2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

# A Novel Hybridization of Opposition-based Learning and Cooperative Co-evolutionary for Large-Scale Optimization

Borhan Kazimipour*, Mohammad Nabi Omidvar*,Xiaodong Li*, A. K. Qin*†
*School of Computer Science and Information Technology, RMIT University, Melbourne, 3000, Victoria, Australia
Email:{borhan.kazimipour, mohammad.omidvar, xiaodong.li, kai.qin}@rmit.edu.au
†School of Automation, Southeast University, Nanjing, China, 210096

***Abstract*—Opposition-based learning (OBL) and cooperative co-evolution (CC) have demonstrated promising performance when dealing with large-scale global optimization (LSGO) problems. In this work, we propose a novel framework for hybridizing these two techniques, and investigate the performance of simple implementations of this new framework using the most recent LSGO benchmarking test suite. The obtained results verify the effectiveness of our proposed OBL-CC framework. Moreover, some advanced statistical analyses reveal that the proposed hybridization significantly outperforms its component methods in terms of the quality of finally obtained solutions.**

## I. Introduction

Optimization problems are ubiquitous. With the advances in science and technology there is a growing need for solving complex optimization problems. Two major factors that contribute to the complexity of an optimization problem are dimensionality of the search space and variable interaction [1]. In the past decade, large-scale global optimization (LSGO) has been an active area of research in the evolutionary computing community [2], [3], [4].

A wide range of meta-heuristic algorithms have emerged, especially for solving large-scale optimization problems [5], [6], [7], [8], [9], [10], [11]. One paradigm that received the most attention in the field of evolutionary optimization is *cooperative co-evolution* (CC) [12]. Cooperative co-evolution follows a divide-and-conquer paradigm where a large-scale problem is decomposed into a set of lower-dimensional subproblems [13]. Each subproblem which is often easier to optimize is co-adapted in a round-robin fashion. The modular nature of a CC framework makes it ideal for solving large-scale problems. A major challenge in using cooperative co-evolution is finding the optimal decomposition of the decision variables into several subcomponents. Many different algorithms such as random grouping [5], delta grouping [14], variable interaction learning [9], and differential grouping [6] have been proposed for problem decomposition.

Another promising approach for dealing with large-scale problems is *opposition-based learning* (OBL) [15], [16]. OBL [16] was initially introduced to the evolutionary computing community in order to improve the performance of differential evolution (DE) [17]. OBL has been successfully applied to many applications such as multi-objective optimization, optimization in noisy environments, and large-scale optimization [18], [19]. The key concept of OBL is to check both the current estimates (candidate solutions) and their opposites simultaneously. The *central opposition theorem* [20] states that the probability that the opposite of a candidate solution is closer to the global optimum is higher than the probability of a second random guess. Consequently, considering both the estimate and its opposite increases the probability of finding a solution closer to the global optimum.

In this study, a novel framework is proposed to introduce the concept of OBL into a CC framework in order to improve its performance when dealing with large-scale optimization problems. Despite its simplicity, the proposed framework is very general such that any OBL and CC technique can be involved as its components. The most recent LSGO benchmark suite [4] is used in this study to demonstrate the effectiveness of the simplest implementations of the proposed framework. Finally, several advanced non-parametric statistical methods are applied to validate the significance of the findings.

The rest of the paper is organized as follows. A brief review of the CC framework and OBL concept is provided in Section II. The novel hybridization of OBL and CC is discussed in Section III. Section IV is dedicated to the empirical studies of the proposed methods, its comparison with a baseline method and the discussions of the obtained results. Some ideas for future works are also presented in Section V. Finally Section VI concludes the paper.

## II. Background

In this section a brief review of a general cooperative co-evolutionary framework, and the core concepts of opposition-based learning are presented.

### A. Cooperative Co-evolution

Cooperative Co-evolution (CC) [12] follows a divide-and-conquer approach to deal with complex optimization problems. The modular nature of a CC framework makes it ideal for large-scale optimization problems [5], [6], [11], [21], [22]. In the simplest form, CC divides a $D$-dimensional problem to $D$ 1-dimensional subproblems which are easier to optimize. It is clear that there are many different ways of subdividing a problem into a set of smaller subproblems. A major factor that makes a particular decomposition (subdivision) superior to another is variable interaction.

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

Variable interaction or non-separability is defined as the degree to which the quality of a variable is affected by values taken by other variables [23]. In an extreme case where every pair of decision variables interact with each other, the problems are referred to as fully non-separable. In practice, not every pair of the decision variables interact with each other. In a more common class of problems called partially separable problems, usually a subset of the decision variables interact, forming several clusters of interacting variables where there is no interaction between groups. Therefore, an ideal decomposition of variables in a CC framework is to automatically identify the underlying interaction structure of the decision variables and form the subcomponents such that the inter-subcomponent interactions are minimized.

Many decomposition methods have been proposed for a CC framework. Random grouping [5], [24] is one of the early attempts in tackling large-scale partially separable problems. In random grouping the decision variables are randomly allocated to a set of subcomponents the number of which is determined by the user. Then, the subcomponents are optimized in a round-robin fashion. In order to increase the probability of placing two interacting variables in a common subcomponent for several iterations, the decision variables are randomly allocated to subcomponents at the beginning of each cycle. Other decomposition methods such as delta grouping [14], variable interaction learning [9], and differential grouping [6] are more sophisticated techniques that have a higher accuracy in detecting interacting variables. Since the focus of this research is not on investigating the effect of various grouping techniques, we confined our experiments to random grouping only.

Algorithm 1 shows a general CC framework. The `grouping` function is used to decompose a problem into a set of smaller subproblems (line 1). This function represents any grouping method such as random grouping [5] or differential grouping [6]. The `rand` function is used to generate the initial random population which is then evaluated to find the initial best individual (lines 2, 3). The loop on line 4 forms the main co-evolutionary cycle. At the beginning of each cycle the subcomponents are reformed if a dynamic grouping algorithm such as random grouping [5] or delta grouping [14] is used. Static grouping algorithms such as differential grouping [6] are only called once at the beginning of the algorithm. The inner loop is used to iterate over the subcomponents and optimize them in a round-robin fashion (lines 8-13). The `optimizer` function is any evolutionary optimizer that is used to optimize each subcomponent for a predetermined number of fitness evolutions ($FE$).

It should be noted that the best individual that is found so far is passed to `optimizer` as a *context vector* which is used to evaluate the individuals in a subcomponent. This is where the cooperation happens in a CC framework. Use of a context vector is essential because the individuals in a subcomponent are incomplete solutions and cannot be directly evaluated using the objective function ($f$). Therefore, the variables of an individual must be evaluated within the context of a complete solution. There are other ways of creating a context vector such as randomly selecting an existing solution. However, in this study we use the best individual as the context vector. Once each subcomponent is optimized, the corresponding subcomponent ($subpop$) and the context vector ($best$) are updated (line 11).

