component selectors*

In the exploration phase, semi-uniform techniques uniformly allocate a portion of the budget to all components regardless of their contributions. In the exploitation phase, these algorithms spend a part of the remaining resources to the component with the highest estimated contribution. ϵ-greedy is a famous example of semi-uniform techniques:

$$k^{(t+1)} = \begin{cases} \overset{*}{k} & \text{if } \mathtt{rand}(1) \geq \epsilon \\ \lceil \mathtt{rand}(1) \cdot K \rceil & \text{otherwise} \end{cases}, \tag{21}$$

where $\overset{*}{k}{}^{(t)}$ denotes the component with the highest estimated contribution at epoch t, $\mathtt{rand}(1)$ generates a random number in range $[0, 1]$ and $\lceil . \rceil$ denotes the ceiling function. In Eq. (21), the $\epsilon \in (0, 1)$ can be either constant or variable. As the above equation states, decreasing ϵ will result in a more greedy selection. Some of other variants of semi-uniform techniques e.g., ϵ-first, GreedyMix and LeastTaken are discussed in [35].

3.5.2. *Probability-based component selectors*

To avoid uniform budget distribution over all non-optimal components, probability-based strategies map each $\mu_k^{(t)}$ to a different probability of being selected. These probabilities are usually proportional to the expected contributions. SoftMax (or Boltzmann Exploration) is a widely-used technique from this category [5]. In

SoftMax, the probability of selecting the kth component in the next epoch is:

$$p_k^{(t+1)} = \frac{e^{\frac{\mu_k^{(t)}}{\tau}}}{\sum_{i=1}^{K} e^{\frac{\mu_i^{(t)}}{\tau}}}, \tag{22}$$

where τ is a temperature parameter that controls exploration–exploitation balance. For $\tau = 0$, Softmax only selects the most contributing component, while it picks all components uniformly when τ tends towards infinity.

Some probability-based techniques e.g., Probability Matching and Adaptive Pursuit algorithms explicitly guarantee the minimum and maximum likelihoods that a component can be selected [36]. Other forms of these techniques, e.g., Adaptive Probability Matching, Power Probability Matching, Exp3 and SoftMix are explained in [35,36].

3.5.3. Interval-based component selectors

These algorithms compute the confidence interval which indicates to what extent we are confident of the accuracy of estimated contributions. The well-known Upper Confidence Bound (UCB) algorithms [30] follow the rule of *'optimism in the face of uncertainty'* [31]. This means the components that are less explored so far have a higher chance of being selected even if their expected contributions are not very high. For example, UCB1 selects the next component according to the following rule:

$$k^{(t+1)} = \underset{1\le i\le K}{\arg\max}\left(\mu_i^{(t)} + \alpha \cdot \sqrt{\frac{2\ln(t)}{n_i^{(t)}}}\right), \tag{23}$$

where α is a scaling factor that controls the trade-off between exploration and exploitation. As the equation suggests, by decreasing α the algorithm greedily selects the most contributive component more often than the less-explored components that might have some undiscovered potentials. POKER (Price Of Knowledge and Estimated Reward) is another example of interval-based strategies which explicitly takes the horizon (i.e., the maximum number of objective function calls) into account. POKER and other variants of UCB are described in [15,35].

There exist extensive literature about the advantages and disadvantages of these techniques in different applications [15,35,36]. However, there is no single study that compares them in the context of computational budget assignment when dealing with imbalanced problems.

3.6. Notes and discussions

Before proceeding to the empirical studies, there are a few special topics worth highlighting.

3.6.1. BBCC and other contribution-aware techniques

As mentioned earlier, BBCC is a general framework rather than a single specific technique. This means, one can easily create special instances of the BBCC framework by adopting different techniques for each of its modules. Indeed, it can be shown that all available contribution-aware CCs are special cases of the BBCC framework.

For example, the normalized improvement in Eq. (7) is used as the improvement measure in all variants of CBCC. Conversely, CCFR uses the absolute difference between the fitness values of the best solutions before and after epoch t as the improvement measure (i.e., $|\overset{*}{f}{}^{(t-1)} - \overset{*}{f}{}^{(t)}|$).

The contribution estimator of CBCC1 and CBCC2 is simply the average of improvement over all epochs as presented in Eq. (10). CBCC3, however, only considers the last recorded improvement which is a special case of Eq. (11) with $L = 1$. In CCFR, contributions are calculated according to Eq. (13) where $\alpha = 0.5$. It also uses a stagnation detection technique to reset the estimations if it finds a stagnant subpopulation. In fact, this is an active approach to deal with the dynamics in the contributions that was discussed in Section 3.4.

CCFR and all variants of CBCC adopt the same semi-uniform component selection strategy. In their exploitation phase, they always select the most contributing component: $k^{(t+1)} = \arg\max \mu_i^{(t)}$. In contrast, all components are optimized in a round-robin fashion in the exploration phase: $k^{(t+1)} = k^{(t)} + 1$. The main difference between these algorithms is the rules they determine to switch between exploration and exploitation phases (see Section 2.3).

The so-called CCEA-MAB can be identified as another instance of BBCC although the notion of 'species' as defined in [34] is different from what we refer to as separable components or subfunctions. In CCEA-MAB, a pool of species is randomly formed from the current population and underperforming species may be replaced by a fresh randomly generated species (note that the species are not independent by design). The Eq. (8) with $\delta_f = 0$ is used in CCEA-MAB as the binary improvement measure. It also adopts a rank-based contribution estimator (namely modified Area-Under-Curve based on [37]) and an interval-based species selector similar to Eq. (23). In CCEA-MAB the epoch length δ_t is fixed to 1.

3.6.2. Dynamic grouping/decomposition

As clearly stated in Algorithm 2, it is possible to adopt a dynamic decomposition (e.g., [5]) in the BBCC framework. In this case, BBCC should update $\boldsymbol{\delta}$ and $\boldsymbol{\mu}$ matrices whenever the indices of a component (i.e., $\mathcal{D}_k$) have been changed, a component is removed or new components are created [34]. It is also possible to inherent these values from parent(s) to the child component(s) if the new components are created by splitting or merging actions. Obviously, too frequent changes in the decomposition adversely affect the learning capability of contribution-aware CCs, including BBCC instances, as the recorded information becomes inoperative very soon.

3.6.3. Epoch length

Although all CC techniques should tune the epoch length (i.e., δ_t value) to achieve the maximum performance, this parameter could have more influence on the contribution-aware CC techniques, including BBCC instances. Choosing a large value for δ_t has advantages such as improving the robustness of the measured improvements in the face of fitness fluctuations as well as decreasing the overhead costs of component switching. On the other hand, a too large δ_t may increase the delay in response to the dynamics of measured improvements. Since the number of function evaluations is fixed, as the epoch length increases, the number of epochs must be decreased. This would decrease the exploration capability of the algorithms. In Section 4.3 we will show that for some implementations of BBCC, there are a range of parameter settings for which the performance of the algorithm is less affected by δ_t.

4. Experiments

This section consists of six parts: Section 4.1 provides the details of the studied techniques and performance metrics. Section 4.2 includes six case studies that deeply investigate the behavior of a BBCC instance called BBCC1 (as discussed in Section 4.1.1) in different scenarios covering 30 large-scale optimization problems in total. This comprehensive set of case studies help us evaluate the performance of BBCC1 in dealing with a wide range of imbalance sources and levels. Then, we perform a brief sensitivity analysis on BBCC1 generic parameters: population size and epoch length in Section 4.3. A set of 16 different settings has been examined to compare BBCC1 with a CC counterpart on 30 large-scale minimization functions. The objective of this part is to measure the robustness of the framework as the parameter values change. Next, we conduct a comparison study between BBCC1 and two variants of round-robin CCs on partially separable problems from CEC'13 LSGO benchmark functions collection (see Section 4.4). Then, we implement a set of comparative studies between BBCC1 and other contribution-aware CCs in Section 4.5. The primary objective here is to study the relative performance of the proposed framework in comparison with the available budget allocation techniques. Finally, we compare BBCC1 and the winners of the past competitions on large-scale optimization problems to verify that even a basic implementation of the framework can provide competitive outcome (see Section 4.6).

We use the well-known CEC'13 LSGO benchmark functions to compare BBCC with other available techniques. This test suite is the most recent and widely used large-scale optimization benchmark in this domain. However, only 8 of 15 functions are partially separable with some degrees of imbalanced contributions. In addition, the non-uniform contributions introduced in these 8 functions are set arbitrarily which does not cover all common cases. Therefore, we expand this set to 30 large-scale partially separable imbalanced functions to carry out the case studies and sensitivity analysis. Since these problems were not available when the previous algorithms were proposed, it is not possible to compare them against the other researchers' results. Instead, we use the widely studied CEC'13 LSGO benchmarks to compare BBCC instances with other methods.

