Cooperative Co-evolution with Online Optimizer Selection for Large-Scale Optimization

Yuan Sun
Computing and Information Systems
The University of Melbourne
Parkville, VIC, Australia
yuan.sun@unimelb.edu.au

Michael Kirley
Computing and Information Systems
The University of Melbourne
Parkville, VIC, Australia
mkirley@unimelb.edu.au

Xiaodong Li
School of Science
RMIT University
Melbourne, VIC, Australia
xiaodong.li@rmit.edu.au

ABSTRACT

Cooperative co-evolution (CC) is an effective framework that can be used to solve large-scale optimization problems. It typically divides a problem into components and uses one optimizer to solve the components in a round-robin fashion. However the relative contribution of each component to the overall fitness value may vary. Furthermore, using one optimizer may not be sufficient when solving a wide range of components with different characteristics. In this paper, we propose a novel CC framework which can select an appropriate optimizer to solve a component based on its contribution to the fitness improvement. In each evolutionary cycle, the candidate optimizer and component that make the greatest contribution to the fitness improvement are selected for evolving. We evaluated the efficacy of the proposed CC with Optimizer Selection (CCOS) algorithm using large-scale benchmark problems. The numerical experiments showed that CCOS outperformed the CC model without optimizer selection ability. When compared against several other state-of-the-art algorithms, CCOS generated competitive solution quality.

CCS CONCEPTS

• **Theory of computation → Online learning algorithms**; **Divide and conquer**;

KEYWORDS

Large-scale optimization, cooperarive co-evolution, algorithm selection, algorithm hybridization, resources allocation

ACM Reference Format:
Yuan Sun, Michael Kirley, and Xiaodong Li. 2018. Cooperative Co-evolution with Online Optimizer Selection for Large-Scale Optimization. In *GECCO '18: Genetic and Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205455.3205625

1 INTRODUCTION

Large-scale global optimization (LSGO) problems are challenging for Evolutionary Algorithms (EAs) to solve. This may be partially attributed to the fact that the search space and the complexity of an optimization problem grow quickly as the dimensionality increases [19, 31]. Cooperative Co-evolution (CC) [22] has been used with some success when scaling up EAs to tackle very high dimensional search and optimization problems [7, 14, 18, 23, 33]. The CC framework divides a LSGO problem into a number of components that are solved cooperatively. When optimizing each component, representatives (typically the best sub-solutions found) from each of the other components are combined with individuals in the optimized component, to form complete candidate solutions that can be evaluated.

The original CC framework employs only one EA to solve the components in a round-robin fashion: the components are optimized sequentially with an equal computational budget [22]. However, the relative contribution of each component to the overall fitness value may vary [19]. To address this issue, the contribution based CC framework is proposed to efficiently allocate computational resources between components [20, 32].

Another limitation of the original CC framework is that it may be insufficient when attempting to solve a wide range of components using only one optimizer. Algorithm selection and hybridization techniques have been suggested as attractive approaches to balance exploration and exploitation, capable of generating promising results [1, 9, 17]. However, such mechanisms have not been thoroughly investigated within the context of CC.

In this paper, we propose a novel CC framework, which can select an appropriate optimizer from a portfolio to optimize a component. In the first evolutionary cycle, the candidate optimizers are sequentially used to optimize the components. The fitness improvement by each optimizer when solving each component is calculated. In the remaining cycles, the optimizer and component pair that makes the greatest historical contribution to the fitness improvement is selected, and only the selected optimizer is used to solve the corresponding component.

We have evaluated the efficacy of the proposed CC with optimizer selection (CCOS) using the benchmark problems from the special session on LSGO at CEC'2010 [28]. Comprehensive numerical simulations showed that CCOS can successfully select an appropriate optimizer to solve the benchmark problems. Significantly, CCOS outperformed the base-line CC algorithm without optimizer selection ability in most cases. When compared against several other state-of-the-arts, CCOS achieved statistically comparable or significantly better solution quality.

The remainder of this paper is organized as follows. In Section 2, we introduce the state-of-the-art algorithms that can be used to solve LSGO problems. In Section 3, the proposed CCOS is described

GECCO '18, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5618-3/18/07…$15.00
https://doi.org/10.1145/3205455.3205625

in detail, and the additional computational complexity induced by the online optimizer selection process is analyzed. Section 4 describes experiments to evaluate the proposed CCOS algorithm. The following section presents and analyzes the experimental results. Conclusion is drawn in the last section.

2 RELATED WORK

In this section, we briefly describe the state-of-the-art techniques that can be used to 'scale up' EAs for solving LSGO problems.

2.1 Cooperative Co-evolution

The CC framework tackles a LSGO problem using a divide-and-conquer strategy. It divides the original problem into a number of low-dimensional components that are solved cooperatively. A standard CC algorithm consists of two stages: *decomposition* and *optimization*.

In the decomposition stage, a LSGO problem is decomposed into several components. A number of decomposition methods have been proposed in recent years, e.g., random grouping [33], cooperative co-evolution with variable interaction learning [3], differential grouping (DG) [18], extended DG [25], global DG [15], fast interdependency identification [8], DG2 [21] and recursive DG (RDG) [26], to name a few.

In the optimization stage, an EA is typically used to optimize the components based on a context vector. The context vector is a complete candidate solution, typically consisting of the best sub-solutions from each component. When optimizing the i^{th} component, the context vector (excluding the i^{th} sub-solution) is used to combine with the individuals in the i^{th} component, to form complete candidate solutions that can be evaluated.

The original CC framework [22] optimizes the components in a round-robin fashion. That is the computational resources are equally distributed to each component. However in problems with imbalanced components [19], the weight of each component to the overall fitness value may be very different. To efficiently solve such problem, the contribution based CC (CBCC) is proposed, which allocates computational resources to components based on their values of fitness improvement [20]. In each cycle, it selects and optimizes the component that makes the greatest accumulated contribution to the fitness improvement. Then the contribution of the selected component c_i is updated as follows:

$$U_i = \hat{U}_i + \hat{y}_b - y_b, \tag{1}$$

where $\hat{U}_i$ is the previous accumulated contribution; $\hat{y}_b$ and y_b are the best fitness values found before and after c_i was optimized. Without loss of generality, we assume minimization problems in this paper. The fitness improvement in a minimization problem refers to the reduction of objective value or cost.