**Algorithm 1:** CC($f, lbounds, ubounds, n, dynamic$)

```
1.  groups ← grouping(f, lbounds, ubounds, n)    ▷ grouping stage.
2.  pop ← rand(popsize, n)    ▷ optimization stage.
3.  (best, best_val) ← min(f(pop))
4.  for i ← 1 to cycles do
5.      if dynamic = True then
6.          groups ← grouping(f, lbounds, ubounds, n)
7.      end if
8.      for j ← 1 to size(groups) do
9.          indicies ← groups[j]
10.         subpop ← pop[:, indicies]
11.         (subpop, best, best_val) ← optimizer(f, best, subpop, FE)
12.         pop[:, indicies] ← subpop
13.     end for
14. end for
```

### B. Opposition-Based Learning

The concept of opposition-based learning (OBL) was originally introduced to the machine learning community by Tizhoosh in 2005 [25] and then to the evolutionary computing field by Rahnamayan in 2006 [18]. The key idea of optimization based on OBL is to simultaneously consider the estimates (candidate solutions generated by an EA) and their opposites. For a clear explanation of OBL, we need the concept of opposite numbers to be defined. Let $\mathbf{x} = (x_1, \ldots, x_D)$ be a $D$-dimensional point where $x_i$ are real-valued variables bounded by $[a_i, b_i]$ for $\forall i \in \{1, \ldots, D\}$. Then $\breve{\mathbf{x}}$ is the opposite of $\mathbf{x}$ where each of its coordinates is defined as $\breve{x}_i = a_i + b_i - x_i$.

Core theorems of opposition-based learning are summarized next.

**Theorem 1** (first opposition theorem)**.** *Given any function $f(x), x \in [a, b]$ with its global optimum at $x^\star \neq \frac{a+b}{2}$, for the estimate solution $x$ and its opposite $\breve{x}$ we have:*

$$P\left(|\breve{x} - x^\star| < |x - x^\star|\right) = \frac{1}{2},$$

*where $P(\cdot)$ is the probability function.*

Theorem 1 states that a candidate solution $x$ and its opposite $\breve{x}$ have the equal probability of being closer to the global optimum.

**Theorem 2** (second opposition theorem)**.** *For an increasingly monotone function $f(x)$ we have:*

$$P\left(f(x_r) < \max\left\{f(x), f(\breve{x})\right\}\right) = \frac{3}{4},$$

*where $x$ is the first random guess, $\breve{x}$ is the opposite point of $x$, and $x_r$ is the second random guess.*

**Theorem 3** (central opposition theorem)**.** *Assuming a black-box optimization problem $f(x)$, and letting $x$, and $x_r$ be the first and second random guesses from a uniform distribution over the interval $[a, b]$. We have:*

$$P\left(|\breve{x} - x^\star| < |x_r - x^\star|\right) > P\left(|x_r - x^\star| < |\breve{x} - x^\star|\right).$$

Theorem 3 states that the probability of the opposite of a candidate solution $\breve{x}$ being closer to the global optimum is higher than the probability of a second random guess being closer to the global optimum.

As a consequence of opposition-based learning theorems, an optimization algorithm should simultaneously evaluate both the candidate solutions and their opposites in order to increase the probability of finding solutions closer to the global optimum. In the context of evolutionary algorithms, instead of performing pairwise comparisons between each individual and its opposite, usually the original population of estimates and the population of their opposites are merged and then the top half of the candidate solutions (as ranked according to fitness function) are selected for the next iteration and the remaining candidate solutions are eliminated.

Several variants of OBL including quasi OBL (QOBL) [26], quasi-reflection OBL (QROBL) [27], generalized OBL (GOBL) [28], [29] and current optimum OBL (COOBL) [30] have been proposed. The main difference between these variations is in the way they produce opposite points. In QOBL, for example, *quasi-opposites* are used instead of actual opposites. A quasi-opposite of a point $\mathbf{x}$ is generally defined as a real number randomly chosen from a uniform distribution bounded by its opposite $\breve{\mathbf{x}}$ and the middle point $\mathbf{m} = (m_1, \ldots, m_D)$ where $m_i = a_i + \frac{a_i+b_i}{2}, \forall i \in \{1, \ldots, D\}$. As another example, in COOBL, the current best estimate is used as the opposition pivot instead of the middle point [30].

OBL can be used for population initialization at the beginning of the optimization, and for generating random variations during the optimization process. In initialization step, a temporary population (a.k.a. original population) is generated using conventional random number generators or more advanced initialization techniques [31], [32]. Then, the opposite population is produced and merged with the original population. Finally, the best subpopulation of all generated points is selected based on the fitness function and treated as the initial population of evolutionary algorithms [33].

Similar approach to OBL initialization can also be applied to the current population during the evolutionary process. This idea which is generally referred as *generation jumping* works as follows. After generating new population by conventional evolutionary operators (e.g., crossover and mutation), the opposite population is calculated with probability $J_r$ (i.e., jumping rate) [34]. Then, both populations are merged and all individuals are evaluated. Finally, the fittest individuals are selected to survive and the worst ones are eliminated. Since the boundaries of variables (i.e., $a_i$ and $b_i$) may change during each iteration, the opposite population must be calculated dynamically according to the current boundaries (or in COOBL according to the current best population member) [30].

## III. Proposed Methods

In this section the proposed method is discussed in details. As mentioned in Section II-A, CC framework generally works as follow. The decision variables of a given problem are divided into a number of groups using a decomposition method. Then, an arbitrary optimization algorithm (e.g. DE) is employed to optimize each subcomponent in a round-robin fashion for a predetermined number of iterations.

Generally, the members of a subcomponent only contain part of the decision variables. Therefore, a context vector is needed for their evaluation. The context vector can be a randomly chosen member or the current best candidate which contains all $D$ variables. Next, the corresponding variables of each subcomponent member are plugged into the context vector and evaluated using the fitness function.

**Algorithm 2:** OBL-CC($f, lbounds, ubounds, n, dynamic, J_r$)

```
1.  groups ← grouping(f, lbounds, ubounds, n)    ▷ grouping stage.
2.  pop ← rand(popsize, n)    ▷ optimization stage.
3.  (best, best_val) ← min(f(pop))
4.  for i ← 1 to cycles do
5.      if dynamic = True then
6.          groups ← grouping(f, lbounds, ubounds, n)
7.      end if
8.      for j ← 1 to size(groups) do
9.          indicies ← groups[j]
10.         subpop ← pop[:, indicies]
11.         (subpop, best, best_val) ← optimizer(f, best, subpop, FE)
12.         if rand(1, 1) < J_r then
13.             oppsubpop ← opposite(subpop, lbounds, ubounds, n)
14.             allsubpop ← union(subpop, oppsubpop, n)
15.             subpop ← select(f, best, allsubpop, FE)
16.         end if
17.         pop[:, indicies] ← subpop
18.     end for
19. end for
```

In the traditional OBL framework, the OBL operators are applied to the entire population for all the decision variables. Unlike the traditional OBL, in a CC framework the OBL operators are applied to each subcomponent separately. More specifically, suppose a $D$-dimensional problem is divided into $N$ subcomponents, each of which has $d_j$ variables $\forall j \in \{1, \ldots, N\}$ where $D = d_1 + \cdots + d_N$. Then, for the $j$th subcomponent, the opposite of every candidate solution is calculated only with respect to its $d_j$ dimensions and all other $D - d_j$ variables remain unchanged. For evaluation, the opposite points are plugged into the same context vector as used to evaluate the original candidates.