Algorithm 3: BBCC1($f, D, N, \mathbf{l}, \mathbf{u}, \delta_t$)

```
1:  $t \leftarrow 0$                                                        ▷ Epoch counter
2:  $\mathbf{X} \leftarrow \mathtt{rand}(N, D) \circ (\mathbf{u} - \mathbf{l}) + \mathbf{l}$       ▷ Initialization
3:  $\mathbf{F}[:, t] \leftarrow \mathtt{evaluate}(f, \mathbf{X})$                  ▷ Evaluation
4:  $\overset{*}{i}_t \leftarrow \mathtt{argmin}(\mathbf{F}[:, t])$                  ▷ Find best solution
5:  $\overset{*}{f}_t \leftarrow \mathbf{F}[\overset{*}{i}, t]$                      ▷ Best objective value
6:  $\mathbf{v} \leftarrow \mathbf{X}[\overset{*}{i}, :]$                            ▷ Initial Context Vector
7:  $\mathcal{D} \leftarrow \mathtt{idealGrouping}(f)$                         ▷ Decomposition Eq. (3)
8:  $K \leftarrow \mathtt{size}(\mathcal{D})$                                  ▷ Number of components
9:  $\mathbf{n}[1{:}K] \leftarrow 0$
10: $\boldsymbol{\mu}[1{:}K, t] \leftarrow \infty$                              ▷ Initial contribution
11: repeat
12:     $t \leftarrow t + 1$                                                ▷ Increase epoch counter
13:     if $\mathtt{rand}(1) \geq \epsilon$ then
14:         $k \leftarrow \mathtt{argmax}(\boldsymbol{\mu}[:, t])$                 ▷ Exploitation Eq. (21)
15:     else
16:         $k \leftarrow \mathtt{ceil}(\mathtt{rand}(1) \cdot K)$                ▷ Exploration Eq. (21)
17:     end if
18:     $\mathbf{n}[k] \leftarrow \mathbf{n}[k] + 1$
19:     $\mathbf{X}[:, \mathcal{D}[k]] \leftarrow \mathtt{DE}(f, \mathbf{X}[:, \mathcal{D}[k]], \mathbf{v}, \delta_t)$   ▷ One epoch DE
20:     $\mathbf{F}[:, t] \leftarrow f(\mathbf{X})$                              ▷ Re-evaluate
21:     $\overset{*}{i}_t \leftarrow \mathtt{argmin}(\mathbf{F}[:, t])$              ▷ Find best solution
22:     $\mathbf{v}[\mathcal{D}[k]] \leftarrow \mathbf{X}[\overset{*}{i}, \mathcal{D}[k]]$     ▷ Update Context Vector
23:     $\overset{*}{f}_{t-1} \leftarrow \overset{*}{f}_t$                          ▷ Old best objective value
24:     $\overset{*}{f}_t \leftarrow \mathbf{F}[\overset{*}{i}, t]$                   ▷ New best objective value
25:     $\boldsymbol{\delta}[k, t] \leftarrow (\overset{*}{f}_{t-1} - \overset{*}{f}_t)/(\overset{*}{f}_{t-1} + 10^{-8})$   ▷ Equation (7)
26:     $\boldsymbol{\mu}[k, t] \leftarrow \mathtt{sum}(\boldsymbol{\delta}[k, :])/\mathbf{n}[k]$   ▷ Equation (10)
27: until $t \cdot N \cdot \delta_t \geq 3000 \cdot D$                        ▷ Termination
28: return $\mathbf{X}[\mathtt{argmin}(\mathbf{F}[:, t]), :]$                  ▷ Return final solution
```

4.1. Experiments setup

In all experiments we chose to study the simplest possible implementation of BBCC (see Section 4.1.1). This helps us to avoid unnecessary complications and minimize the effects of the factors that are out of the scope of this study. This paper shows that even a simple implementation of BBCC can provide very competitive results. Thus, the main objective of the following experimental studies is to provide a proof of concept and show the potentials of the framework.

4.1.1. BBCC1: The simplest BBCC implementation

Algorithm 3 shows the pseudocode of BBCC1. A normalized fitness improvement in Eq. (7) with epoch length $\delta_t = 50$ is used to measure the improvements after optimizing each component. As in Eq. (10), the basic average of all improvements is adopted for contribution estimation. We adopt the simplest component selector here, ϵ-greedy in Eqs. (21). The ϵ value is set to 0.1 which means the most rewarding component will be selected with at least 90% chance. The remaining 10% of the budget will be spent uniformly on all components regardless of their estimated contribution.

BBCC1 adopts the ideal grouping (e.g., [7]) as the decomposition algorithm and DE/rand/1/bin as the optimizer. The DE parameters are set based on the advised values in [38]. Therefore, the population size $N = 50$, weighting factor $F = 0.5$, and crossover rate $CR = 0.9$ are used for the experiments.

4.1.2. Performance metrics

In this study, we consider two aspects when analyzing the performance of the algorithms: components *selection distribution* and final *solution quality*. These numbers are recorded for 51 independent runs of each algorithm where the maximum number of objective function evaluation for each run is fixed to 3,000,000.

For the case studies, we analyze the selection distributions to study the performance of BBCC1 in identifying the most contributing component and effective budget allocation. These distributions are calculated based on the number of times each component is selected during the optimization. These statistics, which are presented using a variety of diagrams, indicate how precisely BBCC1 can find the most contributing component, and how efficient the exploration–exploitation balance is maintained.

We assess the final solution quality in terms of the objective value of the best solution found in each single run. We use this metric especially for sensitivity analysis and comparative studies. In addition to the fitness mean and standard deviation of the final solutions, *nWins* scores, *Win–Tie–Loss* values and Friedman rankings are also reported in the comparison tables [39].

Each nWins value indicates the number of times BBCC1 significantly improves other algorithms, subtracting the number of times it performs significantly worse than its competitors. Therefore, nWins scores take values in the range $\{-N_a, +N_a\}$ when BBCC1 is compared with N_a other algorithms. The values in each Win–Tie–Loss triple respectively indicates the number of times BBCC1 performs significantly better than, statistically similar to, or significantly worse than a particular competitor. For both metrics, the non-parametric Mann–Whitney U test (a.k.a. Wilcoxon rank-sum test) with 95% confidence interval is adopted as the significance test. Any comparison with a p-value greater than or equal to 0.05 is considered as statistically similar performances. In cases where the detailed results of an algorithm were not available, we substitute Mann–Whitney U test with the standard two-sided t-test. Although we cannot guarantee that the result distributions satisfy the t-test assumptions, at least it allows us to perform some significant test in the absence of detailed results.

For further statistical analyses, we also perform Friedman and Quade significance tests [39]. For each comparison study, we provide Friedman ranks and orders, as well as Friedman and Quade p-values. When the overall p-values are smaller than 0.05, which signals a significant difference between the compared algorithms, we perform post-hoc tests to find the pairs that are responsible for that large difference. In this case, we adjust p-values using the famous Holm's and Bonferroni's correction techniques to control family-wise error rate [39].

Table 1
Case study problems: coefficients (C_k) and components' sizes ($|\mathcal{D}_k|$). The symbol k indicates the subproblem index.

Problems	C_k	$\|\mathcal{D}_k\|$
f_1–f_5	1	100, . . . , 100
f_6–f_{10}	2^k	100, . . . , 100
f_{11}–f_{15}	10^k	100, . . . , 100
f_{16}–f_{20}	1	50, 50, 50, 100, 100, 100, 100, 150, 150, 150
f_{21}–f_{25}	1	25, 25, 50, 50, 75, 75, 100, 150, 200, 250
f_{26}–f_{30}	1	100, . . . , 100

Table 2
Case study problems: base functions. The $f_{i-j}(\mathcal{D}_k)$ indicates the kth subproblems of problems $\{i, \ldots, j\}$.

Base	Problem (components)
Elliptic	$f_1, f_6, f_{11}, f_{16}, f_{21}, f_{26-29}(\mathcal{D}_3, \mathcal{D}_4)$
Rastrigin	$f_2, f_7, f_{12}, f_{17}, f_{22}, f_{26-28}(\mathcal{D}_5, \mathcal{D}_6), f_{30}(\mathcal{D}_3, \mathcal{D}_4)$
Ackley	$f_3, f_8, f_{13}, f_{18}, f_{23}, f_{26-27}(\mathcal{D}_7, \mathcal{D}_8), f_{29-30}(\mathcal{D}_5, \mathcal{D}_6)$
Schwefel 1.2	$f_4, f_9, f_{14}, f_{19}, f_{24}, f_{26}(\mathcal{D}_9, \mathcal{D}_{10}), f_{28-30}(\mathcal{D}_7, \mathcal{D}_8)$
Rosenbrock	$f_5, f_{10}, f_{15}, f_{20}, f_{25}, f_{27-30}(\mathcal{D}_9, \mathcal{D}_{10})$
Sphere	$f_{26-30}(\mathcal{D}_1, \mathcal{D}_2)$

4.2. Case studies

In this part, we conduct a series of experiments to study the performance of BBCC1 component selection mechanism (see Algorithm 3). In particular, we analyze the effectiveness of component selector in four different scenarios: uniform contributions, unequal component coefficients, nonuniform component sizes, and problems with mixed landscapes (i.e., heterogeneous components). We deliberately choose these scenarios to investigate the effect of each imbalance source in a controlled environment.

