An Adaptive Coevolutionary Differential Evolution Algorithm for Large-scale Optimization

Zhenyu Yang, Jingqiao Zhang, Ke Tang, Xin Yao and Arthur C. Sanderson

Abstract— **In this paper, we propose a new algorithm, named JACC-G, for large scale optimization problems. The motivation is to improve our previous work on grouping and adaptive weighting based cooperative coevolution algorithm, DECC-G [1], which uses random grouping strategy to divide the objective vector into subcomponents, and solve each of them in a cyclical fashion. The adaptive weighting mechanism is used to adjust all the subcomponents together at the end of each cycle. In the new JACC-G algorithm: (1) A most recent and efficient Differential Evolution (DE) variant, JADE [2], is employed as the subcomponent optimizer to seek for a better performance; (2) The adaptive weighting is time-consuming and expected to work only in the first few cycles, so a detection module is added to prevent applying it arbitrarily; (3) JADE is also used to optimize the weight vector in adaptive weighting process instead of using a basic DE in previous DECC-G. The efficacy of the proposed JACC-G algorithm is evaluated on two sets of widely used benchmark functions up to 1000 dimensions.**

I. Introduction

EVOLUTIONARY optimization has achieved great success on many numerical and combinatorial optimization problems in recent years [3]. However, as evolutionary algorithms (EAs) are applied to increasingly large and complex problems, their scalability has become one of the most urgent challenges. In general, the performance of conventional EAs deteriorates rapidly as the dimensionality of the search space increases [4]. To tackle the puzzle, our research is focused on solving problems that are at least one magnitude larger than the state-of-the-art in evolutionary optimization.

Cooperative coevolutionary paradigm has been proved to be a promising attempt for tackling those large scale problems. Cooperative Coevolution (CC) [5] was originally proposed as a general framework for applying EAs to large and complex problems using a divide-and-conquer strategy. In CC, the objective problem (such as an objective variables vector) is decomposed into smaller subproblems and each of them is assigned to a species (i.e. subpopulation). The species evolve mostly separately with cooperation happening only during fitness evaluation. In the domain of numerical optimization, a large scale problem is corresponding to a high dimensional objective vector. The process of EAs under the CC framework for large scale numerical optimization can be summarized into three major steps [1]:

1) **Problem Decomposition:** Decompose the high dimensional objective vector into smaller subcomponents.
2) **Subcomponent Optimization:** Evolve each subcomponent separately using a certain EA.
3) **Cooperative Combination:** Combine the solutions of all subcomponents to form the final solution.

Since Step 3) is often embedded in the fitness evaluation operations, the problem decomposition approach and the subcomponent optimization method become two most critical issues in the CC framework [6]. Initial efforts for problem decomposition used two simple methods, i.e., one-dimensional based and splitting-in-half strategies [7], [8], [9]. The one-dimensional based strategy decomposes a high-dimensional vector into single variables. Since it does not consider interdependencies among variables, it is unable to tackle nonseparable problems, in which interaction exists between objective variables. The splitting-in-half strategy always decompose a high-dimensional vector into two equal halves and thus reduces an n-dimensional problems into two $\frac{n}{2}$-dimensional problems. If n is large, the $\frac{n}{2}$-dimensional problems would still be very large and challenging to solve. To overcome these shortcomings, we have proposed a more general **random grouping** based problem decomposition method in [1]. It divides a high-dimensional vector into several subcomponents according to a predefined group size, and thus each subcomponent contains only a subset of the original objective variables. To further increase the probability of grouping interacted variables in the same subcomponent, the grouping structure will be changed dynamically after each *cycle*, which refers to one complete evolution of all subcomponents. Since interdependencies between different subcomponents may still exist after each time of random grouping, an **adaptive weighting** strategy is applied at the end of every cycle to provide an extra chance to evolve all the objective variables at the same time. CC based EAs with this kind of decomposition is denoted as EACC-G [1] previously. As for another important aspect in the CC framework, i.e., subcomponent optimization, any existing EAs can be introduced as the subcomponent optimizer. The task is to select an effective and efficient EA and amend it if necessary according to the characteristics of these subcomponents.

