**Proceedings Paper:**
Omidvar, MN orcid.org/0000-0003-1944-4624, Kazimipour, B, Li, X et al. (1 more author) (2016) CBCC3 — A contribution-based cooperative co-evolutionary algorithm with improved exploration/exploitation balance. In: 2016 IEEE Congress on Evolutionary Computation (CEC). 2016 IEEE Congress on Evolutionary Computation (CEC), 24-29 Jul 2016, Vancouver BC, Canada. IEEE , pp. 3541-3548. ISBN 978-1-5090-0623-6

https://doi.org/10.1109/cec.2016.7744238

# CBCC3 – A Contribution-Based Cooperative Co-evolutionary Algorithm with Improved Exploration/Exploitation Balance

Mohammad Nabi Omidvar*, Borhan Kazimipour†, Xiaodong Li†, Xin Yao*
* School of Computer Science, University of Birmingham,
Birmingham B15 2TT, U.K., E-mail: {m.omidvar, x.yao}@cs.bham.ac.uk)
†School of Computer Science and Information Technology, RMIT University,
Melbourne, VIC 3000, Australia, Email:{borhan.kazimipour, xiaodong.li}@rmit.edu.au

***Abstract*—Cooperative Co-evolution (CC) is a promising framework for solving large-scale optimization problems. However, the round-robin strategy of CC is not an efficient way of allocating the available computational resources to components of *imbalanced* functions. The imbalance problem happens when the components of a partially separable function have non-uniform contributions to the overall objective value. Contribution-Based Cooperative Co-evolution (CBCC) is a variant of CC that allocates the available computational resources to the individual components based on their contributions. CBCC variants (CBCC1 and CBCC2) have shown better performance than the standard CC in a variety of cases. In this paper, we show that over-exploration and over-exploitation are two major sources of performance loss in the existing CBCC variants. On that basis, we propose a new contribution-based algorithm that maintains a better balance between exploration and exploitation. The empirical results show that the new algorithm is superior to its predecessors as well as the standard CC.**

## I. Introduction

Since their inception, Evolutionary Algorithms (EAs) [1] have been applied to a multitude of optimization tasks exhibiting very different characteristics, ranging from discrete single-objective problems to continuous non-linear multi-objective problems. The derivative-free nature of EAs make them suitable for a wide range of complex black-box optimization problems. Like any other optimization method, the performance of EAs are affected by the dimensionality of given problems [2].

Divide-and-conquer is a common problem solving technique in which a complex problem is tackled by being broken into a set of smaller and simpler subproblems. Cooperative Co-evolution (CC) [3] is an explicit means of problem decomposition in evolutionary algorithms. Liu et al. [4] showed empirical evidence that the CC framework can be an effective way of solving large-scale continuous optimization problems.

One of the considerations of using CC framework lies in finding an efficient and effective decomposition for a given problem. In many applications, the domain knowledge can help practitioners to identify problem components. One of the popular examples is the conversion of a multi-objective optimization problem to a single objective problem where the new objective function is essentially a weighted sum of all given objective functions. Even if such domain knowledge is not available, the advances in the EA allow us to automatically break down a large-scale problem into smaller components by the cost of a few function calls. Recent studies show that new algorithms can decompose most of the existing large-scale benchmarks with 100% accuracy [5–7].

Another important consideration that greatly affects the overall optimization performance is the allocation of available computational resources to various components. In a classic CC, all components receive an equal share of the available computational resources through the round-robin optimization of all components in each cycle. However, it has been shown that the round-robin strategy can waste a considerable amount of the available resources on the so called *imbalanced* functions [8]. In this context, imbalance refers to the unequal contribution of different components to the overall objective value [8–10]. Omidvar et al. [8] have shown that in the presence of strong imbalance in the contribution of components to the overall objective value, only a few components have dominant effect on the overall improvement of the objective value, making the contributions of the other components negligible. In such situations, most of the effort in optimizing the components with a lower contribution is wasted.

To alleviate the imbalance issue, Contribution-Based Cooperative Co-evolution (CBCC) [8] has been proposed in which the components with a higher contribution are given a higher share of the available resources. While the components' contribution maybe unknown to the optimizer, CBCC adopt simple rules to approximate these values and dynamically select the most contributing component for the next round of optimization. Note that although accurate grouping improves CBCC performance, having an ideal decomposition is not a prerequisite. In [11] the authors showed that even in the presence of significant decomposition noise, CBCC achieves better results than the classic round-robin CC with the same computational budget.

CBCC has two major variants: CBCC1 and CBCC2. CBCC1 is more exploratory and gives the components an opportunity to update their contributions more frequently, whereas CBCC2 is more exploitative and commits to the component with the highest contribution until it becomes stagnant. CBCC showed significant improvement over the classic CC on the CEC'2010 LSGO benchmark suite [12]. However, the degree of imbalance in the CEC'2010 benchmarks is limited since the underlying components of most functions have equal contributions. The CEC'2013 LSGO benchmark suite has been

proposed with the aim of improving upon the CEC'2010 benchmarks to better resemble the major properties of real-world problems. One such change is the inclusion of a more challenging imbalance scheme in the CEC'2013 benchmarks. The preliminary experimental results on the CEC'2013 benchmarks showed that the new imbalance scheme poses a serious challenge to CBCC [10].

In this paper, we further investigate the cause of CBCC's poor performance on the CEC'2013 benchmarks. We show that the resource allocation policy of CBCC1 and CBCC2 are over-exploratory and over-exploitative, respectively. These two schemes are biased towards the components that have an initial good contribution and therefore do not respond to the local changes in the contribution of each component in a timely manner. Based on these findings, we propose an improved version of Contribution-Based Cooperative Co-evolution, CBCC3, that significantly outperforms the classic CC as well as CBCC1 and CBCC2.

The rest of this paper is organized as follows: Section II reviews the imbalance functions and CBCC algorithms. Section III contains analysis of CBCC1 and CBCC2 on the CEC'2013 LSGO benchmarks. Section IV consists of the details of the new algorithm – CBCC3. Section V presents the empirical results and an analysis of CBCC3's performance with respect to its predecessors. Finally, Section VI concludes the paper.

## II. Background

### A. The Imbalance Problem

Many real-world problems have a modular nature. This is often implemented by taking a linear combination of a several sub-functions (components) [13]. Such weighting mechanism places a different level of emphasis on each component. This means that some components may have considerably larger impact on the overall objective value than the other components. In optimization problems, the issue of unequal contributions is called *the imbalance problem* [8–10]. A function that exhibits the imbalance problem is called an *imbalanced function* [10]. For optimizing an imbalanced function, under limited resources, it is reasonable to use more resources for solving the most contributing components. It is therefore obvious that equal allocation of computational resources among all components is an inefficient strategy and will waste the limited resources.