For these case studies, we designed a set of 1000-dimensional problems each of which has exactly 10 partially separable components. We fixed the dimensionality and number of components to minimize the effect of these factors on the outcome of the studies.

The chosen problems fall into six categories each containing five problems. All problems within a category have the same source and level of imbalance, while their base functions differ. Having five different base functions in each category helps us to reduce the chance of biased conclusion towards a specific landscape.

Since the order of the base functions is fixed, the problems which their indices modulo 5 are equal have the same base function. This helps us to study the impact of an imbalance source and level on each single base function. For example, by comparing f_5, f_{10} and f_{15}, we can investigate the effects of three levels of unequal coefficients on Rosenbrock's problems.

Table 1 provides the subproblem sizes and coefficients. The symbol k in this table indicates component indices. As mentioned in the table, f_1–f_5 are fully balanced problems, f_6–f_{15} have unequal coefficients, and f_{16}–f_{25} consist of components with nonuniform sizes. The problems in the last category (f_{26}–f_{30}) consist of components with equal sizes and coefficients, but different landscapes.

Table 2 summarizes the base functions, where the numbers in the parenthesis indicate component indices. For f_1–f_{25} the component indices are omitted as all components of each of them have the same base function. Note that all the base functions are borrowed from CEC'13 LSGO benchmark suite, and shifted and rotated using the same algorithms. Furthermore, special transformations such as ill-conditioning, symmetry breaking, and adding smooth local irregularities are employed based on the suggestions in [40]. Therefore, the interested readers can repeat the experiments by following the available guidelines. More details about the problems are provided in the following subsections.

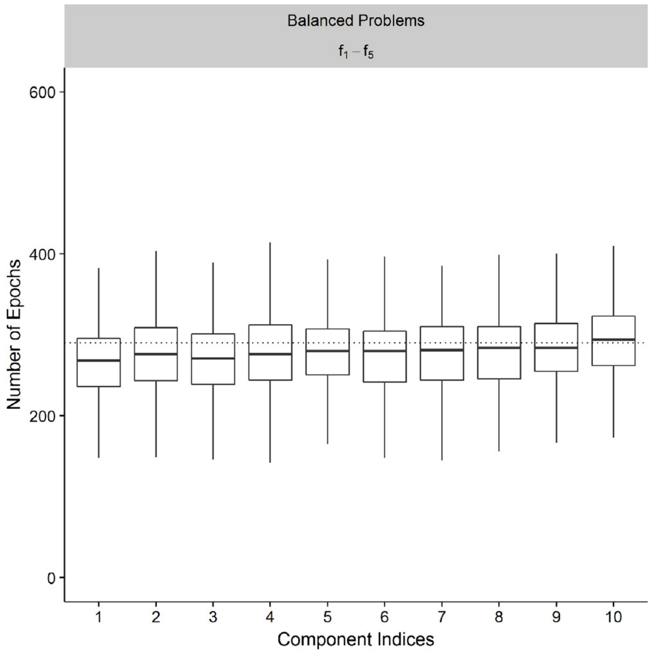

Fig. 2. The performance of BBCC1 on balanced problems.

4.2.1. The uniform contributions

In the first part of the case studies, we investigate the behavior of BBCC1 on balanced problems. For this case, we use f_1–f_5 where each of them has 10 components with equal sizes, coefficients and base functions. Fig. 2 presents the performance of BBCC1 on these problems.

The x-axis of Fig. 2 represents the components indices and the y-axis shows the number of epochs each component is selected. To produce a concise graphic, the statistics of similar components are aggregated. Therefore, the left-most box summarizes the average number of epochs the first components of f_1–f_5 are selected. The dashed line in the figure indicates the performance of a round-robin CC.

As Fig. 2 shows, all components of balanced problems are selected equally, regardless of their base functions. This confirms that in the case of uniform contributions, BBCC1 allocates the resources uniformly (the same as round-robin CCs). Therefore, applying BBCC framework on balanced problems should not degrade the performance of CC algorithms.

4.2.2. The unequal coefficients

In the second set of the case studies, we investigate the effect of unequal component coefficients using the first three categories of the problems. In contrast to the first group, the second and third categories consist of problems having components with nonuniform coefficients. The only difference between these two categories is the level of imbalance (i.e., the variance in the coefficients' magnitude), in a sense that the imbalance in f_{11}–f_{15} are more severe than f_6–f_{10} (see Table 1).

Fig. 3 presents the performance of BBCC1 on f_6–f_{15}. The figure consists of two facets, each of which belongs to one of the categories. In contrast to the balanced functions in Fig. 2, we observe a marked increase in selection rates as the indices increase in Fig. 3. Recall that the magnitude of a coefficient is an exponential function of the component index (i.e., 2^k for f_6–f_{10} and 10^k for f_{11}–f_{15}). Therefore, as we move towards the right side of a facet, the contributions increase.

The strong correlation between the coefficients and average number of selection approves that BBCC1 correctly identified the

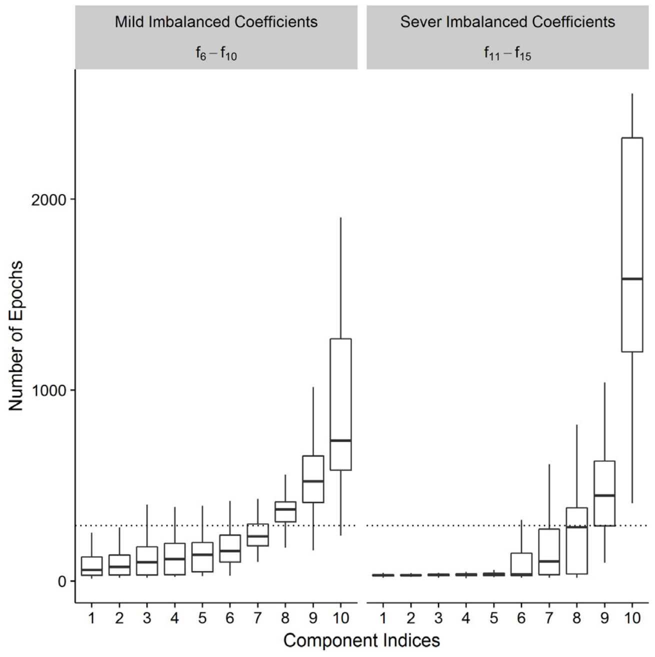

Fig. 3. The effect of imbalance in component coefficients on the number of epochs a component is selected by BBCC for optimization.

most contributing component among the others. A comparison between the BBCC1 resource allocation and traditional round-robin component selector (i.e., the dashed line) reveals that the adaptive component selectors can advance the CCs outcome by allocating a larger portion of the resources to the most contributing components.

Another important observation from comparing categories with various levels of coefficient imbalance is the differences in the selection variances (height of a box plot is an indication of the variance in the corresponding population). Fig. 2 shows that the variance in the selection among the first group of problems (f_1–f_5) is negligible and remains constant for all components. For imbalanced problems in Fig. 3, however, the variance rapidly increases as the coefficients grow. Particularly for the severely imbalanced problems (f_{11}–f_{15}), the variances for the first five components (with the smallest coefficients) are almost zero. Nevertheless, this number for the last component (with the largest coefficient) is extremely large. To investigate whether the differences between search landscapes caused the large variances in Fig. 3, we compare two groups of problems with different base functions.

Fig. 4 compares the behavior of BBCC1 on Rastrigin and Schwefel functions in three scenarios. As the plot shows, regardless of the base function, the selection ratio is constant for balanced problems (the solid lines for f_2 and f_4), and increasing for imbalanced problems (the dashed lines for f_7, f_9, f_{12} and f_{14}). However, the slope of the growth largely depends on the search landscape in the case of imbalanced problems. For example, the 10th component of f_{12} is selected almost 1000 times more often than the last component of f_{14} which has the same coefficient and size, but different base function. This observation suggests that base functions may affect the contribution of components. We will explore this topic further in Section 4.2.4.

4.2.3. The unequal dimensionality

In this part of the case studies, we investigate the impact of unequal component sizes using the fourth and fifth categories of problems (f_{16}–f_{25}). As presented in Table 1, all components of a problem in these categories have the same coefficients and base functions, while their sizes may vary. To create two problem sets with different imbalance levels, the ranges of subproblem sizes are deliberately chosen to be different in the fourth and fifth categories. Particularly, for f_{16}–f_{20} the component sizes are limited to 50, 150 and 200, while for f_{20}–f_{25} they vary from 25 to 250 (see Table 1).

Note that the nonuniform coefficients (studied in Section 4.2.2) only affects the importance (fitness range) of components, while varying component sizes also influence the complexity of the subproblems. This means, for f_{16}–f_{25} components with larger indices not only have a greater impact on the overall fitness values, but also tend to demand more objective function calls to be solved. Therefore, if a particular optimizer fails to make significant progress in the optimization of the large-size subproblems, the component selector may switch to components with more manageable sizes.

Figs. 5 and 6 illustrate the average number of times that components with the same size are selected. As a general trend, components with higher dimensionality are selected more frequently regardless of the level of imbalance. This pattern is expected from contribution-aware CCs as components with a larger number of decision variables tend to have a stronger impact on the overall fitness than the other components.