Although EACC-G has shown promising performance on many benchmark functions, the random grouping and adaptive weighting strategies are still in their infancy, and can be improved further. One improvement can be carried

Zhenyu Yang, Ke Tang and Xin Yao are with the Nature Inspired Computation and Applications Laboratory, the Department of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China. Xin Yao is also with CERCIA, the School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, U.K. Jingqiao Zhang and Arthur C. Sanderson are with the Center for Automation Technologies and Systems, Rensselaer Polytechnic Institute, Troy, NY 12180, U.S.A. (emails: zhyuyang@mail.ustc.edu.cn, zhangj14@rpi.edu, ketang@ustc.edu.cn, x.yao@cs.bham.ac.uk, sandea@rpi.edu).

Corresponding author: Ke Tang (+86–551–3600754).

978-1-4244-2959-2/09/$25.00 ©

out through controlling when to and when not to apply the adaptive weighting strategy. It is obvious that saving fitness evaluations (FEs) is very important for EAs. Efficient algorithms should consume FEs only when it is very likely to make progress. In our recent experience, we found the adaptive weighting is quite FEs consuming and it often does not help in the **later stages** of evolutionary search. It would be desirable if the adaptive weighting is adopted only when it is effective to help EACC-G make progress. Otherwise, the corresponding FEs should be saved for subsequent evolution. Another direct improvement to EACC-G is to introduce some recently advanced EAs as subcomponent optimizer. Previously, we used a Differential Evolution (DE) variant, SaNSDE [10], as subcomponent optimizer and thus implemented a DECC-G algorithm. However, along with the development of evolutionary optimization, even in the domain of DE, some more effective and efficient methods have been proposed [11], [12], [13], [14]. We are expected to get a better scratch line for optimizing large scale problems with these advanced DE variants.

Based on the directions above, we propose an improved version of DECC-G algorithm in this paper. In the new algorithm, a probability based mechanism is implemented to detect when to and when not to apply the adaptive weighting strategy. The detection is useful to avoid wasting FEs on worthless adaptive weighting process. As for the subcomponent optimizer, we utilize a most recent adaptive DE variants, JADE [2], which has shown very fast and reliable convergence performance on a set of widely used benchmark functions. It adopted a novel greedy "DE/current-to-pbest" mutation strategy and updates control parameters to appropriate values in an adaptive manner. To make it more robust on difficult multimodal problems, we also amend JADE slightly by introducing some **local pbest** (i.e. lpbest) members to replace its pbest. A similar local knowledge used in [15] has been verified to be useful for increasing DE's exploration ability. After considering the name of both EACC-G and JADE, the new large scale optimization algorithm is denoted as JACC-G in this paper. The performance of JACC-G will be evaluated on two sets of benchmark functions up to 1000 dimensions.

The rest of this paper is organized as follows: Section II gives the background; Section III describes the proposed JACC-G in detail; Section IV presents the experimental results; Finally, Section V concludes this paper briefly.

II. Background

A. CC with Random Grouping and Adaptive Weighting

"As evolutionary algorithms are applied to the solution of increasingly complex systems, explicit notions of modularity must be introduced to provide reasonable opportunities for solutions to evolve in the form of interacting coadapted subcomponents" [5]. Examples of this show up in the need for rule hierarchies in classifier systems and subroutines in genetic programming [8]. CC is a general framework for applying EAs to large and complex problems using a divide-and-conquer strategy. In CC, the objective problem (such as a vector) is decomposed into smaller subproblems and each of them is assigned to a species (i.e. subpopulation). The species are evolved mostly separately with the only cooperation happening during fitness evaluations. This implies the problem decomposition approach and the subproblem solver are the most critical issues in the CC framework.

As analyzed in Section I, early problem decomposition methods such as the one-dimensional based and splitting-in-half strategies both have their own limitations. So a more general CC framework with random grouping and adaptive weighting is proposed in [1]. The main steps of the new CC framework which is denoted as EACC-G can be summarized as follows:

1) Initialize a population in the search space randomly:
$$\mathbf{P} = \{P(i,j) \mid i = (1, \cdots, NP), j = (1, \cdots, n)\}$$
where NP denotes the population size, and n denotes the dimension of objective vector.
2) Set $c = 1$ to start a new *cycle*.
3) Divide the n-dimensional objective vector into several groups randomly based on a group size s, i.e., $\mathbf{G} = \{\mathbf{G}_1, \mathbf{G}_2, \cdots, \mathbf{G}_m\}$ (assuming $n = s * m$). Obviously, each $\mathbf{G}_i$ represents a subcomponent.
4) Construct the subpopulation $\mathbf{P}'$ based on the population $\mathbf{P}$ and subcomponent $\mathbf{G}_c$:
$$\mathbf{P}' = \{P(i,j) \mid i \in \{1, \cdots, NP\}, j \in \mathbf{G}_c\}$$
5) Optimize the subcomponent $\mathbf{G}_c$ with subpopulation $\mathbf{P}'$ using a certain EA for a predefined number of FEs.
6) If $c < m$ then $c = c + 1$, and go to Step 4).
7) Select the best, the worst and a random individuals of current population $\mathbf{P}$, and put them into a set S.
8) For each member in S: (a) assign a weight to each of its subcomponents; (b) optimize the weights with a certain EA for a predefined number of FEs; (c) if improved, update the member by applying the best weights achieved.
9) Stop if halting criteria are satisfied; otherwise go to Step 2) for the next *cycle*.

Here a *cycle* consists of one complete evolution of all subcomponents. The main advantages of EACC-G are: (1) It evolves a group of variables (called a subcomponent) together. The trade-off between capturing variable-interaction and not exceeding EAs' capability can be well controlled by the parameter group size; (2) The grouping structure will be changed dynamically, which can further increase the probability of grouping interacted variables together; (3) An adaptive weighting is executed among all subcomponents at the end of each cycle to provide an extra chance to evolve all the variables at the same time.

We further describe how and why the adaptive weighting strategy works.

1) For any individual, $\mathbf{x} = (x_1, x_2, \cdots, x_n)$, of a population, it is true that:
$$f(\mathbf{x}) \equiv f(\mathbf{w_c} \cdot \mathbf{x})$$

where $\mathbf{w_c} = (1, 1, \cdots, 1)$ is a constant weight vector.

2) To obtain better fitness value, we can apply a weight w_i to each component of $\mathbf{x}$, and then optimize the weight vector. So we achieve:

$$\min_{\mathbf{w}} f(\mathbf{w} \cdot \mathbf{x}) \leq f(\mathbf{w_c} \cdot \mathbf{x}) \equiv f(\mathbf{x})$$

where $\mathbf{w} = (w_1, w_2, \cdots, w_n)$ is the weight vector over the individual $\mathbf{x}$.

3) However, optimizing the weight vector $\mathbf{w}$ is as hard to optimize as the original individual $\mathbf{x}$, because they are in the same high dimension. EACC-G splits the n-dimensional vector $\mathbf{x}$ into m ($m \ll n$) subcomponents, so we can alternatively apply a weight to each of these subcomponents and only optimize a much lower dimensional vector $\tilde{\mathbf{w}} = (\tilde{w}_1, \tilde{w}_2, \cdots, \tilde{w}_m)$:

$$\mathbf{w}' = (\underbrace{\tilde{w}_1, \cdots, \tilde{w}_1}_{s}, \underbrace{\tilde{w}_2, \cdots, \tilde{w}_2}_{s}, \cdots, \underbrace{\tilde{w}_m, \cdots, \tilde{w}_m}_{s})$$

$$\min_{\mathbf{w}} f(\mathbf{w} \cdot \mathbf{x}) \leq \min_{\mathbf{w}'} f(\mathbf{w}' \cdot \mathbf{x}) \leq f(\mathbf{x})$$

where s denotes the dimension of each subcomponent and m denotes the number of subcomponents (assuming $n = m * s$).

Thus, the adaptive weighting strategy provides a tradeoff between optimizing a high-dimensional vector $\mathbf{w}$ and no weighting at all. Furthermore, since the variables of a subcomponent is controlled integrally by changing the weight of it, the process of optimizing the weight vector can also be viewed as a coarse adjustment over all subcomponents.

B. JADE: An Adaptive Differential Evolution Algorithm

Differential Evolution (DE) [16], [17] is a simple yet effective population-based algorithm for global optimization. We have recently proposed a new adaptive differential evolution algorithm JADE [2] [14]. JADE implements a novel mutation strategy and updates control parameters in an adaptive manner, while its initialization, crossover and selection operations follow the basic procedure of classic DE.