Algorithm 2 shows the general proposed framework which we call OBL-CC. The framework is very similar to a traditional CC framework except for the inclusion of OBL operations on lines 12-16. OBL's generation jumping happens with $J_r$ probability (line 12). This means that if the random number generated by the `rand` function is less than $J_r$, the algorithm applies OBL operator on the $j$th subcomponent. Otherwise, normal CC procedure will be continued. Note that different schemes for generation jumping have been proposed [34]. For example, $J_r$ can be a fixed value or a monotonically increasing or decreasing function of generation number. All variants of generation jumping schemes can be applied to the proposed framework.

On line 13 the opposite solutions of the $j$th subcomponent is generated using the `opposite` function. Here the `opposite` function represents any OBL variation such as QOBL [26] or COOBL [30]. In this study, for example, OBL [18], and QOBL [26] strategies are examined. Then, the original subcomponent and its opposite are merged using the `union` function. Finally, the merged subcomponent is evaluated using the $best$ context vector, and the top half of the candidate solutions is chosen using the `select` function to enter the next iteration (lines 14, 15). This procedure must be repeated for every subcomponent when the random number produced by the `rand` function is less than $J_r$.

## IV. EXPERIMENTS

In this section, we compare the performance of two simple implementations of the proposed framework with DECC [5] as the baseline method.

### A. Experimental Setup

For the hybridized algorithm two opposition strategies are used: OBL [18] and QOBL [26]. These two strategies are the two most widely used opposition strategies according to [19]. In order to perform a fair comparison between algorithms, the SaNSDE [22] algorithm is used as the subcomponent optimizer, in all experiments. SaNSDE is a variation of DE that dynamically adapts the crossover rate and the scaling factor of DE. All common control parameters (e.g. population size and DE strategies) are also fixed for all algorithms.

For each single strategy, four $J_r$ schemes are examined including two constants (0.3 and 0.6), one monotonically increasing (0∼0.6) and one monotonically decreasing (0.6∼0) schemes. In succeeding parts and tables, the generation jumping rate schemes are denoted in the parenthesis following the algorithm name. For example, QOBL-CC(0∼0.6) means the hybrid of QOBL and CC with monotonically increasing jumping rate from 0 to 0.6. See [34] for more details regarding generation jumping rate schemes.

In all experiments, CEC'2013 LSGO benchmark functions [4] are used as the testbed. Each algorithm and function is evaluated for 51 independent runs. Following the framework used in [35] and [33] , the $i$th runs of all algorithms are initialized by the same seed while $i$th and $j$th ($i \neq j$) runs of any single algorithm have different initial seeds.

To be consistent with the CEC'2013 competition guidelines, the maximum number of function evaluations is limited to $3e + 6$ for each run. Note that OBL-CC and QOBL-CC call the fitness function in the selection part (Algorithm 2, line 15) as well as in the optimization part (Algorithm 2, line 11). For both parts the evaluations are counted and the maximum number of total function evaluations is limited to $3e+6$. Consequently, the proposed framework has no overhead complexity in terms of the number of function evaluation.

### B. Benchmark Suite

The CEC'2013 LSGO benchmark suite is currently the latest proposed benchmark in the field of large-scale optimization [4]. The suite consists of 15 1000-dimensional unconstrained continuous functions. The functions are grouped into five distinct categories according to their degree of separability:

- G1: fully separable functions ($f_1$-$f_3$);
- G2: partially separable functions with a separable subcomponent ($f_4$-$f_7$);
- G3: partially separable functions with no separable subcomponents ($f_8$-$f_{11}$);
- G4: overlapping functions ($f_{12}$-$f_{14}$);
- G5: fully non-separable function ($f_{15}$).

To improve previously proposed benchmark suites (e.g., CEC'2010 [3]), new functions with non-uniform subcomponent sizes and overlapping subcomponents have been introduced to the new suite. Furthermore, new transformations such as ill-conditioning, symmetry breaking and irregularities have been added to the CEC'2013 LSGO benchmark functions. More information regarding this suite can be found in [4].

### C. Analysis and Discussions

The results obtained from 51 independent runs of DECC are compared with the performance of OBL-CC and QOBL-CC using four generation jumping schemes which are presented in Table I. The statistics presented in the table shows that both OBL and QOBL successfully improved the CC framework (DECC in this case).

Table II compares eight hybrid OBL/QOBL methods with DECC as the control method on five groups of benchmark functions (see Section IV-B). Following [31], each cell of the win-draw-loss table (see Table II) consists of three numbers in $\alpha$-$\beta$-$\gamma$ style. In each triplet, $\alpha$ denotes the number of functions on the corresponding group which the competitor (i.e. hybrid method) significantly improves the control method (i.e. DECC). The next number, $\beta$, shows how many times DECC and its hybrid competitor perform statistically similar. And, $\gamma$ denotes the number of functions that hybridization has adverse effects on the performance of DECC. Note that in this table two algorithms are considered to be significantly different if the $p$-value of Wilcoxon rank-sum test is less than 0.05, and statistically similar otherwise.

As the last row of Table II confirms, except for OBL-CC(0.6) and OBL-CC(0.6∼0), the hybridization generally improves the performance of DECC (i.e., $\alpha > \gamma$). Moreover, QOBL-CC(0.6∼0) with the greatest $\alpha$ value shows the best performance (according to win-draw-loss measure) while OBL-CC(0.3) with the smallest $\gamma$ value is the most reliable hybridization (i.e., has the least risk of failure) among all the others.

Table II also indicates that the family of QOBL-CC is more effective in dealing with problems with some degrees of non-separability (G2-G5), while is less recommended to be used in dealing with fully separable problems (G1).

Table III provides the results of nWins procedure. nWins is a N×N comparison method which compares each single method with all the others. Following [36], when an algorithm significantly outperforms one of its competitors, its nWins score is increased by $+1$ and the looser is penalized by $-1$. If both algorithms perform statistically similar, their nWins scores remain unchanged.

The provided nWins scores in Table III confirm our findings from the win-draw-loss table (see Table II). All hybrid methods obtain better nWins scores than DECC which shows the success of hybridization of OBL/QOBL with CC framework. Table III also confirms that QOBL-CC methods show weak performance in dealing with fully separable functions (G1). However, for the functions with some degrees of non-separability QOBL-CC variants show very good performance. Indeed, for G2-G5, the best performer is always from QOBL-CC family.