The only exceptions in Figs. 5 and 6 are Ackley's functions (f_{18} and f_{23}) which the smallest subproblems are selected the most. This behavior does not challenge the importance of dimensionality, but instead, it suggests that the basic DE/rand/1/bin is not powerful enough to improve very large Ackley's subproblems with the given budget. In these cases, the fitness improvements gained from optimizing smaller components is relatively larger than searching high-dimensional search spaces. As a result, BBCC1 selects the simpler subproblems more often than the higher dimensional components which our basic optimizer is incapable to improve significantly. This example highlights the effects of optimizer's power on the components' relative contributions. Neglecting this factor may result in investing too much on optimization of components that are too complex for the adopted optimizer to obtain any significant improvement.

4.2.4. The unequal landscapes

To find out whether BBCC can differentiate between components with different landscapes, we apply it to the sixth category of problems, i.e., f_{25}–f_{30}, each of which has a unique combination of five different base functions (see Table 2). To make the combinations different, for each problem, we leave out one of the six possible base functions. For example, f_{26} and f_{30} do not include Rosenbrock and Elliptic components, respectively. As shown in Table 1, all components of f_{26}–f_{30} have the same coefficient and size (i.e., $\forall k$, $c_k = 1$ and $|\mathcal{D}_k| = 100$).

Each pie chart in Fig. 7 corresponds to one of the problems and shows the distribution of selected components. Note that, each function consists of ten components with five different landscapes. Therefore, each pie has only five parts. We chose to use pie charts instead of line chart because, as opposed to the other problems, there is no meaningful relationship between the index and contribution of a component.

Fig. 7 reveals that BBCC1 chooses components according to their base functions. For example, when Elliptic components exist in a problem (i.e., f_{26}–f_{29}), they are selected more often than any other components such that at least 60% of the budget is allocated to them. After Elliptic, Rosenbrock and Schwefel subproblems are respectively the second and the third highly selected components, when they are included in a problem. On the other hand, Rastrigin, Ackley and Sphere base functions are the least selected components among the others such that only 2% of the budget is allocated to each of them.

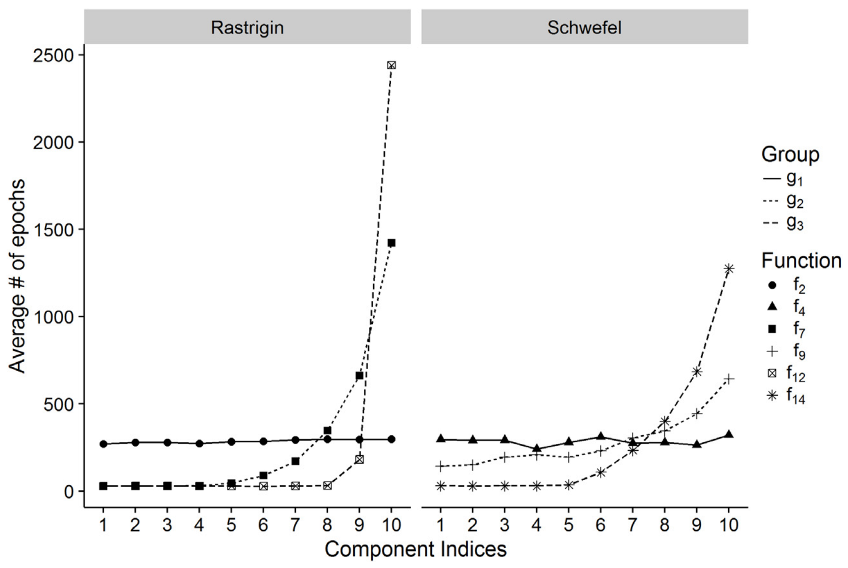

Fig. 4. The effect of search landscape on the behavior of component selector when coefficients are imbalanced.

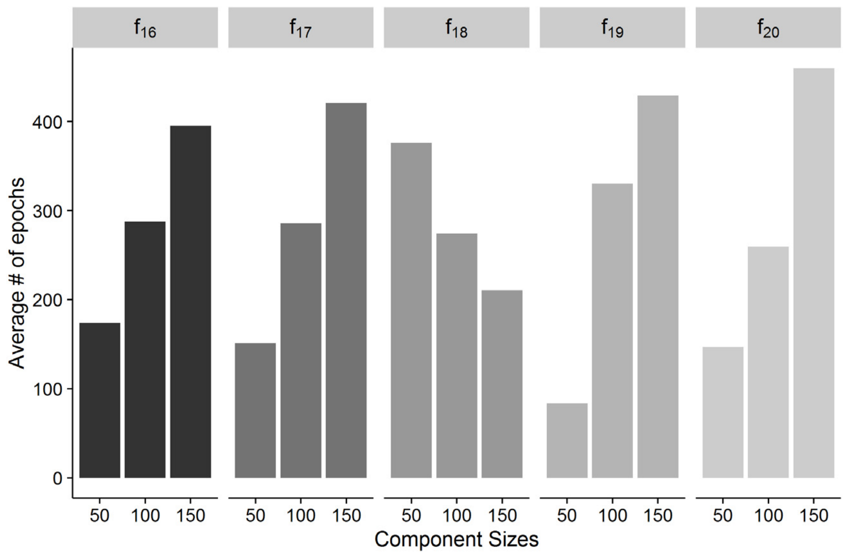

Fig. 5. The effect of moderate imbalance in component sizes on the number of epochs a component is selected by BBCC for optimization.

Another interesting observation from Fig. 7 is that both the simplest (i.e., Sphere) and most difficult (i.e., Ackley's) base functions, for this particular optimizer, are among the least selected components. The rationale for this behavior is that since the Sphere subproblems are easy to solve, they need very few epochs to reach a point that their contributions become relatively insignificant. In another extreme case, the 100-dimensional Ackley subproblems are very difficult for our DE/rand/1/bin. In other words, given the same number of function calls, it is less likely to receive larger improvement over Ackley's subproblems than the others. In such a case, BBCC1 selects components that bring the largest improvement for each single epoch. Note that adopting a different subproblem optimizer may change the order of selection and the portion of computational budget designated to each component. However, BBCC should still be able to adapt to the power of the adopted optimizer and effectively allocate the resources. Due to the limited space, we postpone further empirical studies on the adaptiveness of BBCC to an adopted optimizer to future work.

4.3. Sensitivity analysis

In the previous part, we studied the sensitivity of BBCC1 to different types of imbalance, base functions, and imbalance levels. In this part, we analyze its sensitivity to the values of its parameters. Since BBCC can potentially have numerous instances each having its own set of parameters, a comprehensive parameter analysis demands a separate study. Furthermore, a number of parameters are unique to some specific implementations. For example, ϵ and τ values are only meaningful in ϵ-greedy and Softmax component selectors, respectively. As a result, we choose the simplest implementation of BBCC (i.e., BBCC1) to only study generic parameters that exist in all variants of this framework. Nonetheless, other implementations of BBCC may present different degrees of sensitivity to these generic parameters.

Here, we study the population size $N \in \{10, 20, 50, 100\}$ and epoch length $\delta_t \in \{10, 20, 50, 100\}$ which is the number of iterations each component is optimized when selected. Both of these parameters directly affect the budget allocated to a selected component. More precisely, a selected component consumes $N \times \delta_t$

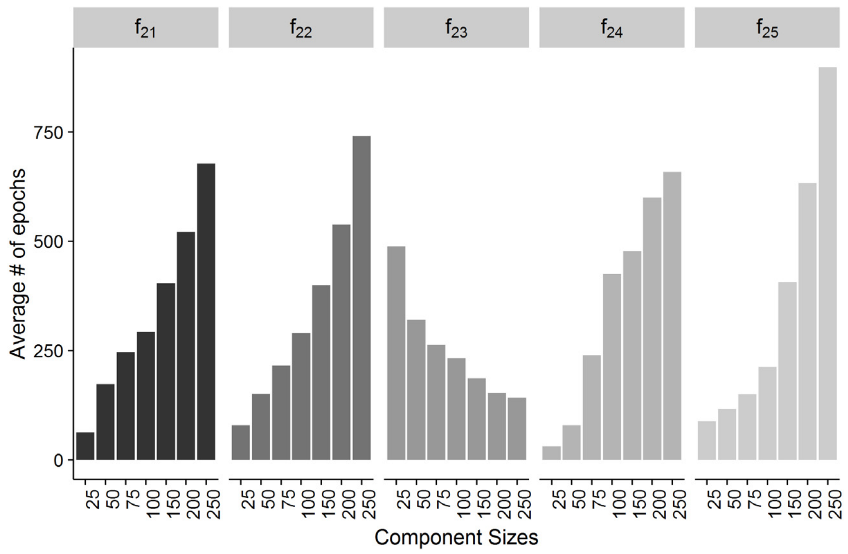

Fig. 6. The effect of severe imbalance in component sizes on the number of epochs a component is selected by BBCC for optimization.