The CBCC framework could respond too slowly to the recent fitness improvement. A component which is stagnant (i.e., the overall fitness cannot be further improved by solving this component) may still be selected and evolved for many cycles if its previous contribution was very large. To address this issue, a more effective contribution accumulation criterion is proposed [32]:

$$U_i = (\hat{U}_i + \hat{y}_b - y_b)/2. \tag{2}$$

This criterion gives more weight to the latest fitness improvement. The contribution in the early cycles becomes less important as the evolutionary process progresses. Therefore, such criterion can respond faster to a more recent fitness improvement.

The criterion Eq. (2) works well when the identified components are additively separable, e.g., $f(\mathbf{x}) := f_1(\mathbf{x}_1) + f_2(\mathbf{x}_2)$, where $\mathbf{x}_1$ and $\mathbf{x}_2$ are the decision sub-vectors in components c_1 and c_2 respectively. The absolute value of the fitness improvement in component c_2 is independent with component c_1, and vice verse:

$$\hat{y}_b - y_b = \big(f_1(\mathbf{x}_1) + f_2(\hat{\mathbf{x}}_2)\big) - \big(f_1(\mathbf{x}_1) + f_2(\mathbf{x}_2)\big) = f_2(\hat{\mathbf{x}}_2) - f_2(\mathbf{x}_2), \tag{3}$$

where $\hat{\mathbf{x}}_2$ and $\mathbf{x}_2$ are the best sub-solutions found before and after c_2 is optimized. However, if c_1 and c_2 are not additively separable, e.g., $f(\mathbf{x}) := f_1(\mathbf{x}_1)f_2(\mathbf{x}_2)$, the absolute value of the fitness improvement in component c_2 is influenced by the fitness of component c_1:

$$\hat{y}_b - y_b = f_1(\mathbf{x}_1)f_2(\hat{\mathbf{x}}_2) - f_1(\mathbf{x}_1)f_2(\mathbf{x}_2) = f_1(\mathbf{x}_1)\big(f_2(\hat{\mathbf{x}}_2) - f_2(\mathbf{x}_2)\big). \tag{4}$$

The original CC uses only one EA to solve all the components. A memetic CC is proposed to enhance the search of an EA with local search methods [27]. In each cycle, after applying an EA, different local search methods were used to further solve separable and non-separable components. In Section 3, we will propose a more general optimizer selection / hybridization framework.

2.2 Techniques Other Than Cooperative Co-evolution

In addition to CC, there are other techniques that can be used to address the additional challenges inherent in LSGO problems. Representative techniques include the model complexity control [6] and random projection [10] methods for estimation of distribution algorithms; the multiple strategies [2] and generalized opposition-based learning [30] methods for differential evolution; the social learning [5] and pairwise competition [4] methods for particle swarm optimization; and the multiple trajectory search [29] as well as multiple offspring sampling [11] methods for algorithm hybridization. Due to page limits, we cannot describe these techniques in detail.

3 COOPERATIVE CO-EVOLUTION WITH COMPONENT OPTIMIZER SELECTION

In this section, we describe a novel CC framework that can select an appropriate optimizer to solve a component in the evolutionary process.

The proposed CC framework consists of two stages: a decomposition stage and an enhanced optimization stage. In the decomposition stage, any decomposition method can be used to decompose a LSGO problem into several components. In the optimization stage, the candidate optimizer that contributes the most to the fitness improvement is selected to solve a component.

In the first cycle, the candidate optimizers ($\mathbb{A}$) are used to solve the components ($\mathbb{C}$) sequentially. Each candidate optimizer is assigned with equal computational budget to solve each component. The contribution of optimizer a_i ($1 \le i \le |\mathbb{A}|$) when solving component c_j ($1 \le j \le |\mathbb{C}|$) is calculated as the relative fitness improvement [13, 35]:

$$I_{a_i,c_j} = \frac{\hat{y}_b - y_b}{\hat{y}_b}, \tag{5}$$

where $\hat{y}_b$ and y_b are the best fitness values found before and after applying optimizer a_i on component c_j in one cycle. The relative fitness improvement illustrates the strength and weakness of the candidate optimizers when solving the components.

The overall contribution of optimizer a_i when solving component c_j is accumulated as follows:

$$U_{a_i,c_j} = \frac{\hat{U}_{a_i,c_j} + I_{a_i,c_j}}{2}, \tag{6}$$

where $\hat{U}_{a_i,c_j}$ is the latest accumulated contribution of optimizer a_i on component c_j before this cycle starts. The value of $\hat{U}_{a_i,c_j}$ is initialized as 0. The accumulated contribution U_{a_i,c_j} takes into account all relative fitness improvements in the previous cycles. The weight of contributions in the past decays exponentially as the evolutionary process progresses.

In the next cycle, the optimizer and component pair that has the maximum value of accumulated contribution (maximum of U) is selected:

$$< a_i, c_j >= \arg\max_{a_i,c_j} \left\{ U_{a_i,c_j} \middle| 1 \le i \le |\mathbb{A}|, 1 \le j \le |\mathbb{C}| \right\}. \tag{7}$$

Only the selected optimizer a_i is used to solve component c_j in this cycle. In other words, the proposed CC framework only allocates computational resources to the pair of optimizer and component that previously contributes the most to overall fitness improvement. The fitness improvement I_{a_i,c_j} is calculated and the accumulated contribution U_{a_i,c_j} is updated as before. This process continues until the computational budget is exhausted.

Our contribution accumulating criterion differs from Eq. (2) in that our approach considers the relative fitness improvement [13, 35], while Eq. (2) is based on the absolute value of the fitness improvement. We believe that the relative fitness improvement is more reliable in our model as it removes the bias of the order that optimizers are used to solve the components. An optimizer which is used first to solve a component is more likely to have a large value of the absolute fitness improvement, as the initial fitness value is large.

In the following, we analyze the additional computational cost imposed by the online optimizer selection procedure. An naïve approach to implement the optimizer selection model is to store the values of U in an array. In each evolutionary cycle, a linear scan is performed in order to find the maximum value of the array. Therefore the total computational complexity is $\Theta(|\mathbb{A}||\mathbb{C}|T)$, where T is the maximum cycle.