Consider the population (or subpopulation if the subcomponent optimization is concerned) at a certain generation, $\{\mathbf{x}_i = (x_{1,i}, x_{2,i}, \cdots, x_{s,i}) | i = (1, \cdots, NP)\}$, where NP is the population size and s is the group size (i.e., the dimension of a subcomponent). In JADE, mutation vectors are generated according to a relatively greedy strategy DE/current-to-pbest/1 as follows:

$$\mathbf{v}_i = \mathbf{x}_i + F_i(\mathrm{x}^p_{\mathrm{best}} - \mathbf{x}_i) + F_i(\mathbf{x}_{r1} - \tilde{\mathbf{x}}_{r2}), \qquad (1)$$

where x^p_{best} is randomly chosen as one of the best $100p\%$, $p \in (0, 1]$, individuals in the current population, $\mathbf{x}_{r1} (r1 \neq i)$ is a vector randomly selected from the population, and $\tilde{\mathbf{x}}_{r2}$ is a vector (distinct from $\mathbf{x}_i$ and $\mathbf{x}_{r1}$) randomly chosen from the union of the current population and an external archive of inferior solutions. The archive is initialized to be empty and then we add to it the parent solutions that fail in the selection process of (3) at each generation. If the archive size exceeds a certain threshold, say NP, then some solutions are randomly removed from the archive to keep its size at NP.

After mutation, a binary crossover operation forms offspring vectors $\mathbf{u}_i = (u_{1,i}, u_{2,i}, \cdots, u_{s,i})$ in the following manner:

$$u_{j,i} = \begin{cases} v_{j,i} & \text{if} \mathrm{rand}_j(0,1) \leq CR_i \text{ or } j = j_{\mathrm{rand}} \\ x_{j,i} & \text{otherwise,} \end{cases} \qquad (2)$$

where rand_j(a,b) is a uniform random number on the interval [a, b] and newly generated for each j, $j_{\mathrm{rand}} = \mathrm{randint}_i(1, s)$ is an integer randomly chosen from 1 to s and newly generated for each i, and the crossover probability, $CR_i \in$ [0, 1], roughly corresponds to the average fraction of vector components that are inherited from the mutation vector. Then, the offspring vector is compared to the corresponding parent vector and the better one survives and becomes a parent vector in the next generation, i.e.,

$$\mathbf{x}_i = \begin{cases} \mathbf{u}_i & \text{if } f(\mathbf{u}_i) \leq f(\mathbf{x}_i) \\ \mathbf{x}_i & \text{otherwise,} \end{cases} \qquad (3)$$

In JADE, the constant parameter F_i and CR_i in (1) and (2) are randomly regenerated at each generation; i.e.,

$$CR_i = \mathrm{randn}_i(\mu_{CR}, 0.1), \qquad (4)$$

$$F_i = \mathrm{randc}_i(\mu_F, 0.1), \qquad (5)$$

where the mean μ_{CR} and location parameter μ_F are updated in an adaptive manner:

$$\mu_{CR} = (1 - c)\mu_{CR} + c \cdot \mathrm{mean}_A(S_{CR}), \qquad (6)$$

$$\mu_F = (1 - c)\mu_F + c \cdot \mathrm{mean}_L(S_F), \qquad (7)$$

where S_{CR} and S_F are the respective sets of all successful crossover probabilities and successful mutation factors obtained in the selection process of (3) at the current generation, c is a positive constant between 0 and 1 and $\mathrm{mean}_A(\cdot)$ is the usual arithmetic mean and $\mathrm{mean}_L(\cdot)$ is the Lehmer mean

$$\mathrm{mean}_L(S_F) = \frac{\sum_{F \in S_F} F^2}{\sum_{F \in S_F} F}, \qquad (8)$$

which plays more weight on larger mutation factor F to improve evolutionary search progress.

JADE has shown its success for a variety of benchmark problems of different characteristics [2] [14] and in different real world applications including credit decision making [18], air traffic management [19], and combinatorial auction [20].

III. JACC-G: The New Improved Algorithm

In this section, we propose a new CC based algorithm, JACC-G, for large scale optimization. The JACC-G is an advanced version of the previous DECC-G with improvements in the following three aspects:

1) The adaptive weighting strategy is altered from a necessary module to an optional one. A probability based detection mechanism will be adopted to control when to and when not to apply the adaptive weighting. The motivation is to avoid wasting FEs if the adaptive weighting becomes worthless.