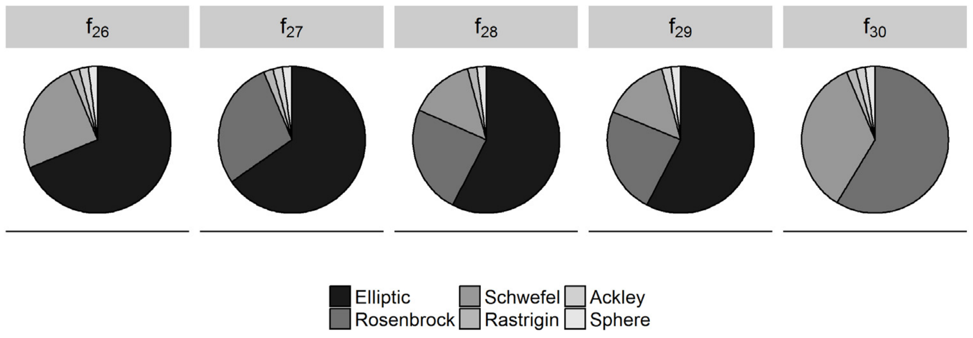

Fig. 7. Average number of component selection.

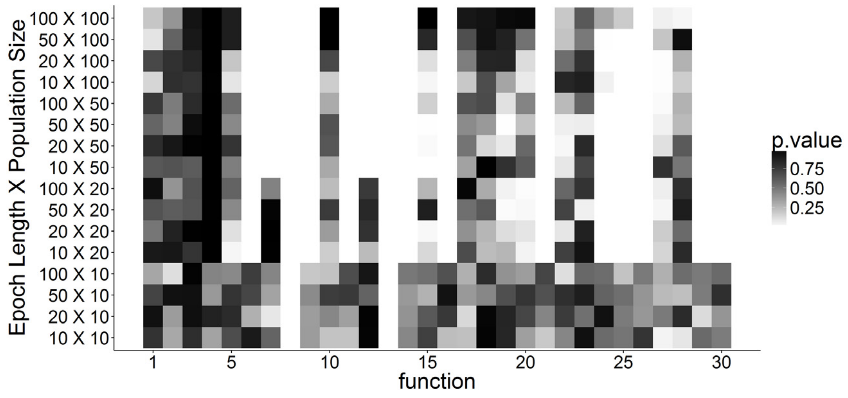

Fig. 8. Comparing BBCC and CC performance in 16 different parameter settings. The brighter a tile is, the more likely that BBCC outperforms the CC with the same parameter values.

function calls in each epoch. In this study, we compare BBCC1 and its round-robin counterpart, CC1, using the same parameter setting on all the problems that we introduced in the case studies. The quality of final solutions obtained by these algorithms is compared using a one-sided Mann–Whitney U test with 95% confidence level where the H_1 is that the distribution of BBCC results is shifted to the left of the distribution of the CC outcomes. The resulting p-values are depicted in Fig. 8. In this heatmap, a bright cell (i.e., small p-value) indicates a strong superiority of the BBCC1 over CC1. The black cells, in contrast, determine both algorithms perform statistically similar.

The x-axis of Fig. 8 indicates the function indices and the y-axis shows the $\delta_t \times N$. Therefore, the row at the bottom of the plot belongs to the epochs with the least allocated budget (only 100 function calls per epoch) and the row at the top associates with the epochs with the largest portion of the budget (10,000 function call per epoch). Since the total function calls are fixed, the number of epochs decreases as δ_t or N increase.

An observation from the figure is that the four rows at the bottom have the most number of dark cells. This suggests that $N = 10$ is not a good choice for this particular implementation of BBCC framework. Although having small population provides more exploration opportunities as the number of epochs increases, it seems DE/rand/1/bin cannot make any significant progress with this small population. Perhaps some other optimizers or improvement measures can remedy or alleviate this problem.

The cells in the five left-most columns are also dark colored which means BBCC1 and CC1 perform statistically comparable on f_1–f_5. This is not surprising as we already showed in the case studies that BBCC1 allocated budget uniformly when the problems are balanced (see Section 4.2.1). This shows that it is safe to use BBCC even if we are not sure whether a given problem is balanced or not.

For many of the problems with mild dimensional imbalance (especially f_{17}–f_{19}), BBCC1 demonstrates negligible improvement over CC1. The most effective parameter settings for this category are $\delta_2 \in \{50, 100\}$ and $N = 20$.

The behavior of BBCC1 on Rosenbrock's and Ackley's problems is particularly interesting. In Rosenbrock's case, the more severe the imbalance the more likely BBCC1 outperforms CC1 (compare f_{15} with f_{10}, and f_{25} with f_{20}). It is also evident that BBCC1 handles the dimensional imbalance better than the coefficient imbalance in the Rosenbrock's case (compare f_{20} and f_{25} with f_{10} and f_{15}).

In Ackley's cases, BBCC1 easily outperforms CC1 on coefficient imbalance (i.e., f_8 and f_{13}), however, it is less likely to improve the baseline in dimensionally imbalanced problems (i.e., f_{18} and f_{23}). Once again, it confirms that DE/rand/1/bin is incapable of solving large-scale Ackley's functions.

For the last category of problems that includes mixed landscape components, BBCC1 improves CC1 performance in most of the cases as long as $N > 10$. Based on Fig. 8, we can conclude that the most difficult case for BBCC1 is f_{28} followed by f_{27}. These problems consist of all base functions except Ackley's and Schwefel's functions, respectively. From Fig. 7, we know that BBCC1 tends to spend most of the resources on Elliptic and Rastrigin's subproblems when dealing with f_{27} and f_{28}

Fig. 8 also suggests that the sensitivity to N may differ from one search landscape to another. For example, the effective range of N is considerably wider for Ackley's instances (e.g., f_8 and f_{13}) than for Rastrigin's problems (e.g., f_7 and f_{12}). The vertical blocks of four tiles (where N is fixed but δ_t varies) that are visible in several parts of Fig. 8 is another indication of the sensitivity of BBCC1 (or DE) to population size.

There is almost no significant evidence in Fig. 8 that suggests δ_t has a great influence on the relative performance of BBCC1. Indeed, in most of the cases, variation in δ_t has little or no effect on the p-values. In the other cases, it seems that having large δ_t when N is also large (e.g., $N = 100$ and $\delta_t \geq 50$ for f_{10}, f_{15} and f_{17}–f_{20}) has an adverse effect on BBCC1. Here, the limited number of epochs can be the reason. Since the maximum number of objective function calls is fixed, having large values for these parameters results in a very few number of epochs.

Overall, this sensitivity analysis suggests that the behavior of BBCC1 may be affected by the landscape structure. Regardless of the base functions, the most effective range for N is $\{20, \ldots, 100\}$, while one should avoid choosing large values for both parameters at the same time. According to Fig. 8, setting both values to 50 results in a significant improvement (BBCC1 over CC1) in the majority of instances.

4.4. BBCC vs. round-robin CCs

In this part, we compare BBCC1 with two variants of round-robin CCs: CC1 and CC2. The main difference between these two CCs is the adopted EA since we use DE/rand/1/bin and SaNSDE as the subproblem optimizers in CC1 and CC2, respectively. For the comparisons, we use all partially separable imbalanced problems from CEC'13 LSGO benchmark set which are f_4-f_{11}. In addition, the effective decomposition for f_1–f_3 and f_{12}–f_{15} is unknown and existing decomposition techniques typically return a single component in which case using a CC framework is irrelevant.

Table 3
BBCC1 vs. round-robin CCs on CEC'13 LSGO imbalanced benchmarks. The μ and σ symbols represent mean and STD values, respectively. The best average fitness values are shown in bold.

Function		BBCC1	CC1	CC2	nWins
f_4	μ	3.27e+08	1.52e+09	**1.97e+08**	**0**
	σ	1.41e+08	5.56e+08	1.51e+08	
f_5	μ	**1.45e+06**	4.20e+06	2.66e+06	**2**
	σ	3.23e+05	2.54e+06	7.12e+05	
f_6	μ	**1.04e+06**	1.04e+06	1.06e+06	**0**
	σ	1.47e+05	1.47e+05	1.49e+03	
f_7	μ	2.14e+05	**1.93e+05**	5.12e+07	**1**
	σ	1.56e+05	5.04e+04	3.67e+07	
f_8	μ	**6.96e+12**	3.02e+14	7.19e+13	**2**
	σ	4.21e+12	1.82e+14	6.07e+13	
f_9	μ	**1.21e+08**	5.69e+08	2.85e+08	**2**
	σ	3.42e+07	8.59e+07	6.20e+07	
f_{10}	μ	**9.21e+07**	9.28e+07	9.43e+07	**0**
	σ	1.30e+07	1.31e+07	3.64e+05	
f_{11}	μ	**5.99e+07**	5.68e+08	2.62e+10	**2**
	σ	6.64e+07	1.53e+08	3.10e+10	
W-T-L:		–	**5-3-0**	**5-2-1**	
Rank:		**1.3125**	**2.3125**	**2.3750**	
Order:		**1**	**2**	**3**	

As Table 3 shows, BBCC1 finds the best solutions in six out of eight cases. In the Ackley's instances (i.e., f_6 and f_{10}), however, the improvements are not statistically significant. According to the Win–Tie–Loss (W–T–L) numbers, BBCC1 significantly outperforms each of the round-robin CCs in five problems. The Friedman ranks at the bottom of the table confirm that BBCC1 is the best performer among the compared CC variants.