The imbalance issue of real-world problems has been emphasized in the recent benchmark functions for large-scale optimization [9, 10]. A very simple yet flexible method to introduce various levels of imbalance into partially separable functions is through explicit weighting of components:

$$f(\mathbf{x}) = \sum_{i=1}^{m} w_i . f_i(\mathbf{x}_i), \tag{1}$$

where $m$ is the number of components, $f_i(\cdot)$ is the $i$th sub-function, $\mathbf{x}_i$ is the decision vector of the $i$th sub-function, and $w_i$ is the weight associated to the $i$th sub-function which is calculated as follows:

$$w_i = 10^{s\mathcal{N}(0,1)},$$

**Algorithm 1:** $(x^\star, f^\star) =$ `CBCC`$(f, \underline{\mathbf{x}}, \overline{\mathbf{x}}, \mu, n, \Gamma_{\max}, \gamma, v)$

```
(g, x_1, ..., x_g, Γ) = grouping(f, x_lo, x_hi, n) ;
P = rand(μ, n) ;
for i = 1 → g do
    cv_i = rand(1, |x_i|);
cv = (cv_1, ..., cv_g) ;
f_c = f(cv) ;
Δ = zeros(1, g) ;
Γ = zeros(1, g) ;
while Σ_{i=1}^{g} Γ_i < Γ_max − Γ − 1 do
    for i = 1 → g do
        f_p = f_c ;
        (cv, Γ_i, f_c) = optimizer(P, cv, x_i, γ) ;
        δ = f_p − f_c ;
        Δ_i = Δ_i + δ ;
    j = max_index(Δ);
    δ = Δ_j ;
    while δ ≠ 0 do
        f_p = f_c ;
        (cv, Γ_j, f_c) = optimizer(P, cv, x_j, γ) ;
        δ = f_p − f_c ;
        Δ_j = Δ_j + δ ;
        if v = 1 then
            break ;
x* = cv ; f* = f(x*) ;
return (x*, f*) ;
```

where $\mathcal{N}(0,1)$ is a Gaussian distribution with zero mean and unit variance. The parameter $s$ which is specified by the user controls the level of imbalance. Setting $s$ to zero removes the explicit weighting, and makes the non-uniformity of component sizes the only sources of imbalance. In the CEC'2013 LSGO benchmark suite, $s$ is set to 3.

Beside explicit weighting that was described, there are other sources of implicit imbalance. For example, non-uniform subproblem sizes is another source of imbalance [10]. Having different landscapes (i.e., basis functions) can also cause imbalance in real-world problems.

At the time of this writing, there are few algorithms that address the imbalance issue in the context of large-scale optimization. These attempts are limited to the original Contribution-Based Cooperative Co-evolution (CBCC) [5, 8] and their recent variant called Cooperative Co-evolution with Adaptive Optimizer Iterations (CCAOI) [14]. All variants of CBCC (CBCC1, CBCC2 and CCAOI) showed statistically significant improvements over the standard CC. In Section III we study the performance of CBCC1 and CBCC2 to show that without a proper balance between exploration and exploitation in resource allocation mechanism, the benefit of even an *ideal* decomposition cannot be fully realized.

### B. Contribution-Based Cooperative Co-evolution

This subsection contains the details of Contribution-Based Cooperative Co-evolution [8]. Algorithm 1 shows how CBCC uses the contribution information to select various components for optimization. Table I contains a brief description of the variables used in Algorithm 1. After the initialization, each of the components that were formed using an arbitrary decomposition technique (i.e., `grouping` function) are optimized in a round-robin fashion. This step, which was previously called *testing* phase, is necessary to measure the initial contributions

Table I: Important variables used in Algorithm 1.

| Variable | Description |
|---|---|
| $\mathbf{x}^\star$ | the best solution vector found by the algorithm. |
| $f^\star$ | the objective value for of $\mathbf{x}^\star$. |
| $f$ | the function handle of the objective function. |
| $\underline{\mathbf{x}}$ | vector of lower bound constrains of the decision variables. |
| $\overline{\mathbf{x}}$ | vector of the upper bound constrains of the decision variables. |
| $\mu$ | the population size. |
| $n$ | the dimensionality of the objective function. |
| $\Gamma_{\max}$ | the maximum number of available objective function evaluations. |
| $\gamma$ | the number of times that the component optimizer optimizes each component. |
| $v$ | the version number of CBCC which is either 1 or 2. |
| $\mathbf{cv}$ | the context vector that is used by the optimizer to construct a complete solution for evaluation. |

of components (Algorithm 1, lines 10-14). In this paper, we sometimes refer to the testing phase as the exploration phase. Similar to traditional CC, any arbitrary optimizer can be adopted in CBCC (i.e., `optimizer` function) to optimize the components. It can be seen that the vector $\mathbf{\Delta}$ keeps track of the changes in the fitness of components.

In the next phase which we call the exploitation phase (Algorithm 1, lines 17-21), the component with the largest entry in $\mathbf{\Delta}$ is selected for further optimization. The difference between the two versions of the CBCC (i.e. CBCC1 and CBCC2) is in this phase. In CBCC1, the selected component is optimized for only one iteration whereas in CBCC2, the selected component is optimized for as long as it improves the fitness. When no improvement is observed, the algorithm enters the exploration phase to give all of the components another equal chance to update their contributions in $\mathbf{\Delta}$. This shows that CBCC1 is more conservative than CBCC2 and performs more exploration to find the component with the highest contribution. On the other hand, CBCC2 is greedier and commits to a component as long as its immediate contribution is non-zero. It should be noted that the contribution information is accumulated from the first cycle.

## III. Case Studies

In this section, we conduct a short empirical study on the performance of CBCC1 and CBCC2 while DECC (a CC algorithm that uses SaNSDE [15] as the component optimizer.) is chosen as the baseline. To keep it simple, we use ideal decomposition for all experiments. Throughout this paper we limit ourselves to the partially separable functions of the CEC'2013 LSGO benchmark suite ($f_4$-$f_{11}$). The other functions which are not included in this study are either fully separable ($f_1$-$f_3$), overlapping ($f_{13}$-$f_{14}$), or fully non-separable ($f_{15}$), for which there is no unique decomposition. Moreover, these functions cannot be classified as imbalanced functions.

Table II contains the median, mean, and the standard deviation of 25 independent runs for DECC, CBCC1 and CBCC2. We used Wilcoxon rank-sum [16] test to compare both variants of CBCC against the baseline. The highlighted entries show that CBCC performs significantly better than DECC using a 95% confidence interval ($p$-value $= 0.05$). If the performance of either version of CBCC is not statically different from DECC, the entries are marked with the symbol '$\approx$'. If DECC significantly outperforms either of the CBCC variants, the symbol '$\downarrow$' is shown next to the entry having the worse performance.