TABLE I. MAIN STATISTICS OF 51 INDEPENDENT RUNS OF TWO HYBRID ALGORITHMS (OBL-CC AND QOBL-CC) USING FOUR JUMPING RATE SCHEMES (INDICATED IN PARENTHESIS AS 0.3, 0.6, 0∼0.6 AND 0.6∼0) WITH THE BASELINE METHOD (DECC).

| Functions | Stats | DECC | OBLCC(0.3) | OBLCC(0.6) | OBLCC(0∼0.6) | OBL(0.6∼0) | QOBLCC(0.3) | QOBLCC(0.6) | QOBLCC(0∼0.6) | QOBLCC(0.6∼0) |
|---|---|---|---|---|---|---|---|---|---|---|
| $f_1$ | min | 0.00e+00 | 0.00e+00 | 0.00e+00 | 0.00e+00 | 0.00e+00 | 2.02e-09 | 9.87e+03 | 8.94e-11 | 1.61e-10 |
| | median | 0.00e+00 | 0.00e+00 | 0.00e+00 | 0.00e+00 | 0.00e+00 | 2.81e-05 | 1.12e+06 | 5.75e-07 | 3.19e-08 |
| | max | 0.00e+00 | 4.75e-23 | 4.25e-21 | 1.04e-22 | 1.43e-22 | 1.39e+04 | 9.68e+07 | 2.71e+06 | 5.81e-03 |
| | mean | 0.00e+00 | 9.32e-25 | 1.26e-22 | 2.05e-24 | 2.84e-24 | 6.66e+02 | 6.72e+06 | 1.16e+05 | 1.83e-04 |
| | std | 0.00e+00 | 6.65e-24 | 5.93e-22 | 1.46e-23 | 2.00e-23 | 2.52e+03 | 1.74e+07 | 4.71e+05 | 9.39e-04 |
| $f_2$ | min | 2.37e+02 | 2.59e+01 | 1.69e+01 | 1.39e+01 | 9.95e+01 | 4.00e+03 | 5.74e+03 | 5.54e+03 | 1.60e+02 |
| | median | 2.75e+02 | 3.88e+01 | 2.39e+01 | 2.59e+01 | 1.14e+02 | 4.68e+03 | 6.41e+03 | 6.46e+03 | 2.09e+02 |
| | max | 3.50e+02 | 6.07e+01 | 3.68e+01 | 4.78e+01 | 1.50e+02 | 5.72e+03 | 7.28e+03 | 7.19e+03 | 2.92e+02 |
| | mean | 2.78e+02 | 3.96e+01 | 2.50e+01 | 2.64e+01 | 1.18e+02 | 4.72e+03 | 6.42e+03 | 6.48e+03 | 2.13e+02 |
| | std | 2.84e+01 | 7.65e+00 | 4.77e+00 | 5.94e+00 | 1.10e+01 | 3.83e+02 | 3.75e+02 | 4.07e+02 | 3.05e+01 |
| $f_3$ | min | 1.28e-13 | 1.03e-13 | 8.53e-14 | 9.95e-14 | 8.53e-14 | 1.15e+01 | 2.00e+00 | 3.04e+00 | 4.29e-05 |
| | median | 1.39e-13 | 1.10e-13 | 9.95e-14 | 1.10e-13 | 1.10e-13 | 1.23e+01 | 1.08e+01 | 1.09e+01 | 1.26e+00 |
| | max | 1.39e-13 | 1.10e-13 | 8.43e-02 | 1.14e-13 | 1.10e-13 | 1.31e+01 | 1.19e+01 | 1.22e+01 | 1.41e+00 |
| | mean | 1.36e-13 | 1.09e-13 | 1.65e-03 | 1.09e-13 | 1.07e-13 | 1.22e+01 | 9.86e+00 | 1.01e+01 | 1.16e+00 |
| | std | 4.19e-15 | 2.13e-15 | 1.18e-02 | 2.49e-15 | 5.61e-15 | 3.65e-01 | 2.43e+00 | 2.54e+00 | 3.49e-01 |
| $f_4$ | min | 3.74e+09 | 7.06e+09 | 1.33e+10 | 7.55e+09 | 9.35e+09 | 1.51e+09 | 1.49e+09 | 2.10e+09 | 2.66e+09 |
| | median | 1.68e+10 | 1.86e+10 | 3.31e+10 | 1.89e+10 | 2.44e+10 | 5.38e+09 | 5.88e+09 | 5.84e+09 | 6.29e+09 |
| | max | 3.85e+10 | 5.57e+10 | 8.77e+10 | 5.38e+10 | 6.24e+10 | 1.12e+10 | 1.91e+10 | 1.26e+10 | 1.49e+10 |
| | mean | 1.73e+10 | 2.11e+10 | 3.59e+10 | 2.15e+10 | 2.52e+10 | 5.36e+09 | 6.89e+09 | 6.06e+09 | 6.47e+09 |
| | std | 7.43e+09 | 9.86e+09 | 1.49e+10 | 9.93e+09 | 1.04e+10 | 2.02e+09 | 3.56e+09 | 2.64e+09 | 2.73e+09 |
| $f_5$ | min | 3.47e+06 | 3.48e+06 | 2.92e+06 | 3.51e+06 | 3.62e+06 | 3.72e+06 | 3.06e+06 | 3.06e+06 | 3.30e+06 |
| | median | 5.65e+06 | 5.17e+06 | 5.41e+06 | 5.39e+06 | 5.45e+06 | 5.67e+06 | 5.89e+06 | 5.77e+06 | 5.50e+06 |
| | max | 9.44e+06 | 8.38e+06 | 8.69e+06 | 7.96e+06 | 1.14e+07 | 1.05e+07 | 8.66e+06 | 8.86e+06 | 1.20e+07 |
| | mean | 5.82e+06 | 5.42e+06 | 5.51e+06 | 5.54e+06 | 5.60e+06 | 5.98e+06 | 5.91e+06 | 5.90e+06 | 5.74e+06 |
| | std | 1.32e+06 | 1.29e+06 | 1.19e+06 | 1.10e+06 | 1.37e+06 | 1.26e+06 | 1.27e+06 | 1.23e+06 | 1.64e+06 |
| $f_6$ | min | 1.14e-03 | 3.52e-10 | 1.57e+00 | 4.41e-04 | 3.00e-04 | 3.95e+02 | 8.29e+04 | 1.00e+05 | 1.84e+00 |
| | median | 5.46e+04 | 5.50e+02 | 7.96e+02 | 6.21e+02 | 7.16e+02 | 8.64e+04 | 1.36e+05 | 1.40e+05 | 6.58e+02 |
| | max | 1.23e+05 | 1.05e+05 | 1.01e+05 | 9.34e+04 | 1.13e+05 | 1.05e+06 | 1.94e+05 | 1.86e+05 | 1.27e+05 |
| | mean | 3.96e+04 | 1.66e+04 | 3.06e+04 | 2.08e+04 | 3.00e+04 | 9.53e+04 | 1.37e+05 | 1.40e+05 | 3.61e+04 |
| | std | 3.97e+04 | 3.07e+04 | 3.67e+04 | 3.05e+04 | 3.57e+04 | 1.40e+05 | 2.39e+04 | 1.95e+04 | 4.14e+04 |
| $f_7$ | min | 6.95e+07 | 3.72e+07 | 1.70e+07 | 2.71e+07 | 4.27e+07 | 8.21e+05 | 4.06e+06 | 1.28e+06 | 1.09e+06 |
| | median | 3.16e+08 | 1.57e+08 | 5.86e+07 | 1.77e+08 | 2.98e+08 | 2.10e+06 | 8.85e+06 | 3.51e+06 | 9.26e+06 |
| | max | 1.00e+09 | 1.86e+09 | 5.41e+08 | 2.52e+09 | 4.31e+09 | 7.22e+06 | 4.44e+07 | 1.05e+07 | 3.10e+07 |
| | mean | 3.71e+08 | 3.57e+08 | 1.05e+08 | 3.07e+08 | 4.61e+08 | 2.55e+06 | 1.08e+07 | 3.83e+06 | 1.09e+07 |
| | std | 2.43e+08 | 4.49e+08 | 1.18e+08 | 4.02e+08 | 6.79e+08 | 1.47e+06 | 6.51e+06 | 1.69e+06 | 7.44e+06 |
| $f_8$ | min | 9.83e+13 | 1.05e+14 | 1.26e+14 | 1.21e+14 | 1.25e+14 | 4.66e+13 | 5.61e+13 | 5.47e+13 | 5.70e+13 |