4.5. BBCC vs. contribution-aware CCs

In this part we conduct a series of comparison studies between BBCC1 and other contribution-aware CCs such as CBCC [7,12], MOFBVE [8], and CCFR [13] variants.

Table 4 compares BBCC1 with CBCC1, CBCC2 and three variants of CBCC3. The Friedman ranks and orders show that BBCC1 achieves the best overall results in comparison with CBCCs, and closely followed by CBCC3 when $p_t \in \{0, 0.05\}$. The W–T–L numbers reveal that it significantly improves CBCC1 and CBCC2 in five out of eight problems while each of them outperforms BBCC1 in only one task. BBCC1 significantly improves CBCC3 variants in half of the cases, whilst performs statistically similar in two problems and losses in the other two.

The average nWins score of BBCC1 in Table 4 is +35%. These scores show that BBCC1 significantly outperforms all variants of CBCC in f_5, f_7, f_9 and f_{11}. It also improves all CBCC variants on Ackley's instances (i.e., f_6 and f_{10}), although the improvements are not statistically significant. It is evident in Table 4 that BBCC1 performance on Elliptic (i.e., f_4 and f_8) is lower than CBCC3 variants. Indeed, the fitness values highlighted in bold reveal that BBCC1 finds the best solutions for all problems except the Elliptic instances. Since CBCCs use an adaptive optimizer (i.e., SaNSDE) rather than simple DE/rand/1/bin, one may conclude that adopting different optimizers is a contributing factor in the superiority of CBCCs in Elliptic cases. This hypothesis is supported by the fact that CC2 with SaNSDE also outperforms CC1 with DE/rand/1/bin in Elliptic problems (see Table 3). We intend to investigate this hypothesis by adopting a variety of optimizers on a wider range of problems in our future work.

Table 4
BBCC1 vs. CBCCs on CEC'13 LSGO imbalanced benchmarks. The μ and σ symbols represent mean and STD values, respectively. The best average fitness values are shown in bold.

Function		BBCC1	CBCC1	CBCC2	CBCC3			*nWins*
					$p_t = 0$	$p_t = 0.05$	$p_t = 1$	
f_4	μ	3.27e+08	7.71e+07	8.77e+10	**2.20e+07**	2.97e+07	4.08e+07	**−3**
	σ	1.41e+08	4.05e+07	1.14e+10	8.05e+06	1.56e+07	2.09e+07	
f_5	μ	**1.45e+06**	2.28e+06	2.09e+06	2.13e+06	1.99e+06	2.34e+06	**5**
	σ	3.23e+05	3.55e+05	3.52e+05	3.49e+05	3.61e+05	4.70e+05	
f_6	μ	**1.04e+06**	1.06e+06	1.06e+06	1.05e+06	1.05e+06	1.06e+06	**0**
	σ	1.47e+05	2.18e+03	1.65e+03	1.07e+04	1.97e+03	2.15e+03	
f_7	μ	**2.14e+05**	6.38e+07	8.82e+07	2.09e+07	1.42e+07	4.75e+07	**5**
	σ	1.56e+05	4.01e+07	6.78e+07	3.04e+07	2.18e+07	3.38e+07	
f_8	μ	6.96e+12	1.38e+13	1.88e+12	1.21e+10	**8.23e+09**	1.51e+11	**−3**
	σ	4.21e+12	1.14e+13	2.80e+11	2.40e+10	1.03e+10	2.87e+11	
f_9	μ	**1.21e+08**	2.32e+08	2.03e+08	1.40e+08	1.56e+08	2.02e+08	**5**
	σ	3.42e+07	4.85e+07	2.45e+07	1.55e+07	3.51e+07	5.09e+07	
f_{10}	μ	**9.21e+07**	9.41e+07	9.41e+07	9.22e+07	9.29e+07	9.40e+07	**0**
	σ	1.30e+07	3.91e+05	2.59e+05	1.10e+06	5.81e+05	4.82e+05	
f_{11}	μ	**5.99e+07**	1.58e+10	1.63e+10	4.74e+08	6.24e+08	1.33e+09	**5**
	σ	6.64e+07	2.26e+10	2.76e+10	2.95e+08	3.47e+08	1.41e+09	
W-T-L:		–	**5-2-1**	**5-2-1**	**4-2-2**	**4-2-2**	**4-2-2**	
Rank:		**2.0625**	**5.1875**	**5.0625**	**2.2500**	**2.3125**	**4.1250**	
Order:		**1**	**6**	**5**	**2**	**3**	**4**	

Table 5
BBCC1 vs. MOFBVEs on CEC'13 LSGO imbalanced benchmarks. The μ and σ symbols represent mean and STD values, respectively. The best average fitness values are shown in bold.

Function		BBCC1	MOFBVE				*nWins*
			bi-level	*3-level*	*4-level*	*5-level*	
f_4	μ	**3.27e+08**	9.09e+09	9.92e+09	9.13e+09	7.29e+09	**4**
	σ	1.41e+08	2.60e+09	5.01e+09	3.08e+09	3.63e+09	
f_5	μ	**1.45e+06**	2.69e+06	2.77e+06	2.80e+06	3.01e+06	**4**
	σ	3.23e+05	5.30e+05	4.19e+05	3.61e+05	5.43e+05	
f_6	μ	1.04e+06	**8.56e+04**	9.33e+04	9.25e+04	8.81e+04	**−4**
	σ	1.47e+05	2.41e+04	2.91e+04	2.15e+04	2.59e+04	
f_7	μ	**2.14e+05**	5.85e+06	5.82e+06	9.35e+06	6.53e+06	**4**
	σ	1.56e+05	2.18e+06	2.21e+06	1.26e+07	2.80e+06	
f_8	μ	**6.96e+12**	2.31e+13	1.81e+13	2.54e+13	2.88e+13	**4**
	σ	4.21e+12	8.86e+12	9.05e+12	1.03e+13	1.15e+13	
f_9	μ	**1.21e+08**	2.81e+08	2.65e+08	2.62e+08	1.94e+08	**4**
	σ	3.42e+07	3.09e+07	3.16e+07	2.62e+07	3.30e+07	
f_{10}	μ	9.21e+07	3.34e+04	2.09e+04	2.55e+03	**1.89e+03**	**−4**
	σ	1.30e+07	2.17e+04	2.22e+04	3.31e+02	1.15e+03	
f_{11}	μ	**5.99e+07**	7.64e+08	4.55e+08	4.48e+08	3.83e+09	**3**
	σ	6.64e+07	1.04e+09	3.66e+08	3.97e+08	1.34e+10	
W–T–L		–	**6-0-2**	**6-0-2**	**6-0-2**	**5-1-2**	
Rank:		**2.000**	**3.125**	**3.250**	**3.375**	**3.250**	
Order:		**1**	**2**	**3**	**4**	**3**	

Table 6
BBCC1 vs. CCFRs on CEC'13 LSGO imbalanced benchmarks. The μ and σ symbols represent mean and STD values, respectively. The best average fitness values are shown in bold.

Function		BBCC1	CCFR			*nWins*
			IDG2	DG	CMA-ES	
f_4	μ	3.2e+08	9.6e+07	9.1e+10	**9.5e+07**	**−1**
	σ	1.4e+08	4.0e+07	5.6e+10	4.0e+07	
f_5	μ	**1.4e+06**	4.2e+07	3.0e+06	2.8e+06	**3**
	σ	3.2e+05	3.2e+05	5.2e+05	3.1e+05	
f_6	μ	**1.0e+06**	4.1e+07	1.1e+06	1.0e+06	**1**
	σ	1.4e+05	1.0e+03	1.6e+03	1.0e+03	
f_7	μ	**2.1e+05**	8.2e+08	1.4e+08	2.0e+07	**3**
	σ	1.5e+05	2.9e+07	9.7e+07	2.9e+07	
f_8	μ	6.eE+12	4.6e+11	1.6e+15	**6.6e+10**	**−1**
	σ	4.2e+12	9.5e+10	1.0e+15	9.5e+10	
f_9	μ	**1.2e+08**	8.1e+09	1.9e+08	1.8e+08	**3**
	σ	3.4e+07	2.8e+07	2.8e+07	2.8e+07	
f_{10}	μ	**9.2e+07**	7.9e+08	9.5e+07	9.4e+07	**1**
	σ	1.3e+07	1.8e+05	3.1e+05	1.8e+05	
f_{11}	μ	**5.9e+07**	1.4e+09	2.8e+10	4.1e+08	**3**
	σ	6.6e+07	3.4e+08	6.0e+10	3.4e+08	
W-T-L:		–	**6-0-2**	**6-2-0**	**4-2-2**	
Rank:		**1.500**	**3.375**	**3.375**	**1.750**	
Order:		**1**	**3**	**3**	**2**	

Table 5 compares the results of BBCC1 with bi-level to 5-level MOFBVEs. Putting the special case of Ackley's functions (i.e., f_6 and f_{10}) aside, BBCC1 significantly outperforms all variants of MOFBVE on all other imbalanced problems (except f_{11} for 5-level MOFBVE which there is a tie). Although the BBCC1's average nWins score is +46.87%, it shows no improvement over MOFBVE variants in the Ackley's cases. This is not a surprising observation as from Table 3 we recall that the BBCC1 was incapable of significantly improving the round-robin CCs in these problems.