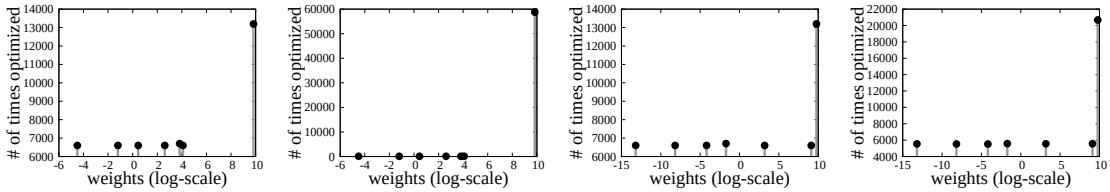
Table II: Comparison between DECC and CBCC1 and CBCC2 on selected function from the CEC'2013 benchmark suite. Highlighted entries are significantly better than the baseline (DECC) based on Wilcoxon rank-sum test ($\alpha = 0.05$).

| Function | Stats | DECC | CBCC1 | CBCC2 |
|---|---|---|---|---|
| $f_4$ | Median | 1.53e+08 | **6.54e+07** | 9.03e+10$^\downarrow$ |
| | Mean | 1.97e+08 | 7.71e+07 | 8.77e+10 |
| | StDev | 1.51e+08 | 4.05e+07 | 1.14e+10 |
| $f_5$ | Median | 2.65e+06 | 2.29e+06$^\approx$ | **2.06e+06** |
| | Mean | 2.66e+06 | 2.28e+06 | 2.09e+06 |
| | StDev | 7.12e+05 | 3.55e+05 | 3.52e+05 |
| $f_6$ | Median | 8.74e+04 | 8.74e+04$^\approx$ | 8.35e+04$^\approx$ |
| | Mean | 8.57e+04 | 8.85e+04 | 8.39e+04 |
| | StDev | 1.95e+04 | 2.88e+04 | 2.36e+04 |
| $f_7$ | Median | 4.53e+07 | 6.23e+07$^\approx$ | 7.85e+07$^\approx$ |
| | Mean | 5.12e+07 | 6.38e+07 | 8.82e+07 |
| | StDev | 3.67e+07 | 4.01e+07 | 6.78e+07 |
| $f_8$ | Median | 5.43e+13 | **1.09e+13** | **1.90e+12** |
| | Mean | 7.19e+13 | 1.38e+13 | 1.88e+12 |
| | StDev | 6.07e+13 | 1.14e+13 | 2.80e+11 |
| $f_9$ | Median | 2.95e+08 | **2.34e+08** | **2.00e+08** |
| | Mean | 2.85e+08 | 2.32e+08 | 2.03e+08 |
| | StDev | 6.20e+07 | 4.85e+07 | 2.45e+07 |
| $f_{10}$ | Median | 7.05e+01 | 7.51e+01$^\approx$ | 7.17e+01$^\approx$ |
| | Mean | 6.90e+01 | 7.44e+01 | 7.16e+01 |
| | StDev | 1.68e+01 | 9.97e+00 | 1.36e+01 |
| $f_{11}$ | Median | 1.51e+10 | 1.41e+09$^\approx$ | 1.44e+09$^\approx$ |
| | Mean | 2.62e+10 | 1.58e+10 | 1.63e+10 |
| | StDev | 3.10e+10 | 2.26e+10 | 2.76e+10 |

We can see from Table II that both CBCC1 and CBCC2 perform statistically similar to DECC on most functions ($62.5\%$), and perform significantly better on only three functions ($37.5\%$). Comparing this with the success rate of CBCC on the CEC'2010 LSGO benchmarks based on the results reported in [5], we can clearly see that CBCC's performance drops significantly on the CEC'2013 LSGO benchmarks (the success rate drops from $60\%$ to $37.5\%$).

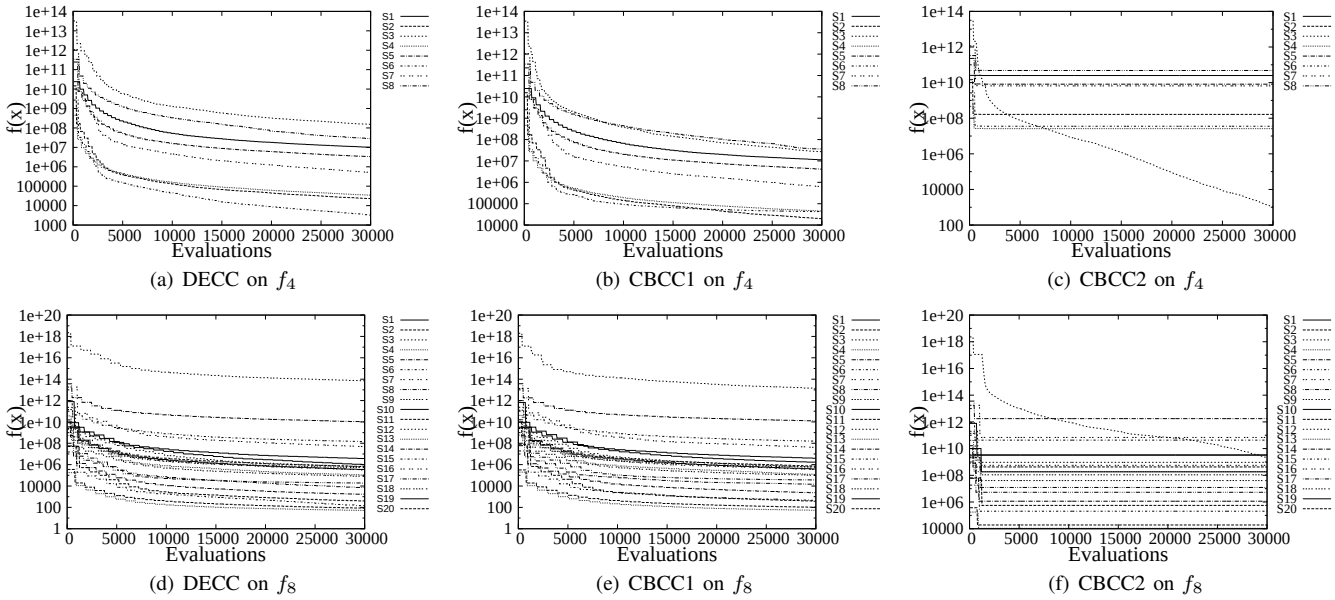
To get a better insight into the behavior of CBCC1 and CBCC2, we have included four plots in Figure 1 that show how the budget has been allocated to various components with respect to their weight. In this set of figures, each bar represents a non-separable component. The $x$-axis shows the weight associated to each component (in logarithmic scale). The height of each bar represents the total number of evaluations allocated to each component. It should be noted that weights are not the only measures of contribution. Other factors such as component sizes, convergence behavior and stagnation can also have considerable effects. However, considering their magnitudes, weights have a strong relationship to components' contributions.