| | median | 3.01e+14 | 3.02e+14 | 3.53e+14 | 3.39e+14 | 3.29e+14 | 1.61e+14 | 1.92e+14 | 1.98e+14 | 1.84e+14 |
| | max | 7.41e+14 | 6.78e+14 | 6.46e+14 | 6.46e+14 | 6.57e+14 | 4.92e+14 | 3.54e+14 | 3.57e+14 | 5.21e+14 |
| | mean | 2.97e+14 | 3.17e+14 | 3.56e+14 | 3.41e+14 | 3.30e+14 | 1.74e+14 | 1.89e+14 | 1.96e+14 | 1.99e+14 |
| | std | 1.16e+14 | 1.20e+14 | 1.18e+14 | 1.16e+14 | 1.26e+14 | 7.80e+13 | 7.16e+13 | 7.23e+13 | 9.71e+13 |
| $f_9$ | min | 2.73e+08 | 2.43e+08 | 2.15e+08 | 2.25e+08 | 2.84e+08 | 2.64e+08 | 2.18e+08 | 2.16e+08 | 2.16e+08 |
| | median | 4.13e+08 | 4.10e+08 | 3.71e+08 | 4.11e+08 | 4.32e+08 | 4.00e+08 | 4.10e+08 | 4.05e+08 | 3.81e+08 |
| | max | 9.86e+08 | 7.75e+08 | 6.56e+08 | 5.71e+08 | 7.73e+08 | 7.51e+08 | 5.73e+08 | 6.85e+08 | 6.79e+08 |
| | mean | 4.44e+08 | 4.29e+08 | 3.85e+08 | 4.03e+08 | 4.42e+08 | 4.13e+08 | 4.13e+08 | 4.15e+08 | 3.88e+08 |
| | std | 1.30e+08 | 1.20e+08 | 1.00e+08 | 9.08e+07 | 9.79e+07 | 9.81e+07 | 7.40e+07 | 7.99e+07 | 8.43e+07 |
| $f_{10}$ | min | 6.57e+06 | 1.47e+05 | 6.72e+06 | 6.70e+06 | 6.71e+06 | 7.54e+06 | 1.62e+05 | 1.62e+05 | 4.72e+06 |
| | median | 1.45e+07 | 1.31e+07 | 1.52e+07 | 1.52e+07 | 1.38e+07 | 1.55e+07 | 1.27e+07 | 1.27e+07 | 1.39e+07 |
| | max | 8.93e+07 | 8.94e+07 | 8.94e+07 | 8.94e+07 | 8.94e+07 | 8.95e+07 | 9.03e+07 | 8.95e+07 | 8.93e+07 |
| | mean | 3.37e+07 | 3.73e+07 | 3.80e+07 | 3.79e+07 | 3.24e+07 | 3.49e+07 | 2.97e+07 | 2.98e+07 | 3.37e+07 |
| | std | 3.45e+07 | 3.70e+07 | 3.65e+07 | 3.65e+07 | 3.35e+07 | 3.38e+07 | 3.35e+07 | 3.33e+07 | 3.45e+07 |
| $f_{11}$ | min | 4.59e+10 | 5.58e+10 | 9.17e+10 | 5.32e+10 | 5.63e+10 | 1.17e+08 | 1.67e+08 | 1.26e+08 | 1.23e+08 |
| | median | 1.87e+11 | 2.01e+11 | 2.20e+11 | 1.82e+11 | 2.33e+11 | 1.78e+08 | 2.82e+08 | 2.47e+08 | 2.78e+08 |
| | max | 8.82e+11 | 1.31e+12 | 8.09e+11 | 6.03e+11 | 7.74e+11 | 2.97e+08 | 7.72e+08 | 4.38e+08 | 5.60e+08 |
| | mean | 2.29e+11 | 2.53e+11 | 2.47e+11 | 2.26e+11 | 2.68e+11 | 1.79e+08 | 2.99e+08 | 2.45e+08 | 2.94e+08 |
| | std | 1.80e+11 | 2.15e+11 | 1.48e+11 | 1.31e+11 | 1.45e+11 | 3.54e+07 | 1.03e+08 | 7.00e+07 | 9.18e+07 |
| $f_{12}$ | min | 1.22e+03 | 1.10e+03 | 1.31e+03 | 1.11e+03 | 1.09e+03 | 1.84e+03 | 8.83e+04 | 1.61e+03 | 1.90e+03 |
| | median | 1.40e+03 | 1.28e+03 | 1.48e+03 | 1.34e+03 | 1.22e+03 | 2.49e+03 | 3.49e+07 | 3.11e+03 | 2.15e+03 |
| | max | 1.63e+03 | 1.51e+03 | 2.04e+03 | 1.54e+03 | 1.43e+03 | 9.37e+05 | 2.39e+09 | 1.45e+09 | 2.65e+03 |
| | mean | 1.40e+03 | 1.27e+03 | 1.50e+03 | 1.33e+03 | 1.23e+03 | 3.23e+04 | 2.35e+08 | 6.65e+07 | 2.18e+03 |
| | std | 8.65e+01 | 8.75e+01 | 1.25e+02 | 8.91e+01 | 7.66e+01 | 1.50e+05 | 4.53e+08 | 2.69e+08 | 1.51e+02 |
| $f_{13}$ | min | 1.41e+10 | 1.53e+10 | 1.65e+10 | 2.01e+10 | 1.83e+10 | 1.79e+08 | 2.96e+08 | 3.11e+08 | 9.69e+08 |
| | median | 3.03e+10 | 3.15e+10 | 3.22e+10 | 3.15e+10 | 3.71e+10 | 4.63e+08 | 7.71e+08 | 6.74e+08 | 2.36e+09 |
| | max | 4.47e+10 | 6.23e+10 | 5.76e+10 | 6.77e+10 | 7.94e+10 | 8.64e+08 | 1.41e+09 | 1.21e+09 | 6.43e+09 |
| | mean | 3.14e+10 | 3.19e+10 | 3.38e+10 | 3.24e+10 | 3.67e+10 | 4.75e+08 | 7.86e+08 | 6.97e+08 | 2.45e+09 |
| | std | 7.66e+09 | 7.93e+09 | 9.70e+09 | 8.58e+09 | 1.17e+10 | 1.45e+08 | 2.52e+08 | 2.07e+08 | 9.77e+08 |
| $f_{14}$ | min | 1.66e+11 | 2.10e+11 | 1.81e+11 | 2.02e+11 | 2.26e+11 | 9.26e+07 | 2.63e+08 | 2.81e+08 | 1.67e+09 |
| | median | 4.46e+11 | 6.21e+11 | 5.54e+11 | 5.50e+11 | 6.39e+11 | 6.42e+08 | 2.50e+09 | 3.01e+09 | 1.36e+10 |
| | max | 8.92e+11 | 1.13e+12 | 1.30e+12 | 1.10e+12 | 1.68e+12 | 4.46e+09 | 1.54e+10 | 1.45e+10 | 1.08e+11 |
| | mean | 4.69e+11 | 6.21e+11 | 6.26e+11 | 5.85e+11 | 6.61e+11 | 1.09e+09 | 3.31e+09 | 3.66e+09 | 1.66e+10 |
| | std | 1.84e+11 | 2.14e+11 | 2.78e+11 | 2.23e+11 | 2.69e+11 | 1.10e+09 | 2.92e+09 | 3.01e+09 | 1.65e+10 |
| $f_{15}$ | min | 4.64e+07 | 4.31e+07 | 4.15e+07 | 4.42e+07 | 4.59e+07 | 2.38e+07 | 1.88e+07 | 2.28e+07 | 2.50e+07 |
| | median | 5.81e+07 | 5.55e+07 | 5.01e+07 | 5.26e+07 | 5.78e+07 | 3.04e+07 | 2.81e+07 | 2.85e+07 | 3.18e+07 |
| | max | 6.99e+07 | 6.22e+07 | 7.20e+07 | 7.16e+07 | 7.04e+07 | 4.45e+07 | 3.89e+07 | 3.95e+07 | 3.96e+07 |
| | mean | 5.79e+07 | 5.45e+07 | 5.12e+07 | 5.39e+07 | 5.80e+07 | 3.10e+07 | 2.82e+07 | 2.93e+07 | 3.22e+07 |
| | std | 5.35e+06 | 4.90e+06 | 5.49e+06 | 5.94e+06 | 5.40e+06 | 4.03e+06 | 4.07e+06 | 4.00e+06 | 3.30e+06 |