As the W–T–L summary in Table 5 shows, BBCC1 performs better than MOFBVEs in the majority of the tasks. Indeed, it significantly improves each MOFBVE instance in at least six out of eight cases. In addition, the bold fitness values in the table reveal that among all five algorithms, BBCC1 found the best solutions for five out of eight problems. Finally, the Friedman ranking identifies BBCC1 as the best algorithm.

Table 6 compares BBCC1 with three variants of CCFR. The main differences between these algorithms are the adopted grouping and optimizer. As the nWins scores (with the average of +50%) reveal, BBCC1 outperform all variants of CCFR on f_5, f_7, f_9, and f_{11}. However, its relative performance on Elliptic instances (i.e., f_4 and f_8) is not promising. In these two cases, CCFR with CMA-ES optimizer performs the best.

The W–T–L numbers in Table 6 show that BBCC1 significantly improves CCFR with SaNSDE optimizer in six cases, and CCFR-CMA-ES in half of the cases. The Friedman statistics ranks BBCC1 as the best performer closely followed by CCFR-CMA-ES.

For further statistical analysis, we perform Friedman and Quade significant tests. Table 7 provides the obtained p-values. As the results show, both test show there are significant differences between the performance of CBCCs and CCFRs when compared with

Table 7
Overall significance tests for BBCC1 vs. contribution-aware CCs. All p-values smaller than 0.05 are shown in bold.

Test	CBCCs	MOFBVEs	CCFRs
Friedman	**1.3e−04**	3.92e−01	**1.94e−03**
Quade	**1.8e−05**	1.48e−01	**7.13e−03**

Table 8
Pairwise significance tests on BBCC1 and CBCCs. All p-values smaller than 0.05 are shown in bold.

Test	*Adjustment*	CBCC1	CBCC2	CBCC3		
				$p_t = 0$	$p_t = 0.05$	$p_t = 1$
Friedman	None	**1.6e−06**	**3.2e−06**	7.3e−01	6.4e−01	**5.5e−04**
	Holm	**8.0e−06**	**1.2e−05**	7.3e−01	1	**1.6e−03**
	Bonferroni	**8.0e−06**	**1.6e−05**	1	1	**2.7e−03**
Quade	None	**4.0e−03**	**4.4e−03**	3.8e−01	5.2e−01	2.2e−01
	Holm	**2.0e−02**	**1.7e−02**	7.7e−01	5.2e−01	6.6e−01
	Bonferroni	**2.0e−02**	**2.2e−02**	1	1	1

Table 9
Pairwise significance tests on BBCC1 and CCFR. All p-values smaller than 0.05 are shown in bold.

Test	*Adjustment*	CCFR		
		IDG2	DG	CMA-ES
Friedman	None	**1.5e−05**	**1.5e−05**	4.6e−01
	Holm	**3.7e−05**	**3.75-05**	4.6e−01
	Bonferroni	**4.5e−05**	**4.5e−05**	1
Quade	None	5.9e−02	**6.9e−03**	6.7e−01
	Holm	1.1e−01	**2.0e−02**	6.7e−01
	Bonferroni	1.7e−01	**2.0e−02**	1

BBCC1. Therefore, we perform pairwise posthoc test to find the pairs that are responsible for these differences. The results are summarized in Tables 8 and 9. Note that we do not perform posthoc tests on MOFBVE results since nor Friedman neither Quade test show a huge difference.

The Friedman posthoc test (a.k.a. Conover's pairwise tests) in Table 8 reveals that BBCC1 performs significantly better than CBCC1, CBCC2, and CBCC3 when $p_t = 1$. The pairwise posthoc-Quade test only confirms a significant difference between BBCC1 and the early variants of CBCC (i.e., CBCC1 and CBCC2). In other cases, BBCC1's performance statistically comparable with CBCC3 variants.

According to Friedman posthoc test in Table 9, BBCC1 performs significantly better than all CCFR variants except when CMA-ES is adopted as the optimizer. The posthoc-Quade tests, however, only confirms a significant difference between BBCC1 and CCFR-DG.

Overall, BBCC1, as the simplest implementation of BBCC framework, is ranked as the best performer when compared with the other available contribution-aware CCs. As mentioned above, the improvements are statistically sound, in many cases. We could probably observe more interesting patterns and detect more significant differences if the number of imbalanced benchmark functions was large enough.

4.6. BBCC vs. state-of-the-art

In the experiments, we compare the BBCC1 results with the two state-of-the-art algorithms: MA-SW-Chains and MOS'13. These two particular algorithms are chosen because MA-SW-Chains is the winner of CEC'10 LSGO competition and MOS won the CEC'13 and CEC'15 LSGO competitions. The main goal of this brief comparison is to show that the simplest implementation of BBCC framework coupled with a very basic optimizer can compete with the most advanced large-scale optimizers. Since BBCC1 could outperform CC1 and other contribution-aware CCs, we expect that by adopting these advanced EAs as the subproblem optimizer of BBCC instances, we can enhance their performance even further.

Table 10
BBCC vs. state-of-the-art algorithms on CEC'13 LSGO benchmark. μ and σ represent mean and STD values, respectively. The bold numbers indicate the best performer for a particular problem.

Function		BBCC1	MA-SW-Chains	MOS-CEC'13
f_4	μ	3.27e+08	4.58e+09	**1.74e+08**
	σ	1.41e+08	2.46e+09	7.87e+07
f_5	μ	**1.45e+06**	1.87e+06	6.94e+06
	σ	3.23e+05	3.06e+05	8.85e+05
f_6	μ	1.04e+06	1.01e+06	**1.48e+05**
	σ	1.47e+05	1.53e+04	6.43e+04
f_7	μ	2.14e+05	3.45e+06	**1.62e+04**
	σ	1.56e+05	1.27e+06	9.10e+03
f_8	μ	**6.96e+12**	4.85e+13	8.00e+12
	σ	4.21e+12	1.02e+13	3.07e+12
f_9	μ	1.21e+08	**1.07e+08**	3.83e+08
	σ	3.42e+07	1.68e+07	6.29e+07
f_{10}	μ	9.22e+07	9.18e+07	**9.02e+05**
	σ	1.30e+07	1.06e+06	5.07e+05
f_{11}	μ	5.99e+07	2.19e+08	**5.22e+07**
	σ	6.64e+07	2.98e+07	2.05e+07
Rank:		**2.000**	**2.375**	**1.625**
Order:		**2**	**3**	**1**

Table 10 presents the results of BBCC1, MA-SW-Chains [18] and MOS [41] on the imbalanced problems from CEC'13 LSGO benchmarks. As the results show, BBCC1 significantly outperforms MA-SW-Chains in five problems and MOS in only two cases. The MOS can outperform BBCC1 in four problems, whilst MA-SW-Chains cannot significantly beat BBCC1 in any cases.

The average nWins score of BBCC1 is positive (+18.75%) and it can achieve the second position based on Friedman ranking. However, none of the statistical tests confirm any significant differences between BBCC1 and the winners of the previous LSGO competitions. The Friedman and Quade p-values for the results in Table 10 are $3.24e{-}01$ and $4.08e{-}01$, respectively. These statistics confirm that the most basic implementation of BBCC can compete with the most advanced EAs on solving large-scale imbalanced optimization problems.

5. Conclusions

In this paper, we have proposed a general framework called bandit-based cooperative coevolution (BBCC) to address the imbalance subproblem contributions in large-scale optimization problems. In contrast with the traditional CC framework which uniformly allocates computational resources to all components, BBCC adopts well-studied bandit algorithms to learn the contribution of each component to the long-term improvement in objective value and allocate resources accordingly.

Through extensive experiments on 30 imbalance large-scale benchmarks, we have shown that BBCC has the ability to handle a variety of scenarios. The sensitivity studies revealed that our simple BBCC implementation is robust with respect to the changes in generic parameter values. Our comparison studies also confirmed that the efficiency of BBCC in exploration–exploitation maintenance helps it to outperform previous contribution-aware techniques. We have also demonstrated that when some degrees of imbalance exists in the problems, even simple instances of BBCC perform statistically similar or better than the state-of-the-art algorithms that ignore such property of the problems.

With respect to the flexibility and generality of the proposed framework, we expect several future studies to analyze BBCC's performance in different scenarios, to expand the framework, and to apply it to real-world applications image matting and adapting recurrent neural networks. We will further examine the sensitivity of BBCC to the accuracy of decomposition and the number of components, study the behavior of BBCC when a dynamic grouping algorithm is adopted, and compare different instances of BBCC in order to find the most effective combination of the improvement measure, contribution estimator, and component selector algorithms.