To improve the efficiency of selection, we can use the max-heap data structure. The nodes of the max-heap are the tuples $< a_i, c_j, U_{a_i,c_j} >$, where $1 \le i \le |\mathbb{A}|$, and $1 \le j \le |\mathbb{C}|$. Therefore, the number of nodes in the max-heap is $|\mathbb{A}||\mathbb{C}|$, and the height of the max-heap is $\log_2(|\mathbb{A}||\mathbb{C}|)$. Initially, the nodes are sequentially placed into a complete binary tree. After the first cycle, the bottom-up algorithm [12] can be used to turn the complete binary tree into a max-heap with respect to the contribution values U, which takes $\Theta(|\mathbb{A}||\mathbb{C}|)$ time. In each of the following cycles, the root of the max-heap is selected as it has the maximum U value. The selected optimizer is then used to solve the component, and the U value of the root is updated. To re-heapify the complete binary tree, we can let the root "sift down" to the bottom, which takes $O\big(\log(|\mathbb{A}||\mathbb{C}|)\big)$ time [12]. Therefore, the total time complexity is $O\big(|\mathbb{A}||\mathbb{C}| + (T-1)\log(|\mathbb{A}||\mathbb{C}|)\big)$. As T is much larger than $|\mathbb{A}|$ or $|\mathbb{C}|$, the total time complexity can be written as $O\big(T\log(|\mathbb{A}||\mathbb{C}|)\big)$.

Algorithm 1 CC with optimizer selection

```
Decompose the LSGO problem into components ℂ.
Initialize the set of candidate optimizers 𝔸.
Initialize the contribution of each component optimizer pair: U_{a_i,c_j} = 0.
Initialize the evolutionary cycle t = 1.
while t ≤ T do
    if t = 1 then
        for i from 1 to |𝔸| do
            for j from 1 to |ℂ| do
                Solve component c_j using optimizer a_i.
                Calculate contribution: I_{a_i,c_j} = (ŷ_b − y_b)/ŷ_b.
                Accumulate contribution: U_{a_i,c_j} = (Û_{a_i,c_j} + I_{a_i,c_j})/2.
            end for
        end for
    else
        < a_i, c_j >= arg max_{a_i,c_j} {U_{a_i,c_j} | 1 ≤ i ≤ |𝔸|, 1 ≤ j ≤ |ℂ|}.
        Solve component c_j using optimizer a_i.
        Calculate contribution: I_{a_i,c_j} = (ŷ_b − y_b)/ŷ_b.
        Accumulate contribution: U_{a_i,c_j} = (Û_{a_i,c_j} + I_{a_i,c_j})/2.
    end if
    Update the evolutionary cycle: t = t + 1.
end while
return the best fitness value found y_b.
```

4 EXPERIMENTAL METHODOLOGY

In this section, numerical experiments are designed to evaluate the efficacy of the proposed CC framework with online optimizer selection.

In the decomposition stage, the RDG [26] method was used to decompose a LSGO problem into components. The RDG method identifies the interaction between two subsets of decision variables based on non-linearity detection. A recursive procedure is adopted to improve decomposition efficiency.

In the optimization stage, the self-adaptive differential evolution with neighborhood search (SaNSDE) [34] and social-learning particle swarm optimization (SL-PSO) [5] were used as the candidate optimizers to solve components. The SaNSDE is a widely used optimizer in the CC domain [18, 21, 25, 26, 33]. The SL-PSO is a competitive optimizer for solving large-scale optimization problems, however it has not been tested within the context of CC. We term our model as CC with optimizer selection (CCOS). In the following, if not specified, we use DE to represent SaNSDE, and PSO to represent SL-PSO for the sake of simplicity.

The proposed CCOS algorithm was used to solve the CEC'2010 LSGO benchmark problems [28].[1] The maximum number of function evaluations (FEs) was set to 10^6. The FEs in each cycle was set to 10^4. The population size for DE and PSO was set to 100. It is noteworthy that the DE and PSO shared the same population in the evolutionary process. This allowed the two optimizers to exchange information and to collaboratively solve the components.

[1] The MATLAB implementation of the CCOS algorithm is available from the following link: https://bitbucket.org/yuans/ccos.

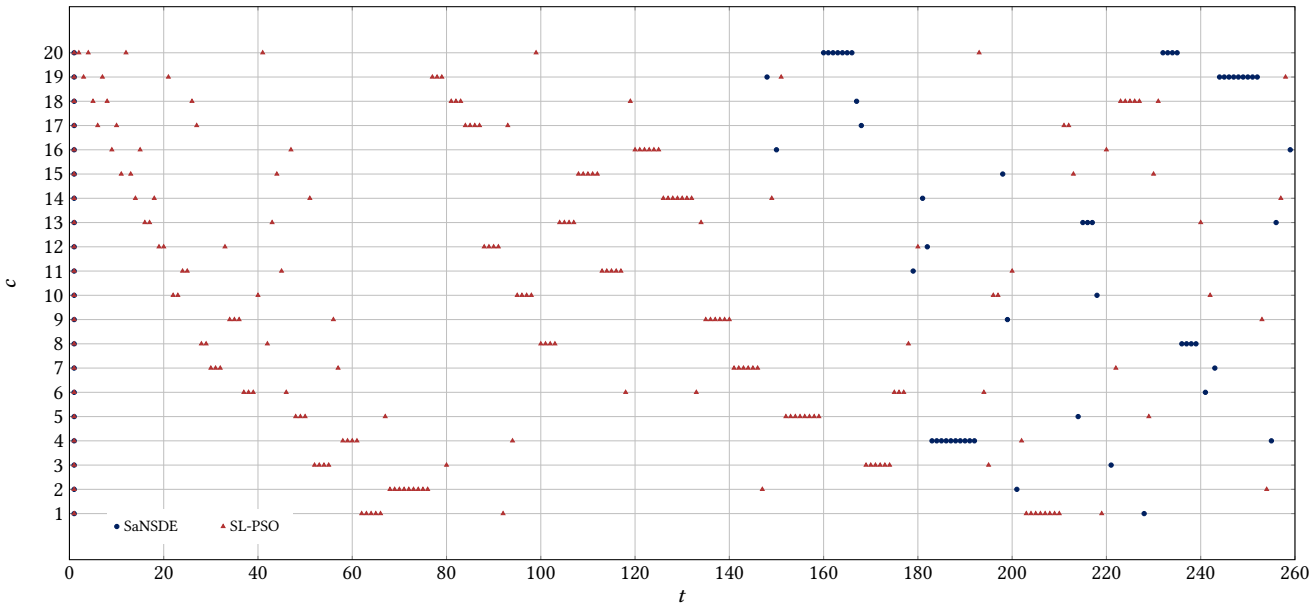

Figure 1: The components and optimizers selected by the CCOS algorithm in the evolutionary process when used to solve the CEC'2010 f_{16} benchmark problem. The horizontal axis (t) represents the number of the evolutionary cycles, while the vertical axis (c) represents the index of the components.