TABLE II. THE WIN-DRAW-LOSS STATISTICS OF TWO HYBRID ALGORITHMS (OBL-CC AND QOBL-CC) USING FOUR JUMPING RATE SCHEMES (INDICATED IN PARENTHESIS AS 0.3, 0.6, 0∼0.6 AND 0.6∼0) WHILE DECC IS SET AS THE CONTROL METHOD.

| Groups | OBLCC(0.3) | OBLCC(0.6) | OBLCC(0∼0.6) | OBLCC(0.6∼0) | QOBLCC(0.3) | QOBLCC(0.6) | QOBLCC(0∼0.6) | QOBLCC(0.6∼0) |
|---|---|---|---|---|---|---|---|---|
| G1 | 2-1-0 | 1-0-2 | 2-1-0 | 2-1-0 | 0-0-3 | 0-0-3 | 0-0-3 | 1-0-2 |
| G2 | 2-2-0 | 1-2-1 | 2-1-1 | 0-3-1 | 2-1-1 | 2-1-1 | 2-1-1 | 2-2-0 |
| G3 | 0-4-0 | 1-2-1 | 0-4-0 | 0-3-1 | 2-2-0 | 2-2-0 | 2-2-0 | 3-1-0 |
| G4 | 1-1-1 | 0-1-2 | 1-1-1 | 1-0-2 | 2-0-1 | 2-0-1 | 2-0-1 | 2-0-1 |
| G5 | 1-0-0 | 1-0-0 | 1-0-0 | 0-1-0 | 1-0-0 | 1-0-0 | 1-0-0 | 1-0-0 |
| Total | 6-8-1 | 4-5-6 | 6-7-2 | 3-8-4 | 7-3-5 | 7-3-5 | 7-3-5 | 9-3-3 |

TABLE III. THE RESULTS OF NWINS PROCEDURE ON TWO HYBRID ALGORITHMS (OBL-CC AND QOBL-CC) USING FOUR JUMPING RATE SCHEMES (INDICATED IN PARENTHESIS AS 0.3, 0.6, 0∼0.6 AND 0.6∼0) AND DECC AS THE BASELINE METHOD.

| Functions | DECC | OBLCC(0.3) | OBLCC(0.6) | OBLCC(0∼0.6) | OBLCC(0.6∼0) | QOBLCC(0.3) | QOBLCC(0.6) | QOBLCC(0∼0.6) | QOBLCC(0.6∼0) |
|---|---|---|---|---|---|---|---|---|---|
| $f_1$ | 5 | 5 | 0 | 5 | 5 | -5 | -8 | -5 | -2 |
| $f_2$ | -2 | 4 | 7 | 7 | 2 | -4 | -7 | -7 | 0 |
| $f_3$ | 2 | 5 | 0 | 5 | 8 | -8 | -5 | -5 | -2 |
| G1: | 5 | 14 | 7 | 17 | 15 | -17 | -20 | -17 | -4 |
| $f_4$ | -1 | -2 | -8 | -3 | -6 | 7 | 4 | 5 | 4 |
| $f_5$ | 0 | 3 | 0 | 0 | 0 | -1 | -1 | -1 | 0 |
| $f_6$ | 1 | 7 | 2 | 4 | 2 | -4 | -7 | -7 | 2 |
| $f_7$ | -7 | -4 | 0 | -4 | -5 | 8 | 3 | 6 | 3 |
| G2: | -7 | 4 | -6 | -3 | -9 | 10 | -1 | 3 | 9 |
| $f_8$ | -3 | -4 | -5 | -4 | -4 | 5 | 5 | 5 | 5 |
| $f_9$ | -2 | 0 | 2 | 0 | -2 | 0 | 0 | 0 | 2 |
| $f_{10}$ | 0 | 0 | 0 | 0 | 0 | -2 | 1 | 1 | 0 |
| $f_{11}$ | -3 | -4 | -4 | -4 | -5 | 8 | 3 | 6 | 3 |
| G3: | -8 | -8 | -7 | -8 | -11 | 11 | 9 | 12 | 10 |
| $f_{12}$ | 2 | 6 | 0 | 4 | 8 | -5 | -8 | -5 | -2 |
| $f_{13}$ | -3 | -3 | -4 | -3 | -7 | 8 | 5 | 5 | 2 |
| $f_{14}$ | 0 | -5 | -5 | -5 | -5 | 8 | 5 | 5 | 2 |
| G4: | -1 | -2 | -9 | -4 | -4 | 11 | 2 | 5 | 2 |
| $f_{15}$ | -7 | -3 | 0 | -3 | -7 | 4 | 7 | 7 | 2 |
| G5 | -7 | -3 | 0 | -3 | -7 | 4 | 7 | 7 | 2 |
| Total | -18 | 5 | -15 | -1 | -16 | 19 | -3 | 10 | 19 |

Furthermore, Table III denotes that DECC is never the single best method among the others. In fact, except for $f_1$ which DECC shares the first position with three variants of OBL-CC, it is one of the worst performers on all functions. The last row of Table III confirms that DECC is the weakest algorithm amongst the others. Consequently, we can conclude that OBL and QOBL hybridization (even with suboptimal $J_r$ values) significantly improve DECC.