CBCC is designed to optimize the components with higher contribution more frequently. Therefore, we expect to see longer bars towards the right side of each figure (i.e., larger weights). Figure 1 shows that components with the highest weight is evaluated more often than the other components. It is also clear that in CBCC2 the component with the highest contribution is emphasized more than the same component when CBCC1 is adopted. The equal length of most of the other bars suggests that the remaining budget is equally divided among the rest of components.




(a) CBCC1 on $f_4$ (b) CBCC2 on $f_4$ (c) CBCC1 on $f_5$ (d) CBCC2 on $f_5$

Figure 1: Number of evaluations used to optimize components for CBCC1 and CBCC2 on $f_4$ and $f_5$ (CEC'2013 LSGO)




(a) DECC on $f_4$ (b) CBCC1 on $f_4$ (c) CBCC2 on $f_4$

(d) DECC on $f_8$ (e) CBCC1 on $f_8$ (f) CBCC2 on $f_8$

Figure 2: Convergence plots of individual components for DECC, CBCC1 and CBCC2 on $f_4$ and $f_8$ (CEC'2013 LSGO).

Two lessons that we can learn from Figure 1 are: *a*) The component with the highest weight is predominantly selected for optimization. This is a good strategy as long as the selected component maintains its high contribution during optimization. Later we will see that this is not necessarily the case, and in several cases CBCC2 fails to switch to another component that just became the most contributing component. *b*) Due to dominance of one component, equal allocation of resources through the exploration phase wastes a considerable amount of objective function evaluations. This can be the main reason that CBCC1 cannot outperform DECC in $f-5$ (see Table II), although it spent more resources on the optimization of the component with the largest weight (see Figure 1).

With respect to CBCC2, we see that the component with the largest weight is given the largest share of the available resources. However, it is unexpected to see that on $f_4$ its overall performance is significantly worse than both CBCC1 and DECC. The reason for this type of behavior can be seen by inspecting the convergence plot of individual components, which are shown in Figure 2 for two selected functions.

As can be seen in Figure 2, the component with the largest weight has an initial large objective value. Therefore, CBCC optimizes this component more frequently than the remaining components. This causes a rapid drop in the objective value of this particular component. This behavior can be clearly seen in Figures 2(c) and 2(f). When the objective value of the selected component drops below those of other components, CBCC should stop optimizing that component in order to give a chance to other components.

It is clear from Figures 2(c) and 2(f) that CBCC2 fails to detect this event, which is the reason behind its poor performance. This happens because in Algorithm 1, the termination condition of the inner while loop on line 17 is total stagnation of the selected component ($\delta = 0$). In other words, as long as the selected component improves, it will be optimized even if its immediate contribution is less than all other components. Note that even if we force CBCC2 to break out of the inner loop, it is highly likely that the same component be selected after the exploration phase, due to accumulation of its contributions from the very beginning rounds of optimization (Algorithm 1 lines 14 and 21). It should be noted CBCC1 is also prone to this error, but it takes more time for the error to become evident on the convergence plots.

In general, two major drawbacks of CBCC1 and CBCC2 can be summarized as follows:

- CBCC's slow response to local changes in the fitness value and its strong reliance on the information accumulated form the early stages of evolution. This is more evident in CBCC2.
- Over exploration by frequent application of the exploration phase in Algorithm 1. This is an inefficient use

**Algorithm 2:** $(x^\star, f^\star) = \texttt{CBCC3}(f, \underline{\mathbf{x}}, \overline{\mathbf{x}}, \mu, n, \Gamma_{\max}, \gamma, p_t)$

```
1  (g, x_1, ..., x_g, Γ) = grouping(f, x_lo, x_hi, n) ;
2  P = rand(μ, n) ;
3  for i = 1 → g do
4      cv_i = rand(1, |x_i|);
5  cv = (cv_1, ..., cv_g) ;
6  f_c = f(cv) ;
7  Δ = zeros(1, g) ;
8  Γ = zeros(1, g) ;
9  while Σ_{i=1}^{g} Γ_i < Γ_max − Γ − 1 do
10     if Σ_{i=1}^{g} Γ_i = 0 or rand() < p_t then
11         for i = 1 → g do
12             f_p = f_c ;
13             (cv, Γ_i, f_c) = optimizer(P, cv, x_i, γ) ;
14             δ = f_p − f_c;
15             if δ ≠ 0 then
16                 Δ_i = δ ;
17     (C, I) = sort(Δ, descending);
18     j = I_1 ;
19     while C_1 > C_2 and Σ_{i=1}^{g} Γ_i < Γ_max − Γ − 1 do
20         f_p = f_c ;
21         (cv, Γ_j, f_c) = optimizer(P, cv, x_j, γ) ;
22         δ = f_p − f_c;
23         if δ ≠ 0 then
24             Δ_j = C_1 = δ ;
25 x* = cv ; f* = f(x*) ;
26 return (x*, f*) ;
```

of the limited resources. This behavior is more evident in CBCC1 and is the main reason behind the similar performance of CBCC to DECC.

In the next section, we explain how these issues can be alleviated by a more immediate feedback mechanism as well as a simple probabilistic exploration mechanism, which are built into CBCC3.

## IV. The Proposed Framework

In the previous section, we identified two major shortcomings of CBCC1 and CBCC2. This section contains the details of a new version of contribution-based cooperative co-evolution, CBCC3, that addresses these shortcomings. Algorithm 2 shows the details of CBCC3. We can see that CBCC3, to a large extent, resembles CBCC1 and CBCC2. However, unlike its predecessors, it does not run the exploration phase in every co-evolutionary cycle. As line 10 of Algorithm 2 shows, the exploration phase happens with probability $p_t$ while at least one execution of this phase is guaranteed at the first cycle. Based on our observations in the previous section, the exploration phase should happen with a relatively low probability. The sensitivity analysis of CBCC3 to $p_t$ will be postponed to Section V.

The other major difference of CBCC3 is its reliance on more recent contribution information to select a component for further optimization (eliminating the use of historical information in CBCC1 and CBCC2). On lines 14-16 and 22-24 of Algorithm 2, we can see that the last non-zero difference in the objective value of two consecutive iterations is recorded as the contribution of a given component ($\mathbf{\Delta}_i$). As was mentioned earlier, the parameter $\gamma$ of `optimizer` determines the number of iterations that it optimizes a component. A larger $\gamma$ stabilizes the value of $\delta$ to a non-zero value.