The initial population was uniformly randomly generated across the search space. Each individual of the population had two suites of parameters: one for DE and the other for PSO. The parameter settings for DE and PSO were consistent with the original papers. When DE (PSO) was selected to update the population, the parameter values for PSO (DE) of the offsprings were copied from their parents. For each benchmark problem, the median, mean and standard deviation of the best solutions found by the CCOS algorithm based on 25 independent runs were recorded.

To show the efficacy of the proposed CCOS algorithm, we compared the performance of CCOS against the performances of CCDE and CCPSO. The CCDE (CCPSO) differs from CCOS in that the former uses only DE (PSO) to solve the components, while the latter uses both DE and PSO as the component optimizers. The performance of the CCOS algorithm was also compared to the performances of several other state-of-the-art algorithms – competitive swarm optimizer (CSO) [4] MOS [11] and MA-SW-Chains [16] with default parameter settings. The MOS algorithm achieved the best performance in the 2011 special issue of the Soft Computing journal, and the MA-SW-Chains algorithm achieved the best performance in the CEC 2010 special session and competition on LSGO. The Wilcoxon rank-sum test [24] with 95% confidence interval (significance level $\alpha = 0.05$) was conducted to identify if the CCOS algorithm generated statistically significantly better solution quality than the other algorithm.

5 EXPERIMENTAL RESULTS

In Section 5.1, we present the details of the components and optimizers selected by the CCOS algorithm in the evolutionary process. In Section 5.2, we analyze the accuracy of CCOS in terms of selecting an appropriate optimizer. Section 5.3 presents the comparison between CCOS against the CC without optimizer selection ability: CCDE and CCPSO. Section 5.4 presents the comparison between CCOS with several other state-of-the-arts.

5.1 Selection Details of CCOS

To track the optimizer selection process of the proposed CCOS algorithm, we selected CEC'2010 f_{16} benchmark problem. The f_{16} is a partially separable problem with 20 components, each with 50 non-separable decision variables. The details of the components and optimizers selected by CCOS in the evolutionary process were shown in Figure 1. In the first cycle, both of the DE and PSO were used to solve each of the 20 components, and the contribution of each optimizer component pair was calculated. In the remaining cycles, the optimizer component pair that historically contributed the most to the fitness improvement was selected and evolved. We observed that in the first 160 cycles, PSO was most frequently selected to solve the components, while in the last 100 cycles, DE was selected more often.

5.2 Selection Ability of CCOS

To investigate whether the proposed CCOS algorithm can select the best optimizer to solve the benchmark problems, we calculated the ratio of the evolutionary cycles that DE and PSO were selected by CCOS:

$$\rho_t = \frac{t_{\mathrm{DE}}}{t_{\mathrm{PSO}}}, \tag{8}$$

where t_{DE} and t_{PSO} denote the number of cycles that DE and PSO were selected in the evolutionary process respectively. The values

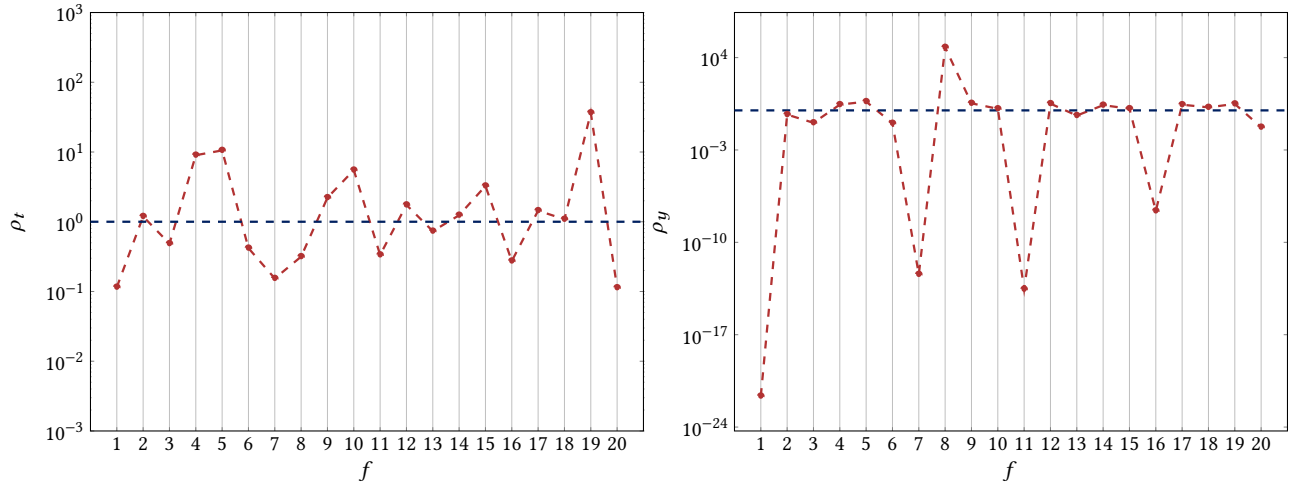

Figure 2: The ratio of the number of evolutionary cycles (ρ_t) that DE and PSO were selected by the CCOS algorithm (left figure), as well as the ratio of the solution quality (ρ_y) generated by CCPSO and CCDE (right figure) when used to solve the CEC'2010 benchmark problems. The horizontal axis (f) represents the index of the benchmark problems. The ratio was calculated using the run that generated the median best solution from 25 independent runs. In each figure, the horizontal dashed line represents the base line $\rho = 1$.

of the ratio ρ_t for the CEC'2010 benchmark problems were plotted in Figure 2. If $\rho_t < 1$, it meant the PSO optimizer was selected more often than DE when used to solve the given problem.

We also plotted in Figure 2 the ratio of the solution quality found by the CCPSO and CCDE algorithms:

$$\rho_y = \frac{y_{\text{PSO}}}{y_{\text{DE}}}, \tag{9}$$

where y_{PSO} and y_{DE} denote the median of the best fitness values found by CCPSO and CCDE based on 25 independent runs. If $\rho_y < 1$, it meant the best solution quality found by CCPSO was smaller than that found by CCDE, therefore the corresponding optimizer PSO was better than DE when used to solve the given problem.