Acknowledgments

This work was supported by National Key R&D Program of China (Grant No. 2017YFC0804003), EPSRC (Grant Nos. EP/J017515/1 and EP/P005578/1), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (Grant No. 2017ZT07X386), Shenzhen Peacock Plan (Grant No. KQTD2016112514355531), the Science and Technology Innovation Committee Foundation of Shenzhen (Grant No. ZDSYS201703031748284) and the Program for University Key Laboratory of Guangdong Province (Grant No. 2017KSYS008).

References

[1] A.F. Villaverde, J.A. Egea, J.R. Banga, A cooperative strategy for parameter estimation in large scale systems biology models, BMC Syst. Biol. 6 (1) (2012) 75.
[2] H. Yi, Q. Duan, T.W. Liao, Three improved hybrid metaheuristic algorithms for engineering design optimization., Appl. Soft Comput. 13 (5) (2013) 2433–2444.
[3] N. Xiong, D. Molina, M.L. Ortiz, F. Herrera, A Walk into Metaheuristics for Engineering Optimization: Principles, Methods and Recent Trends, Int. J. Comput. Intell. Syst. 8 (4) (2015) 606–636.
[4] S. Mahdavi, M.E. Shiri, S. Rahnamayan, Metaheuristics in large-scale global continues optimization: A survey, Inform. Sci. (2014).
[5] M.N. Omidvar, Y. Mei, X. Li, Effective decomposition of large-scale separable continuous functions for cooperative co-evolutionary algorithms, in: IEEE Congress on Evolutionary Computation (CEC'14), IEEE, 2014, pp. 1305–1312.
[6] M.N. Omidvar, M. Yang, Y. Mei, X. Li, X. Yao, DG2: A faster and more accurate differential grouping for large-scale black-box optimization, IEEE Trans. Evol. Comput. (2017) http://dx.doi.org/10.1109/TEVC.2017.2694221.
[7] M.N. Omidvar, X. Li, X. Yao, Smart use of computational resources based on contribution for cooperative co-evolutionary algorithms, in: Proc. of Genetic and Evolutionary Computation Conference, ACM, ACM, 2011, pp. 1115–1122.
[8] S. Mahdavi, S. Rahnamayan, M.E. Shiri, Multilevel framework for large-scale global optimization, Soft Comput. (2016) 1–30.
[9] B. Kazimipour, M.N. Omidvar, X. Li, A. Qin, A sensitivity analysis of contribution-based cooperative co-evolutionary algorithms, in: IEEE Congress on Evolutionary Computation (CEC'15), IEEE, 2015, pp. 417–424.
[10] G.A. Trunfio, Adaptation in cooperative coevolutionary optimization, in: Adaptation and Hybridization in Computational Intelligence, Vol. 18, Springer, 2015, pp. 91–109.
[11] M.N. Omidvar, X. Li, K. Tang, Designing benchmark problems for large-scale continuous optimization, Inform. Sci. 316 (2015) 419–436, http://dx.doi.org/10.1016/j.ins.2014.12.062.
[12] M.N. Omidvar, B. Kazimipour, X. Li, X. Yao, CBCC3 - A contribution-based cooperative co-evolutionary algorithm with improved exploration/exploitation balance, in: IEEE Congress on Evolutionary Computation (CEC'16), IEEE, 2016.
[13] M. Yang, M.N. Omidvar, C. Li, X. Li, Z. Cai, B. Kazimipour, X. Yao, Efficient resource allocation in cooperative co-evolution for large-scale global optimization, IEEE Trans. Evol. Comput. PP (1) (2017) 1–1.
[14] S. Mahdavi, S. Rahnamayan, M.E. Shiri, Cooperative co-evolution with sensitivity analysis-based budget assignment strategy for large-scale global optimization, Appl. Intell. (2017) 1–26, http://dx.doi.org/10.1007/s10489-017-0926-z.
[15] V. Kuleshov, D. Precup, Algorithms for multi-armed bandit problems, arXiv preprint arXiv:1402.6028.
[16] B. Kazimipour, X. Li, A.K. Qin, A review of population initialization techniques for evolutionary algorithms, in: IEEE Congress on Evolutionary Computation (CEC'14), IEEE, 2014, pp. 2585–2592.
[17] A. LaTorre, S. Muelas, J.M. Peña, Multiple offspring sampling in large scale global optimization, in: IEEE Congress on Evolutionary Computation (CEC'12), IEEE, 2012, pp. 1–8.
[18] D. Molina, M. Lozano, F. Herrera, MA-SW-Chains: Memetic algorithm based on local search chains for large scale continuous global optimization, in: IEEE Congress on Evolutionary Computation (CEC'10), IEEE, 2010, pp. 3153–3160.
[19] J. Brest, M.S. Maučec, Self-adaptive differential evolution algorithm using population size reduction and three strategies, Soft Comput. 15 (11) (2011) 2157–2174.
[20] B. Kazimipour, M.N. Omidvar, X. Li, A.K. Qin, A novel hybridization of opposition-based learning and cooperative co-evolutionary for large-scale optimization, in: IEEE Congress on Evolutionary Computation (CEC'14), IEEE, 2014, pp. 2833–2840.
[21] M.A. Potter, K.A. De Jong, A cooperative coevolutionary approach to function optimization, in: Proc. of International Conference on Parallel Problem Solving from Nature, Vol. 2, 1994, pp. 249–257.
[22] M.A. Potter, K.A. De Jong, Cooperative coevolution: An architecture for evolving coadapted subcomponents, Evol. Comput. 8 (1) (2000) 1–29.
[23] M.N. Omidvar, X. Li, Y. Mei, X. Yao, Cooperative co-evolution with differential grouping for large scale optimization, IEEE Trans. Evol. Comput. 18 (3) (2014) 378–393, http://dx.doi.org/10.1109/TEVC.2013.2281543.
[24] E.K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, J.R. Woodward, A classification of hyper-heuristic approaches, in: Handbook of Metaheuristics, Springer, 2010, pp. 449–468.
[25] E.K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: A survey of the state of the art, J. Oper. Res. Soc. 64 (12) (2013) 1695–1724.
[26] J. Gittins, K. Glazebrook, R. Weber, Multi-Armed Bandit Allocation Indices, John Wiley & Sons, 2011.
[27] K. Li, A. Fialho, S. Kwong, Q. Zhang, Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition, IEEE Trans. Evol. Comput. 18 (1) (2014) 114–130.
[28] Á. Fialho, L. Da Costa, M. Schoenauer, M. Sebag, Analyzing bandit-based adaptive operator selection mechanisms, Ann. Math. Artif. Intell. 60 (1) (2010) 25–64, http://dx.doi.org/10.1007/s10472-010-9213-y.
[29] Á. Fialho, M. Schoenauer, M. Sebag, Analysis of adaptive operator selection techniques on the royal road and long k-path problems, in: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, ACM, 2009, pp. 779–786.
[30] P.A. Consoli, L.L. Minku, X. Yao, Dynamic selection of evolutionary algorithm operators based on online learning and fitness landscape metrics, in: Simulated Evolution and Learning, Springer, 2014, pp. 359–370.
[31] A. Fialho, R. Ros, M. Schoenauer, M. Sebag, Comparison-based adaptive strategy selection with bandits in differential evolution, in: Parallel Problem Solving from Nature, PPSN XI, Springer, 2010, pp. 194–203.
[32] W. Gong, Á. Fialho, Z. Cai, H. Li, Adaptive strategy selection in differential evolution for numerical optimization: an empirical study, Inform. Sci. 181 (24) (2011) 5364–5386.
[33] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, Cambridge Univ Press, 1998.
[34] D. Rainville, M. Sebag, C. Gagné, M. Schoenauer, D. Laurendeau, et al., Sustainable cooperative coevolution with a multi-armed bandit, in: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, ACM, 2013, pp. 1517–1524.
[35] J. Vermorel, M. Mohri, Multi-armed bandit algorithms and empirical evaluation, in: Machine learning: ECML 2005, Springer, 2005, pp. 437–448.
[36] M. Kim, R.I.B. McKay, D.K. Kim, X.H. Nguyen, Evolutionary operator self-adaptation with diverse operators, in: Genetic Programming, Springer, 2012, pp. 230–241.
[37] Á. Fialho, M. Schoenauer, M. Sebag, Toward comparison-based adaptive operator selection, in: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, ACM, 2010, pp. 767–774.
[38] B. Kazimipour, X. Li, A.K. Qin, Effects of population initialization in differential evolution for large scale optimization, in: IEEE Congress on Evolutionary Computation (CEC'14), IEEE, 2014, pp. 2404–2411.
[39] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power, Inform. Sci. 180 (10) (2010) 2044–2064.
[40] X. Li, K. Tang, M.N. Omidvar, Z. Yang, K. Qin, H. China, Benchmark Functions for the cec2013 special session and competition on large-scale global optimization, gene 7 (2013) 33, http://goanna.cs.rmit.edu.au/~xiaodong/cec13-lsgo.
[41] A. LaTorre, S. Muelas, J.M. Pena, Large scale global optimization: Experimental results with MOS-based hybrid algorithms, in: IEEE Congress on Evolutionary Computation (CEC'13), IEEE, 2013, pp. 2742–2749.