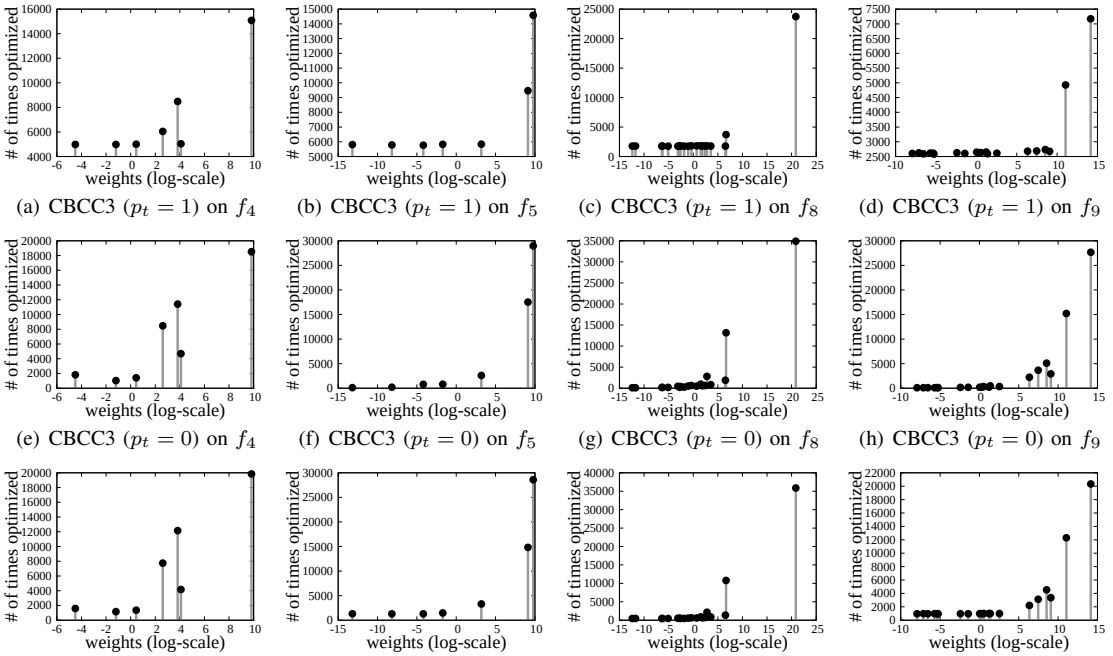
After the exploration phase (Algorithm 2, lines 9-16), the contribution of components are sorted (Algorithm 2, lines 17-18) and the component with the largest contribution is selected for further optimization in the exploitation phase (Algorithm 2, lines 19-24). The vector $\mathbf{I}$ contains the indices of components after being sorted with respect to their contributions. The vector $\mathbf{C}$, like $\mathbf{\Delta}$, contains the contribution information in descending order. Therefore, $\mathbf{C}_1$ is the largest contribution that is associated to the component $\mathbf{I}_1$, and $\mathbf{C}_2$ is the second largest contribution that is associated to the component $\mathbf{I}_2$. In the next phase, which forms the basis of CBCC, the component with the largest contribution is optimized further (Algorithm 2, lines 19-24). We can see that CBCC3 optimizes the selected component ($\mathbf{I}_1$) for as long as its contribution ($\mathbf{C}_1$) is greater than the last recorded contribution of the second most important component ($\mathbf{C}_2$). This ensures that the crossing behavior that was observed in CBCC2 (as shown in Figures 2(c) and 2(f)) does not happen in CBCC3. In other words, when two or more components have similar contributions, the condition of the inner while loop (Algorithm 2, line 19) ensures that all components with non-zero contributions will be given an equal chance to be optimized.

## V. Experiments and Analysis

In this section, we present the experimental results of CBCC3 and compare them with DECC, CBCC1 and CBCC2 using a subset of the CEC'2013 LSGO benchmark suite. The total number of fitness evaluations is set to $3 \times 10^6$ as suggested in [9]. The component optimizer that we used for our experiments is SaNSDE [15]. The parameter $\gamma$ is set to 100, and the population size ($\mu$) is set to 50.

Table III contains the median, mean and standard deviation of 25 independent runs for the aforementioned algorithms using ideal grouping. For the statistical significance test, we first use Kruskal-Wallis [16], which is a non-parametric equivalent of ANOVA, with a 95% confidence interval. If a significant difference is identified among the given algorithms, we run a series of pair-wise Wilcoxon rank-sum tests using Holm $p$-value correction to account for the family-wise error rate [16]. An entry for an algorithm is highlighted if it is not outperformed by any other algorithm on a given function. In order to test the sensitivity of CBCC3 to the parameter $p_t$, CBCC3 was tested with the following $p_t$ values: $\{0, 0.05, 1\}$. Table III clearly shows that CBCC3 is superior to its predecessors (CBCC1 and CBC2) as well as traditional CC (DECC).

For a better understanding of CBCC3's behavior, we have included a series of plots in Figure 3 to show how the computational resources are allocated to various components with respect to their weights (similar to Figure 1). We can see that when $p_t = 1$, which means the exploration phase occurs at every cycle, the allocation pattern is very similar to that of CBCC1. For a clearer comparison between variations of CBCC3 and other algorithms, the number of wins, losses, and ties are reported in Table IV using a pair-wise Wilcoxon rank-sum with 95% confidence interval *without* Holm correction. Holm correction is only needed when performing multiple hypothesis testing. Here, all algorithms are tested with CBCC3 as the baseline.




(a) CBCC3 ($p_t = 1$) on $f_4$ (b) CBCC3 ($p_t = 1$) on $f_5$ (c) CBCC3 ($p_t = 1$) on $f_8$ (d) CBCC3 ($p_t = 1$) on $f_9$

(e) CBCC3 ($p_t = 0$) on $f_4$ (f) CBCC3 ($p_t = 0$) on $f_5$ (g) CBCC3 ($p_t = 0$) on $f_8$ (h) CBCC3 ($p_t = 0$) on $f_9$

(i) CBCC3 ($p_t = 0.05$) on $f_4$ (j) CBCC3 ($p_t = 0.05$) on $f_5$ (k) CBCC3 ($p_t = 0.05$) on $f_8$ (l) CBCC3 ($p_t = 0.05$) on $f_9$

Figure 3: Number of evaluations CBCC3 used to optimize each component of $f_4$, $f_5$, $f_8$, and $f_9$ (CEC'2013 LSGO benchmarks).