We observed that the pattern of the ratio ρ_y was generally consistent with the pattern of ρ_t. On the benchmark problems where CCPSO generated better solution quality than CCDE ($\rho_y < 1$), the corresponding optimizer PSO was selected more often ($\rho_t < 1$) by the CCOS algorithm in the evolutionary process except for the problems f_2 and f_8.

The problem f_2 is fully separable. We observed that both of the ρ_y and ρ_t were close to 1. The median of the best solution quality generated by CCDE was 3.12e+03, which was close to that generated by CCPSO 1.52e+03. The number of the evolutionary cycles that DE and PSO was selected by CCOS were 166 and 134 respectively. The quality of the median best solution generated by CCOS was in between of those generated by CCDE and CCPSO.

The problem f_8 consists of a component of 950 separable decision variables and a component of 50 non-separable decision variables. The median of the best solution quality generated by CCDE was better than that generated by CCPSO. However the number of cycles that DE was selected by CCOS was less than that of PSO. Surprisingly, the CCOS algorithm generated significantly better solution quality than both of the CCDE and CCPSO algorithms (see the convergence curves in Figure 3).

We plotted the details of the components and/or optimizers selected by the CCOS, CCDE and CCPSO algorithms in the evolutionary process in Figure 3. We observed that the non-separable component (c_2) contributed more to the fitness improvement, as c_2 was selected first to be optimized by all the algorithms. Further, DE was more efficient when solving the non-separable component c_2, while PSO was more efficient to solve the separable component c_1. This can be inferred from the selection details of CCOS: DE was selected more often to solve c_2, while only PSO was used to solve c_1. The CCPSO used all the cycles to optimize c_2. However it was not efficient to solve c_2, as indicated by the slow convergence rate. The CCOS algorithm converged equally fast as CCDE in the first 80 cycles when optimizing c_2. However after the first 100 cycles, CCOS converged much faster than CCDE when PSO was used to solve c_1.

5.3 Comparison with CCDE and CCPSO

The box plots of the best fitness values generated by the CCOS, CCDE and CCPSO algorithms when used to solve the CEC'2010 benchmark problems based on 25 independent runs were presented in Figure 4. The results showed that the CCOS algorithm consistently generated statistically significantly better solution quality than at least one of the CCDE and CCPSO algorithms. The number of "win/tie/loss" of CCOS when compared against CCDE and CCPSO based on the Wilcoxon rank-sum test (significance level $\alpha = 0.05$) was 14/2/4 and 12/5/3 respectively. Significantly, the CCOS algorithm achieved statistically significantly better solution quality than both the CCDE and CCPSO algorithms on 7 out of 20 benchmark problems. The results implied that CCOS could potentially select a well-performed optimizer for solving each component.

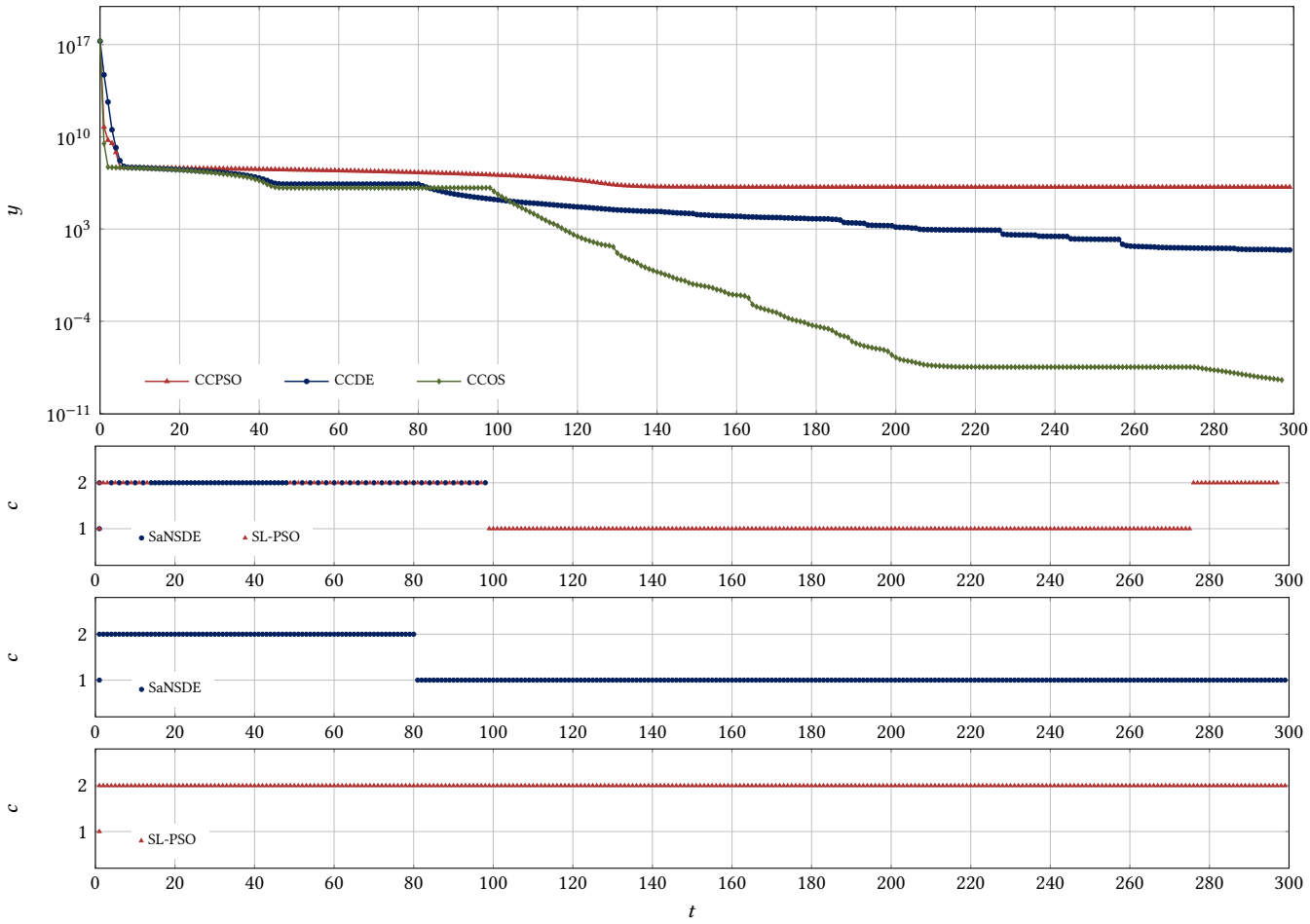

Figure 3: The convergence curves (top figure) and the details of the components and/or optimizers selected by the CCOS (second figure), CCDE (third figure) and CCPSO (bottom figure) in each evolutionary cycle when used to solve the problem f_8. The horizontal axis (t) represents the number of the evolutionary cycles. In the top figure, the vertical axis y represents the fitness value of the median best solution found by each algorithm from 25 independent runs. In the remaining figures, the vertical axis c represents the index of the components.