To avoid family-wise error rate (FWER) and have a stronger conclusion, Friedman test is employed in this study [37]. This procedure is one of the state-of-the-art algorithms for 1×N and N×N comparisons [38]. In this experiment, Friedman procedure is repeated for each single function, group and entire benchmark functions. In all applications, the number of runs are carefully chosen to be consistent with the suggestions provided in [39].

The best and the worst performers according to Friedman's ranking is provided in Table IV. The results obtained from Friedman's ranking are fully compatible with the findings from win-draw-loss and nWins procedures (Tables II and III). As shown in Table IV, except for $f_1$ which DECC performs better than some of the other methods, on all the other functions the proposed hybridizations improve the performance of DECC. This means that the proposed method is effective in solving large-scale problems.

Friedman's ranking (see Table IV) also confirms that QOBL-CC is not as effective as the other hybrid methods in dealing with fully separable problems. However, on partially separable and fully non-separable functions, QOBL-CC variants are always the best performers among the examined methods. The last row of Table IV denotes that DECC is in general the worst algorithm while QOBL-CC(0∼0.6) is the top performer. This clearly confirms that the proposed hybridization successfully improves the performance of DECC on a wide range of large-scale benchmark functions.

Note that several post-hoc procedures (including Li's adjusted $p$-value [40]) are applied to validate Friedman's ranking. These procedures are recommended to be used to compare all methods against the control method. Here, DECC is selected as the control method in order to determine in which cases the top performer (according to Friedman's ranking) significantly improves DECC and in which cases the superior algorithm and DECC perform statistically similar. The results of this experiment are provided in Table IV.

The results of Li's post-hoc procedure are summarized in the last column of Table IV. In that column, 'yes' means the best method performs significantly better than DECC. On the other hand, 'no' in that column indicates the improvements

TABLE IV. THE BEST AND THE WORST METHODS ARE REPORTED BASED ON FRIEDMAN'S RANKING. THE LAST COLUMN SHOWS WHETHER ONE OF THE HYBRIDIZED ALGORITHMS SIGNIFICANTLY IMPROVES THE BASELINE (DECC) OR NOT. STATISTICAL SIGNIFICANCE IS CALCULATED BASED ON THE LI'S ADJUSTED $p$-VALUES [40].

| | Best Method | Worst Method | Improvement |
|---|---|---|---|
| $f_1$ | DECC | QOBL(0.6) | No |
| $f_2$ | OBL(0.6) | QOBL(0.6∼0) | Yes |
| $f_3$ | OBL(0.6) | QOBL(0.3) | Yes |
| G1: | OBL(0.6) | QOBL(0.6) | Yes |
| $f_4$ | QOBL(0.3) | OBL(0.6) | Yes |
| $f_5$ | - | - | No |
| $f_6$ | OBL(0.3) | QOBL(0.6∼0) | Yes |
| $f_7$ | QOBL(0.3) | DECC | Yes |
| G2: | QOBL(0.3) | OBL(0∼0.6) | Yes |
| $f_8$ | QOBL(0.3) | OBL(0.6) | Yes |
| $f_9$ | OBL(0.6) | OBL(0∼0.6) | Yes |
| $f_{10}$ | - | - | No |
| $f_{11}$ | QOBL(0.3) | OBL(0∼0.6) | Yes |
| G3: | QOBL(0.3) | OBL(0∼0.6) | Yes |
| $f_{12}$ | OBL(0∼0.6) | QOBL(0.6) | Yes |
| $f_{13}$ | QOBL(0.3) | OBL(0∼0.6) | Yes |
| $f_{14}$ | QOBL(0.3) | OBL(0∼0.6) | Yes |
| G4: | QOBL(0.3) | OBL(0.6) | Yes |
| $f_{15}$ | QOBL(0.6) | OBL(0∼0.6) | Yes |
| G5: | QOBL(0.6) | OBL(0∼0.6) | Yes |
| sum | QOBL(0∼0.6) | DECC | Yes |

from hybridization is not statistically significant. The last column of Table IV indicates that on 12 out of 15 functions (i.e. all except for $f_1$, $f_5$ and $f_{10}$) the hybridization significantly improves DECC performance. These results once again show the effectiveness of the proposed framework.

## V. FUTURE WORKS

Considering the generality of the proposed hybridization of OBL and CC, every opposition strategy and decomposition technique can be hybridized. Furthermore, the proposed framework allows us to employ different evolutionary algorithms as the core optimizer. Therefore, we are interested in comparing different opposition strategies, decomposition techniques and core optimizers to investigate their effects on the performance of the resulting hybridization. Additionally, as discussed in Section IV-C, the key control parameters, such as $J_r$, affect the performance of the hybrid algorithm. Consequently, sensitivity analysis of the $J_r$ parameter is also the subject of our future investigations. Finally, comparing the performance of this framework (using more advanced opposition strategies and decomposition techniques) with the state-of-the-art methods for tackling large-scale optimization problems is of interest.

## VI. CONCLUSION

In this study, a novel framework for hybridizing oppositional-based learning (OBL) and cooperative co-evolution (CC) for dealing with large-scale optimization problems is proposed. While CC's decomposition strategy breaks the search space into several smaller subspaces, the OBL operators increases the chance of finding the global optimum. The proposed framework has been successfully applied to the most recent large-scale benchmark functions and the results are compared with a baseline method called DECC. The obtained results clearly show that the hybridization significantly improves its parents' performance. Several statistical analysis are also presented in this study all of which confirm the statistical significance of the results.

According to the generality of the proposed framework, two different opposition strategies are studied in this paper. The obtained results indicate that some components (e.g., OBL) are effective in dealing with fully separable functions while the others (e.g. QOBL) are more promising for solving functions with some degrees of non-separability. Further studies may shed light on the usability of each component and help users in choosing the most effective OBL and CC strategies that suit their needs.

The provided results show that the main control parameters can affect the power of the proposed framework. Hence, a deep and comprehensive parameter sensitivity analysis is required to investigate the potential effects of each parameter and provide some rules of thumb for practitioners to easily chose effective parameter values.

## ACKNOWLEDGEMENT

This work was supported by NSFC under Grant No. 61005051, SRFDP under Grant No. 20100092120027 and ARC Discovery Grant (DP120102205).

## REFERENCES

[1] T. Weise, R. Chiong, and K. Tang, "Evolutionary Optimization: Pitfalls and Booby Traps," *Journal of Computer Science and Technology (JCST)*, vol. 27, no. 5, pp. 907–936, 2012, special Issue on Evolutionary Computation.