Table III: Comparison between variants of CBCC3 and CBCC1, 2 and DECC on $f_4$-$f_{11}$ from the CEC'2013 LSGO benchmarks suite. The highlighted entries are significantly better based on pair-wise Wilcoxon rank-sum with Holm $p$-value correction ($\alpha = 0.05$).

| | | | | | CBCC3 | | |
|---|---|---|---|---|---|---|---|
| | Stats | DECC | CBCC1 | CBCC2 | $p_t = 1$ | $p_t = 0$ | $p_t = 0.05$ |
| $f_4$ | Median | 1.53e+08 | 6.54e+07 | 9.03e+10 | 3.61e+07 | **2.18e+07** | **2.51e+07** |
| | Mean | 1.97e+08 | 7.71e+07 | 8.77e+10 | 4.08e+07 | 2.20e+07 | 2.97e+07 |
| | StDev | 1.51e+08 | 4.05e+07 | 1.14e+10 | 2.09e+07 | 8.05e+06 | 1.56e+07 |
| $f_5$ | Median | 2.65e+06 | 2.29e+06 | **2.06e+06** | 2.23e+06 | **2.16e+06** | **1.97e+06** |
| | Mean | 2.66e+06 | 2.28e+06 | 2.09e+06 | 2.34e+06 | 2.13e+06 | 1.99e+06 |
| | StDev | 7.12e+05 | 3.55e+05 | 3.52e+05 | 4.70e+05 | 3.49e+05 | 3.61e+05 |
| $f_6$ | Median | 8.74e+04 | 8.74e+04 | 8.35e+04 | 8.74e+04 | 8.74e+04 | 8.35e+04 |
| | Mean | 8.57e+04 | 8.85e+04 | 8.39e+04 | 8.65e+04 | 8.45e+04 | 7.94e+04 |
| | StDev | 1.95e+04 | 2.88e+04 | 2.36e+04 | 1.88e+04 | 1.92e+04 | 3.43e+04 |
| $f_7$ | Median | 4.53e+07 | 6.23e+07 | 7.85e+07 | 4.68e+07 | **3.86e+05** | **3.27e+05** |
| | Mean | 5.12e+07 | 6.38e+07 | 8.82e+07 | 4.75e+07 | 2.09e+07 | 1.42e+07 |
| | StDev | 3.67e+07 | 4.01e+07 | 6.78e+07 | 3.38e+07 | 3.04e+07 | 2.18e+07 |
| $f_8$ | Median | 5.43e+13 | 1.09e+13 | 1.90e+12 | 7.29e+10 | **2.28e+09** | **5.47e+09** |
| | Mean | 7.19e+13 | 1.38e+13 | 1.88e+12 | 1.51e+11 | 1.21e+10 | 8.23e+09 |
| | StDev | 6.07e+13 | 1.14e+13 | 2.80e+11 | 2.87e+11 | 2.40e+10 | 1.03e+10 |
| $f_9$ | Median | 2.95e+08 | 2.34e+08 | 2.00e+08 | 2.08e+08 | **1.42e+08** | **1.58e+08** |
| | Mean | 2.85e+08 | 2.32e+08 | 2.03e+08 | 2.02e+08 | 1.40e+08 | 1.56e+08 |
| | StDev | 6.20e+07 | 4.85e+07 | 2.45e+07 | 5.09e+07 | 1.55e+07 | 3.51e+07 |
| $f_{10}$ | Median | 7.05e+01 | 7.51e+01 | 7.17e+01 | 7.68e+01 | 7.46e+01 | 7.30e+01 |
| | Mean | 6.90e+01 | 7.44e+01 | 7.16e+01 | 7.66e+01 | 7.44e+01 | 7.15e+01 |
| | StDev | 1.68e+01 | 9.97e+00 | 1.36e+01 | 1.24e+01 | 1.07e+01 | 1.49e+01 |
| $f_{11}$ | Median | 1.51e+10 | 1.41e+09 | 1.44e+09 | 1.07e+09 | **4.49e+08** | **6.31e+08** |
| | Mean | 2.62e+10 | 1.58e+10 | 1.63e+10 | 1.33e+09 | 4.74e+08 | 6.24e+08 |
| | StDev | 3.10e+10 | 2.26e+10 | 2.76e+10 | 1.41e+09 | 2.95e+08 | 3.47e+08 |

We can see that all versions of CBCC3 outperform DECC, CBCC1 and CBCC2. When $p_t = 1$, CBCC3 has more ties with the other algorithms. This is consistent with our observations from Figure 3. When $p_t = 1$, the difference between CBCC3 and other algorithms comes from the way the contributions are quantified as well as the termination criteria of the inner loop in Algorithm 2 (line 19). Table IV clearly shows that the new approach, in which more immediate feedback is used, is more reliable and improves the overall performance. The performance of CBCC3 with $p_t = 0$ (only one exploration phase at the beginning of a run) is better than CBCC3 with $p_t = 1$. This also shows that over-exploration is detrimental to the performance of the algorithm.

Table IV: Number of wins, loses and ties (w/l/t) between all pairs of algorithms using Wilcoxon rank-sum ($\alpha = 0.05$).

| | | | | CBCC3 | | |
|---|---|---|---|---|---|---|
| | DECC | CBCC1 | CBCC2 | $p_t = 0$ | $p_t = 0.05$ | $p_t = 1$ |
| DECC | – | 4/0/4 | 3/2/3 | 5/0/3 | 6/0/2 | 4/0/4 |
| CBCC1 | 0/4/4 | – | 3/1/4 | 5/0/3 | 6/0/2 | 4/0/4 |
| CBCC2 | 2/3/3 | 1/3/4 | – | 5/0/3 | 5/0/3 | 4/1/3 |
| CBCC3, $p_t = 0$ | 0/5/3 | 0/5/3 | 0/5/3 | – | 0/0/8 | 0/5/3 |
| CBCC3, $p_t = 0.05$ | 0/6/2 | 0/6/2 | 0/5/3 | 0/0/8 | – | 0/6/2 |
| CBCC3, $p_t = 1$ | 0/4/4 | 0/4/4 | 1/4/3 | 5/0/3 | 6/0/2 | – |

Figure 4 shows the convergence plots of individual components of CBCC3 using various $p_t$ values on $f_4$ and $f_6$. We can see that the crossing behavior of CBCC2 (Figure 2(c)) does not happen for CBCC3. We can also see that the convergence plots are more concentrated. This is very clear on $f_6$. To show this behavior quantitatively, we reported the standard deviation among the final objective values of all components in Table V.

It is expected that an efficient contribution-based algorithm minimizes the variation between the objective value of individual components. With respect to the convergence plots, such as those shown in Figures 4, we expect to see that a contribution-based algorithm forces the objective value of all components to converge close to a particular value. Assuming that none of the components becomes stagnant, if the objective value

of one component drops below those of other components, its contribution to the overall objective value becomes negligible relative to other components. For this reason, the algorithm should force the objective value of all components close to each other and maintain this trend thereafter. If this does not happen, it means that CBCC is not successful in selecting the best component for optimization at every cycle.