5.4 Comparison with State-of-the-arts

The optimization results of the CCOS, CSO, MOS and MA-SW-Chain algorithms when used to solve the CEC'2010 benchmark problems were presented in Table 1. The CCOS algorithm achieved statistically equal or significantly better solution quality than CSO in 16 out of 20 benchmark problems based on the Wilcoxon rank-sum test (significance level $\alpha = 0.05$). On the fully separable (f_1 to f_3) and fully non-separable (f_{19} and f_{20}) benchmark problems, the CCOS algorithm was outperformed by MOS and MA-SW-Chain. It was noteworthy that the CCOS algorithm (with RDG as the decomposition method) did not perform any actual decomposition for these problems. However on the partially separable problems (f_4 to f_{18}), the CCOS algorithm generally produced comparable or statistically significantly better solution quality than MOS and MA-SW-Chain. The experimental results confirmed the effectiveness of CC as a divide-and-conquer approach.

6 CONCLUSION

In this paper, we have investigated how the use of alternative optimizers at different evolutionary stages impacted on the solution quality generated by the CC when used to solve LSGO problems. Instead of employing only one optimizer to solve all the components, we proposed an online optimizer selection framework to select the best optimizer from a portfolio for each component. At each evolutionary cycle, the component and optimizer pair that previously contributed the most to the overall fitness improvement was selected for evolving. We experimentally demonstrated that the proposed CCOS algorithm was successful in selecting the best optimizer when solving the CEC'2010 benchmark problems. Significantly, CCOS could potentially generate statistically better solution quality than the default CC algorithm with no optimizer selection ability. When compared against several other state-of-the-art algorithms, CCOS also achieved competitive results.

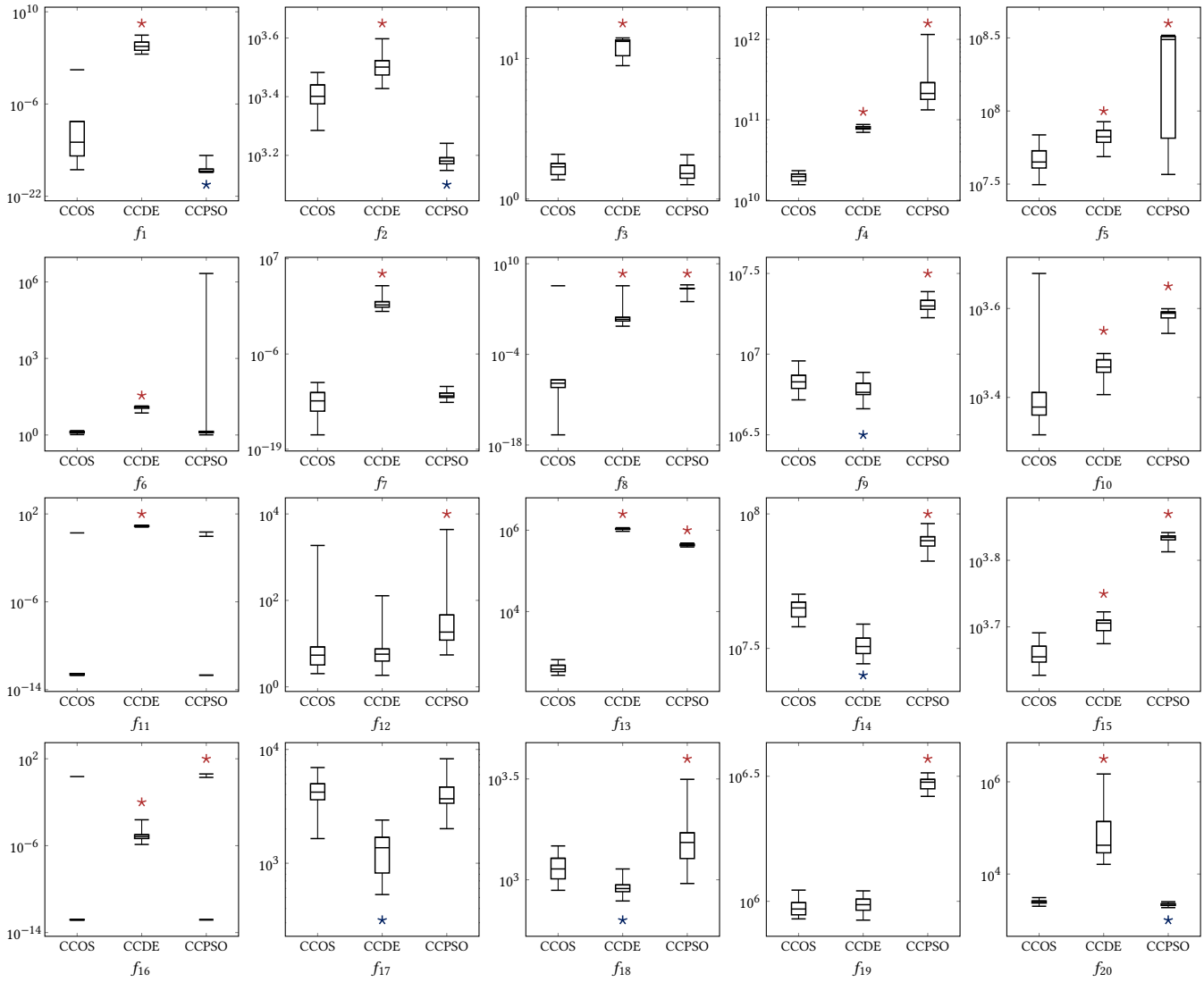

Figure 4: The box plots of the best fitness values generated by CCOS, CCDE and CCPSO when used to solve the CEC'2010 benchmark problems based on 25 independent runs. The notation "⋆" above/under the box plot represents CCOS performed statistically better/worse than the corresponding algorithm based on the Wilcoxon rank-sum test (significance level $\alpha = 0.05$). No "⋆" means CCOS performed equally well with the corresponding algorithm.