[2] K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, , and Z. Yang, "Benchmark functions for the CEC'2008 special session and competition on large scale global optimization," Nature Inspired Computation and Applications Laboratory, USTC, China, Tech. Rep., 2007, http://nical.ustc.edu.cn/cec08ss.php.

[3] 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, http://nical.ustc.edu.cn/cec10ss.php.

[4] 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, Melbourne, Australia, Tech. Rep., 2013, http://goanna.cs.rmit.edu.au/ xiaodong/cec13-lsgo.

[5] Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Information Sciences*, vol. 178, pp. 2986–2999, August 2008.

[6] M. N. Omidvar, X. Li, Y. Mei, and X. Yao, "Cooperative co-evolution with differential grouping for large scale optimization," *Evolutionary Computation, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2013.

[7] Z. Yang, K. Tang, and X. Yao, "Multilevel cooperative coevolution for large scale optimization," in *Proc. of IEEE Congress on Evolutionary Computation*, June 2008, pp. 1663–1670.

[8] D. Molina, M. Lozano, and F. Herrera, "MA-SW-Chains: Memetic algorithm based on local search chains for large scale continuous global optimization," in *Proc. of IEEE Congress on Evolutionary Computation*, july 2010, pp. 3153–3160.

[9] W. Chen, T. Weise, Z. Yang, and K. Tang, "Large-scale global optimization using cooperative coevolution with variable interaction learning," in *Proc. of International Conference on Parallel Problem Solving from Nature*, ser. Lecture Notes in Computer Science, vol. 6239. Springer Berlin / Heidelberg, 2011, pp. 300–309.

[10] A. Zamuda, J. Brest, B. Boskovic, and V. Zumer, "Large scale global optimization using differential evolution with self-adaptation and cooperative co-evolution," in *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, 2008, pp. 3718–3725.

[11] X. Li and X. Yao, "Cooperatively coevolving particle swarms for large scale optimization," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 2, pp. 210–224, April 2012.

[12] 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.

[13] B. Kazimipour, B. Salehi, and M. Z. Jahromi, "A novel genetic-based instance selection method: Using a divide and conquer approach," in *Artificial Intelligence and Signal Processing (AISP), 2012 16th CSI International Symposium on*. IEEE, 2012, pp. 397–402.

[14] M. N. Omidvar, X. Li, and X. Yao, "Cooperative co-evolution with delta grouping for large scale non-separable function optimization," in *Proc. of IEEE Congress on Evolutionary Computation*, 2010, pp. 1762–1769.

[15] S. Rahnamayan and G. G. Wang, "Solving large scale optimization problems by opposition-based differential evolution (ode)," *WSEAS Transactions on Computers*, vol. 7, no. 10, pp. 1792–1804, 2008.

[16] S. Rahnamayan and G. G. Wang, "Investigating in scalability of opposition-based differential evolution," *WSEAS Trans Comput*, vol. 7, pp. 1792–1804, 2008.

[17] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[18] S. Rahnamayan, H. R. Tizhoosh, and M. M. Salama, "Opposition-based differential evolution for optimization of noisy problems," in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*. IEEE, 2006, pp. 1865–1872.

[19] Q. Xu, L. Wang, N. Wang, X. Hei, and L. Zhao, "A review of opposition-based learning from 2005 to 2012," *Engineering Applications of Artificial Intelligence*, 2014.

[20] S. Rahnamayan, H. R. Tizhoosh, and M. Salama, "Opposition versus randomness in soft computing techniques," *Applied Soft Computing*, vol. 8, no. 2, pp. 906–918, 2008.

[21] 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.

[22] 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.

[23] Y. Davidor, "Epistasis Variance: Suitability of a Representation to Genetic Algorithms," *Complex Systems*, vol. 4, no. 4, pp. 369–383, 1990.

[24] M. N. Omidvar, X. Li, Z. Yang, and X. Yao, "Cooperative co-evolution for large scale optimization through more frequent random grouping," in *Proc. of IEEE Congress on Evolutionary Computation*, 2010, pp. 1754–1761.

[25] H. R. Tizhoosh, "Opposition-based learning: a new scheme for machine intelligence," in *Computational intelligence for modelling, control and automation, 2005 and international conference on intelligent agents, web technologies and internet commerce, international conference on*, vol. 1. IEEE, 2005, pp. 695–701.

[26] S. Rahnamayan, H. R. Tizhoosh, and M. M. Salama, "Quasi-oppositional differential evolution," in *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*. IEEE, 2007, pp. 2229–2236.

[27] M. Ergezer, D. Simon, and D. Du, "Oppositional biogeography-based optimization," in *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*. IEEE, 2009, pp. 1009–1014.

[28] H. Wang, Z. Wu, J. Wang, X. Dong, S. Yu, and C. Chen, "A new population initialization method based on space transformation search," in *Natural Computation, 2009. ICNC'09. Fifth International Conference on*, vol. 5. IEEE, 2009, pp. 332–336.

[29] H. Wang, Z. Wu, Y. Liu, J. Wang, D. Jiang, and L. Chen, "Space transformation search: a new evolutionary technique," in *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*. ACM, 2009, pp. 537–544.

[30] Q. Xu, L. Wang, B. He, and N. Wang, "Modified opposition-based differential evolution for function optimization," *Journal of Computational Information Systems*, vol. 7, no. 5, pp. 1582–1591, 2011.

[31] B. Kazimipour, X. Li, and A. Qin, "Initialization methods for large scale global optimization," in *Evolutionary Computation (CEC), 2013 IEEE Congress on*. IEEE, 2013, pp. 2750–2757.

[32] B. Kazimipour, X. Li, and A. Qin, "A review of population initialization techniques for evolutionary algorithms," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2014.

[33] B. Kazimipour, X. Li, and A. Qin, "Effects of population initialization in differential evolution for large scale optimization," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2014.

[34] S. Rahnamayan, H. R. Tizhoosh, and M. M. Salama, "Opposition-based differential evolution (ode) with variable jumping rate," in *Foundations of Computational Intelligence, 2007. FOCI 2007. IEEE Symposium on*. IEEE, 2007, pp. 81–88.

[35] A. Qin and X. Li, "Differential evolution on the cec-2013 single-objective continuous optimization testbed," in *Evolutionary Computation (CEC), 2013 IEEE Congress on*. IEEE, 2013, pp. 1099–1106.

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

[37] S. García, D. Molina, M. Lozano, and F. Herrera, "A study on the use of non-parametric tests for analyzing the evolutionary algorithms behaviour: a case study on the cec2005 special session on real parameter optimization," *Journal of Heuristics*, vol. 15, no. 6, pp. 617–644, 2009.

[38] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced non-parametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," *Information Sciences*, vol. 180, no. 10, pp. 2044–2064, 2010.

[39] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, 2011.

[40] J. Li, "A two-step rejection procedure for testing multiple hypotheses," *Journal of Statistical Planning and Inference*, vol. 138, no. 6, pp. 1521–1527, 2008.