Table V clearly shows the strong correlation between the performance of CBCC and the variability among the objective values of components. For simplicity, the medians of 25 independent runs from Table III are replicated in Table V. Here, without a reference to any statistical significance test, the entry for the algorithm with the smallest median is highlighted. The last three columns of Table V report the standard deviation among the final objective values of individual components across all the 25 independent runs, and the smallest standard deviation is highlighted. It should be noted that these are different from the standard deviations of the 25 independent runs reported in Table III. With the exception of $f_6$, $f_8$, $f_{10}$, we see a strong correlation between low variability among the objective value of individual components and the overall objective value. According to Table III we know that all algorithms perform similarly on functions $f_6$ and $f_{10}$. Therefore, the only exceptions is function $f_8$. Here the conclusion is that CBCC3 with a low $p_t$ value results in lower variability among among its components, which has a direct relationship with the concentration of convergence curves in Figure 4 and its overall good performance as reported in Table III.

Although CBCC3 shows significant improvement over its predecessors and the standard CC, its similar performance on $f_6$ and $f_{10}$ requires further investigation. The convergence plots of CBCC3 on $f_6$ shows that the objective value of several components have an initial improvement and stays unchanged throughout the optimization process. Since the magnitude of the objective value of these components is relatively larger than other components, we expect that CBCC3 allocates a considerable portion of the available computational resources to these components. We know from Algorithm 2 that the exploration phase happens with probability $p_t$ only if the algorithm breaks out of the exploitation phase. The convergence plots do not show whether this happens because of natural stagnation of these components or because of lack of exploration due to dominance of particular components. A drawback of CBCC3 is that it only relies on the magnitude of components' contributions for its selection policy and does not approximate the likelihood of their realization. This can be treated as an online learning problem. One way of taking both the likelihood and the magnitude of contribution is to treat the problem as an online learning problem. In general, the goal of contribution-based cooperative co-evolutionary algorithms is to maximize a long term profit which, in this case, is the final solution quality of an optimization problem through systematic selection and optimization of its components based on an immediate feedback mechanism (contributions). Such studies require a separate in-depth study which is beyond the scope of this work.

## VI. Conclusion

In this paper, we have proposed the CBCC3 algorithm which is an improved version of CBCC1 and CBCC2 that performs significantly better than the traditional cooperative co-evolution on imbalanced functions. We have shown that CBCC1 suffers from over-exploration by excessive application of a exploration phase in which the contribution of all components is re-estimated. Contrary to CBCC1, CBCC2 suffers from over-exploitation by greedily optimizing the component with the highest contribution until it becomes stagnant. In both CBCC1 and CBCC2, the contributions of all components are accumulated from the first cycle and is used to identify the component with the highest contribution to the overall fitness. We have shown that the accumulation of contributions from the first cycle biases the selection mechanism of CBCC1 and CBCC2 towards the components that have an initial good contribution. To alleviate these issues, in CBCC3 a component is optimized as long as its immediate contribution drops below the last recorded contribution of other components. Additionally, CBCC3 completely eliminates the accumulation of contributions and selects a component based on its more recent contribution. Moreover, the exploration phase occurs with some probability $p_t$. The experimental results have shown that CBCC3 performs significantly better than the traditional DECC as well as CBCC1 and CBCC2 with a wide range of $p_t$ values including the extreme values such as 0 or 1. However, a relatively low probability such as 0.05 works the best.

One known shortcoming of CBCC3 is that it only relies on the magnitude of components' contributions and does not approximate the likelihood of their realization. In the future, we are interested in using sequential decision making techniques such as reinforcement learning methods to propose a more robust contribution-based cooperative co-evolutionary algorithm. We are also interested in conducting an in-depth exploration/exploitation analysis based on the methods used by LaTorre et al. [17], as well as a comparative study with the state-of-the-art [17, 18].

## Acknowledgement

This work was partially supported by an EPSRC grant (No. EP/K001523/1) and ARC Discovery grant (No. DP120102205).

## References

[1] T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds., *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

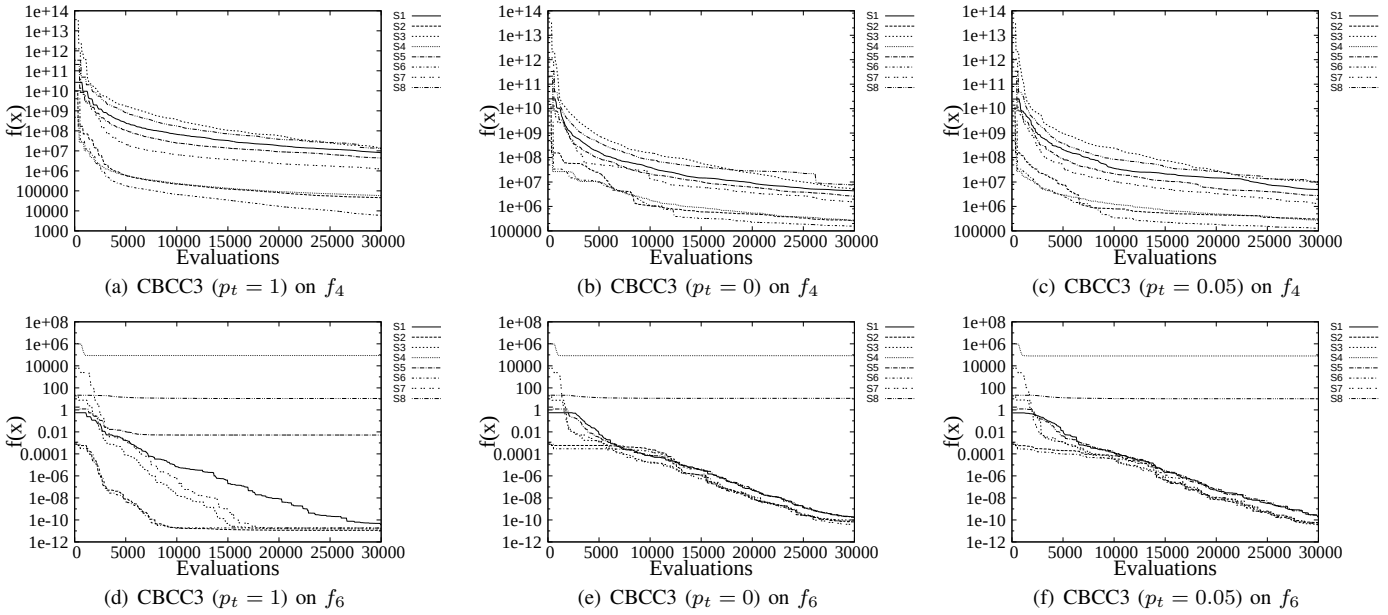
[2] R. E. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[3] M. A. Potter and 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.