REFERENCES

[1] Christian Blum and Andrea Roli. 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)* 35, 3 (2003), 268–308.

[2] Janez Brest and Mirjam Sepesy Maučec. 2011. Self-adaptive differential evolution algorithm using population size reduction and three strategies. *Soft Computing* 15, 11 (2011), 2157–2174.

[3] Wenxiang Chen, Thomas Weise, Zhenyu Yang, and Ke Tang. 2010. Large-scale global optimization using cooperative coevolution with variable interaction learning. In *Parallel Problem Solving from Nature, PPSN XI*. Springer, 300–309.

[4] Ran Cheng and Yaochu Jin. 2015. A competitive swarm optimizer for large scale optimization. *Cybernetics, IEEE Transactions on* 45, 2 (2015), 191–204.

[5] Ran Cheng and Yaochu Jin. 2015. A social learning particle swarm optimization algorithm for scalable optimization. *Information Sciences* 291 (2015), 43–60.

[6] Weishan Dong, Tianshi Chen, Peter Tino, and Xin Yao. 2013. Scaling up estimation of distribution algorithms for continuous optimization. *Evolutionary Computation, IEEE Transactions on* 17, 6 (2013), 797–822.

[7] ChiKeong Goh and Kay Chen Tan. 2009. A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization. *Evolutionary Computation, IEEE Transactions on* 13, 1 (2009), 103–127.

[8] Xiao-Min Hu, Fei-Long He, Wei-Neng Chen, and Jun Zhang. 2017. Cooperation coevolution with fast interdependency identification for large scale optimization. *Information Sciences* 381 (2017), 142–160.

[9] Frank Hutter, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. 2014. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence* 206 (2014), 79–111.

[10] Ata Kabán, Jakramate Bootkrajang, and Robert John Durrant. 2015. Toward large-scale continuous EDA: A random matrix theory perspective. *Evolutionary computation* (2015).

[11] Antonio LaTorre, Santiago Muelas, and José-María Peña. 2011. A MOS-based dynamic memetic differential evolution algorithm for continuous optimization: a scalability test. *Soft Computing* 15, 11 (2011), 2187–2199.

[12] Anany Levitin and Soumen Mukherjee. 2011. *Introduction to the design & analysis of algorithms*. Vol. 3. Pearson Education.

Table 1: The optimization results of the CCOS, CSO, MOS and MA-SW-Chain algorithms when used to solve the CEC'2010 benchmark problems. The notation "↑/‖/↓" (or "w/t/l") represents that CCOS generated statistically "better/equally well/worse" solution quality than the other algorithm.

Func	Stats	CCOS	CSO	MOS	MA-SW-Chain
f_1	median	2.36e-13	4.40e-12‖	0.00e+00↓	2.67e-14↓
	mean	5.76e-02	4.50e-12	1.50e-28	3.80e-14
	std	2.04e-01	5.94e-13	5.55e-28	4.91e-14
f_2	median	2.52e+03	7.33e+03↑	0.00e+00↓	8.47e+02↓
	mean	2.56e+03	7.42e+03	0.00e+00	8.40e+02
	std	2.87e+02	2.86e+02	0.00e+00	4.88e+01
f_3	median	1.69e+00	2.53e-09↓	0.00e+00↓	5.16e-13↓
	mean	1.69e+00	2.60e-09	0.00e+00	5.76e-13
	std	1.94e-01	2.62e-10	0.00e+00	2.73e-13
f_4	median	1.96e+10	7.26e+11↑	4.94e+11↑	3.10e+11↑
	mean	1.95e+10	7.25e+11	5.16e+11	2.97e+11
	std	2.23e+09	1.23e+11	1.85e+11	6.19e+10
f_5	median	4.48e+07	2.00e+06↓	5.00e+08↑	2.30e+08↑
	mean	4.85e+07	2.86e+06	4.93e+08	2.18e+08
	std	1.03e+07	1.79e+06	6.93e+07	5.75e+07
f_6	median	1.29e+00	8.23e-07↓	1.97e+07↑	2.45e+00↑
	mean	1.28e+00	8.21e-07	1.97e+07	1.42e+05
	std	1.23e-01	2.68e-08	1.15e+05	3.96e+05
f_7	median	3.94e-13	2.04e+04↑	2.27e+07↑	7.94e-03↑
	mean	9.15e-12	2.01e+04	3.54e+07	1.17e+02
	std	2.61e-11	3.86e+03	3.22e+07	2.37e+02
f_8	median	3.47e-09	3.87e+07↑	2.14e+06↑	2.76e+06↑
	mean	3.19e+05	3.87e+07	3.75e+06	6.90e+06
	std	1.10e+06	6.81e+04	4.40e+06	1.90e+07
f_9	median	6.73e+06	7.05e+07↑	1.18e+07↑	1.48e+07↑
	mean	6.79e+06	7.03e+07	1.13e+07	1.49e+07
	std	9.97e+05	5.73e+06	1.61e+06	1.61e+06
f_{10}	median	2.39e+03	9.59e+03↑	6.35e+03↑	2.02e+03‖
	mean	2.62e+03	9.60e+03	6.28e+03	2.01e+03
	std	7.17e+02	7.67e+01	3.12e+02	1.59e+02
f_{11}	median	2.17e-13	3.80e-08‖	2.84e+01↑	3.77e+01↑
	mean	1.61e-01	4.02e-08	3.08e+01	3.86e+01
	std	4.68e-01	5.12e-09	6.07e+00	8.06e+00
f_{12}	median	5.41e+00	4.23e+05↑	3.46e+03↑	3.09e-06↓
	mean	8.03e+01	4.37e+05	4.39e+03	3.24e-06
	std	3.72e+02	6.22e+04	2.92e+03	5.78e-07
f_{13}	median	3.85e+02	5.47e+02↑	3.19e+02‖	8.61e+02↑
	mean	4.10e+02	6.29e+02	3.32e+02	9.83e+02
	std	1.09e+02	2.32e+02	1.19e+02	5.66e+02
f_{14}	median	4.47e+07	2.52e+08↑	2.04e+07↓	3.83e+07‖
	mean	4.45e+07	2.49e+08	2.05e+07	3.85e+07
	std	3.32e+06	1.53e+07	3.60e+06	3.86e+06
f_{15}	median	4.52e+03	1.01e+04↑	1.29e+04↑	2.67e+03↓
	mean	4.56e+03	1.01e+04	1.29e+04	2.68e+03
	std	1.68e+02	5.23e+01	3.48e+02	9.95e+01
f_{16}	median	1.49e-13	5.75e-08‖	3.97e+02↑	9.32e+01↑
	mean	1.88e-01	5.89e-08	3.96e+02	9.95e+01
	std	5.53e-01	5.61e-09	3.47e+00	1.53e+01
f_{17}	median	4.22e+03	2.22e+06↑	7.30e+03↑	1.28e+00↓
	mean	4.28e+03	2.20e+06	8.45e+03	1.27e+00
	std	1.22e+03	1.55e+05	5.04e+03	1.24e-01
f_{18}	median	1.13e+03	1.76e+03↑	7.78e+02↓	1.41e+03↑
	mean	1.15e+03	1.73e+03	8.96e+02	1.57e+03
	std	1.66e+02	5.22e+02	4.03e+02	6.73e+02
f_{19}	median	9.32e+05	1.00e+07↑	5.71e+05↓	3.75e+05↓
	mean	9.47e+05	1.01e+07	5.49e+05	3.80e+05
	std	7.61e+04	5.64e+05	8.38e+04	2.34e+04
f_{20}	median	2.43e+03	9.85e+02↓	7.40e+01↓	1.04e+03↓
	mean	2.48e+03	1.05e+03	9.23e+01	1.06e+03
	std	2.41e+02	1.49e+02	8.99e+01	9.38e+01
Sum	w/t/l	-	13/3/4	12/1/7	10/2/8

[13] Ke Li, Alvaro Fialho, Sam Kwong, and Qingfu Zhang. 2014. Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation* 18, 1 (2014), 114–130.

[14] Yi Mei, Xiaodong Li, and Xin Yao. 2014. Cooperative coevolution with route distance grouping for large-scale capacitated arc routing problems. *Evolutionary Computation, IEEE Transactions on* 18, 3 (2014), 435–449.

[15] Yi Mei, Mohammad Nabi Omidvar, Xiaodong Li, and Xin Yao. 2016. A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization. *ACM Trans. Math. Software* 42, 2 (2016), 13.

[16] Daniel Molina, Manuel Lozano, and Francisco Herrera. 2010. MA-SW-Chains: Memetic algorithm based on local search chains for large scale continuous global optimization. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 1–8.

[17] Mario A Muñoz, Yuan Sun, Michael Kirley, and Saman K Halgamuge. 2015. Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges. *Information Sciences* 317 (2015), 224–245.

[18] Mohammad Nabi Omidvar, Xiaodong Li, Yi Mei, and Xin Yao. 2014. Cooperative co-evolution with differential grouping for large scale optimization. *Evolutionary Computation, IEEE Transactions on* 18, 3 (2014), 378–393.

[19] Mohammad Nabi Omidvar, Xiaodong Li, and Ke Tang. 2015. Designing benchmark problems for large-scale continuous optimization. *Information Sciences* 316 (2015), 419–436.

[20] Mohammad Nabi Omidvar, Xiaodong Li, and Xin Yao. 2011. Smart use of computational resources based on contribution for cooperative co-evolutionary algorithms. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM, 1115–1122.

[21] Mohammad Nabi Omidvar, Ming Yang, Yi Mei, Xiaodong Li, and Xin Yao. 2017. DG2: A faster and more accurate differential grouping for large-scale black-box optimization. *IEEE Transactions on Evolutionary Computation* 21, 6 (2017), 929–942.

[22] Mitchell A Potter and Kenneth A De Jong. 1994. A cooperative coevolutionary approach to function optimization. In *Parallel problem solving from nature PPSN III*. Springer, 249–257.

[23] Eman Sayed, Daryl Essam, Ruhul Sarker, and Saber Elsayed. 2015. Decomposition-based evolutionary algorithm for large scale constrained problems. *Information Sciences* 316 (2015), 457–486.

[24] David J Sheskin. 2003. *Handbook of parametric and nonparametric statistical procedures*. CRC Press.

[25] Yuan Sun, Michael Kirley, and Saman Kumara Halgamuge. 2015. Extended differential grouping for large scale global optimization with direct and indirect variable interactions. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, 313–320.

[26] Yuan Sun, Michael Kirley, and Saman Kumara Halgamuge. 2017. A recursive decomposition method for large scale optimization. *IEEE Transactions on Evolutionary Computation* (2017). https://doi.org/10.1109/TEVC.2017.2778089

[27] Yuan Sun, Michael Kirley, and Saman K Halgamuge. 2017. A memetic cooperative co-evolution model for large scale continuous optimization. In *Australasian Conference on Artificial Life and Computational Intelligence*. Springer, 291–300.

[28] Ke Tang, X Yao, and Pn Suganthan. 2010. Benchmark functions for the CEC'2010 special session and competition on large scale global optimization. *Technique Report, USTC, Natrue Inspired Computation and Applications Laboratory* 1 (2010), 1–23.

[29] LinYu Tseng and Chun Chen. 2008. Multiple trajectory search for large scale global optimization. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*. IEEE, 3052–3059.

[30] Hui Wang, Zhijian Wu, and Shahryar Rahnamayan. 2011. Enhanced opposition-based differential evolution for solving high-dimensional continuous optimization problems. *Soft Computing* 15, 11 (2011), 2127–2140.

[31] Thomas Weise, Raymond Chiong, and Ke Tang. 2012. Evolutionary optimization: Pitfalls and booby traps. *Journal of Computer Science and Technology* 27, 5 (2012), 907–936.

[32] Ming Yang, Mohammad Nabi Omidvar, Changhe Li, Xiaodong Li, Zhihua Cai, Borhan Kazimipour, and Xin Yao. 2017. Efficient resource allocation in cooperative co-evolution for large-scale global optimization. *IEEE Transactions on Evolutionary Computation* 21, 4 (2017), 493–505.

[33] Zhenyu Yang, Ke Tang, and Xin Yao. 2008. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences* 178, 15 (2008), 2985–2999.

[34] Zhenyu Yang, Ke Tang, and Xin Yao. 2008. Self-adaptive differential evolution with neighborhood search. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*. IEEE, 1110–1116.

[35] Aimin Zhou and Qingfu Zhang. 2016. Are all the subproblems equally important? Resource allocation in decomposition-based multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* 20, 1 (2016), 52–64.