[4] Y. Liu, X. Yao, Q. Zhao, and T. Higuchi, "Scaling up fast evolutionary programming with cooperative coevolution," in *Proc. of IEEE Congress on Evolutionary Computation*, 2001, pp. 1101–1108.

[5] M. Omidvar, X. Li, Y. Mei, and X. Yao, "Cooperative coevolution with differential grouping for large scale optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 18, no. 3, pp. 378–393, June 2014.

[6] Y. Mei, M. N. Omidvar, X. Li, and X. Yao, "A competitive divide-and-conquer algorithm for unconstrained




(a) CBCC3 ($p_t = 1$) on $f_4$ (b) CBCC3 ($p_t = 0$) on $f_4$ (c) CBCC3 ($p_t = 0.05$) on $f_4$

(d) CBCC3 ($p_t = 1$) on $f_6$ (e) CBCC3 ($p_t = 0$) on $f_6$ (f) CBCC3 ($p_t = 0.05$) on $f_6$

Figure 4: Convergence plots of individuals components for CBCC3 with $p_t = \{0, 0.05, 1\}$ on $f_4$ and $f_6$ (CEC'2013).

Table V: Standard deviation (StDev) among the final objective value of individual components. CBCC3 maintains a lower StDev than CBCC1, 2 and DECC. There is also a strong correlation between better overall performance and lower StDev.

| | Mean | | | | | | Standard deviation | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | CBCC3 | | | | | | CBCC3 | | |
| | DECC | CBCC1 | CBCC2 | $p_t = 1$ | $p_t = 0$ | $p_t = 0.05$ | DECC | CBCC1 | CBCC2 | $p_t = 1$ | $p_t = 0$ | $p_t = 0.05$ |
| $f_4$ | 1.53e+08 | 6.54e+07 | 9.03e+10 | 3.61e+07 | **2.18e+07** | 2.51e+07 | 7.28e+07 | 1.75e+07 | 1.63e+10 | 8.29e+06 | **3.68e+06** | 5.95e+06 |
| $f_5$ | 2.65e+06 | 2.29e+06 | 2.06e+06 | 2.23e+06 | 2.16e+06 | **1.97e+06** | 8.04e+05 | 6.56e+05 | 5.90e+05 | 6.94e+05 | 6.19e+05 | **5.75e+05** |
| $f_6$ | 8.74e+04 | 8.74e+04 | 8.35e+04 | 8.74e+04 | 8.74e+04 | **8.35e+04** | 2.92e+04 | 3.10e+04 | 2.90e+04 | 2.94e+04 | **2.88e+04** | 2.89e+04 |
| $f_7$ | 4.53e+07 | 6.23e+07 | 7.85e+07 | 4.68e+07 | 3.86e+05 | **3.27e+05** | 1.74e+07 | 1.91e+07 | 2.54e+07 | 1.65e+07 | 1.09e+07 | **7.54e+06** |
| $f_8$ | 5.43e+13 | 1.09e+13 | 1.90e+12 | 7.29e+10 | **2.28e+09** | 5.47e+09 | 2.06e+13 | 3.91e+12 | 3.87e+11 | 7.01e+10 | 5.52e+09 | **2.47e+09** |
| $f_9$ | 2.95e+08 | 2.34e+08 | 2.00e+08 | 2.08e+08 | **1.42e+08** | 1.58e+08 | 5.53e+07 | 4.31e+07 | 3.60e+07 | 3.83e+07 | **2.51e+07** | 2.95e+07 |
| $f_{10}$ | 7.05e+01 | 7.51e+01 | 7.17e+01 | 7.68e+01 | 7.46e+01 | **7.30e+01** | 1.04e+01 | 1.09e+01 | 1.09e+01 | 1.13e+01 | **1.09e+01** | **1.09e+01** |
| $f_{11}$ | 1.51e+10 | 1.41e+09 | 1.44e+09 | 1.07e+09 | **4.49e+08** | 6.31e+08 | 8.72e+09 | 5.89e+09 | 6.86e+09 | 3.62e+08 | **9.31e+07** | 1.16e+08 |

large-scale black-box optimization," *ACM Transactions on Mathematical Software (TOMS)*, 2016.

[7] M. N. Omidvar, M. Yang, Y. Mei, X. Li, and X. Yao, "IDG: A faster and more accurate differential grouping algorithm," University of Birmingham, School of Computer Science, Tech. Rep. CSR-15-04, September 2015.

[8] M. N. Omidvar, X. Li, and X. Yao, "Smart use of computational resources based on contribution for cooperative co-evolutionary algorithms," in *Proc. of Genetic and Evolutionary Computation Conference*. ACM, 2011, pp. 1115–1122.

[9] X. Li, K. Tang, M. N. Omidvar, Z. Yang, and K. Qin, "Benchmark functions for the CEC'2013 special session and competition on large-scale global optimization," RMIT University, Australia, Tech. Rep., 2013.

[10] M. N. Omidvar, X. Li, and K. Tang, "Designing benchmark problems for large-scale continuous optimization," *Information Sciences*, vol. 316, pp. 419 – 436, 2015.

[11] B. Kazimipour, M. Omidvar, X. Li, and A. Qin, "A sensitivity analysis of contribution-based cooperative co-evolutionary algorithms," in *Evolutionary Computation (CEC), 2015 IEEE Congress on*, 2015, pp. 417–424.

[12] K. Tang, X. Li, P. N. Suganthan, Z. Yang, and T. Weise, "Benchmark functions for the CEC'2010 special session and competition on large-scale global optimization," Nature Inspired Computation and Applications Laboratory, USTC, China, Tech. Rep., 2009.

[13] S. Mahdavi, M. E. Shiri, and S. Rahnamayan, "Metaheuristics in large-scale global continues optimization: A survey," *Information Sciences*, 2014.

[14] G. Trunfio, "Adaptation in cooperative coevolutionary optimization," in *Adaptation and Hybridization in Computational Intelligence*, 2015, vol. 18, pp. 91–109.

[15] Z. Yang, K. Tang, and X. Yao, "Self-adaptive differential evolution with neighborhood search," in *Proc. of IEEE Congress on Evolutionary Computation*, 2008, pp. 1110–1116.

[16] D. J. Sheskin, *Handbook of parametric and nonparametric statistical procedures*. CRC Press, 2003.

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

[18] A. LaTorre, S. Muelas, and J.-M. Peña, "Large scale global optimization: Experimental results with MOS-based hybrid algorithms," in *Evolutionary Computation (CEC), 2013 IEEE Congress on*, 2013, pp. 2742–2749.