2) The JADE algorithm is adopted as the subcomponent optimizer. It is revised by replacing its pbest with local pbest, lpbest, to make it more robust for difficult multimodal problems.
3) In the adaptive weighting mechanism, we also need to specify a certain optimizer since finding out the best weights is a optimization problem itself. A classical DE is used in DECC-G for simplification. Obviously, we can also introduce the advanced JADE as the weight optimizer to seek a better performance.

Except these improvements, JACC-G is the same with EACC-G described in Section II-A. So the new algorithm can be formulated easily by replacing all the "a certain EA" with "JADE" in Steps 5) and 8), and adding execution condition, which will be given in Section III-A, in Step 8). How and why these modifications in JACC-G work is given in the following Subsections III-A and III-B in detail.

A. Detection on When to Apply Adaptive Weighting

As mentioned before, seeking for the optimal weights with the adaptive weighting strategy is an optimization problem itself. Thus, it also consumes part of the total FEs during the process of evolution. In EACC-G, we specified a same number of FEs for applying each adaptive weighting with that of each subcomponent optimization in each cycle. For example, given a 1000-dimensional problem and 100 as group size, in each cycle EACC-G has to optimize 10 subcomponents and execute 3 times of adaptive weighting operations. In such a case, adaptive weighting takes about $\frac{3}{10+3} \simeq 23\%$ of the total computational effort, which is quite significant.

However, the adaptive weighting is not always helpful in every cycle. To be specific, it is more likely to make progress during the early stages of evolution, i.e., the first several cycles. This is because each weight in the adaptive weighting process controls a group of variables. A modification of the weight will cause a perturbation to all the controlled variables. So this is a quite coarse adjustment over objective variables. With the evolution progressing, the variables need to be controlled more and more accurately. So fine turn rather than the coarse adjustment is more likely to provide improvements. To prevent using the adaptive weighting arbitrarily, we introduce a probability wp, which is initialized as $wp = 1$, to control when to apply the adaptive weighting strategy. And thus the Step 8) of EACC-G in Section II-A is revised as Algorithm 1. The probability wp will be reduced quickly when the adaptive weighting becomes ineffective; thus it is useful to avoid wasting FEs on worthless adaptive weighting process.

```
for (each member in S) do
    if (U(0,1) < wp) then
        Step 8) (a);
        Step 8) (b);
        if (Improved) then
            Step 8) (c);
            wp = min(1, wp * 2) ;
        else
            wp = wp/2 ;
        end
    end
end
```

Algorithm 1: Adaptive Weighting Detection

B. JADE with Local pbest

JADE has shown a significant better performance than the classic DE and several other adaptive DE variants; however its "DE/current-to-pbest" mutation strategy is still quite greedy. This is because the top $100p\%$ individuals are always selected as **pbest** members with probability 1. It might make JADE less robust on some difficult multimodal problems, since greedy mutation strategies have a reputation of possibly leading to false convergence to some competitive local optima. To further improve JADE's robustness on such problems, we propose a new method to select some *local* best members, denoted as **lpbest**, to replace its pbest members as follows:

1) Randomly assign each solution into one of K groups, where $K = \lfloor p * NP \rfloor$, p and NP have the same meanings with that of JADE.
2) The best solution in each group is considered as an lpbest member.

It is clear that every solution can be selected to be an lpbest member with a non-zero probability. If a solution is the i-th best in the population, it is selected as lpbest if and only if it is not in the same group as other $i-1$ even better solutions. By simple combinatorial calculation, we can show that the probability is $(1-1/K)^{(i-1)}$ for the i-th best solution to become an lpbest member. With such lpbest technique, we expect the JADE to be able to find the global optimum consistently.

To demonstrate the effect of lpbest, we give a case study on the well-known Generalized Schwefel's Problem 2.26. The multimodal problem is denoted as f_8 in [21], and can also be seen in Eq. (9). As shown in Fig. 1, the landscape of f_8 appears to be very "rugged", and it is often regarded as being difficult to optimize. Experiments are conducted on 50 and 100 dimension of f_8. Both JADE with pbest and lpbest were run 100 independent runs. The parameter settings for JADE are $NP = 100$, $p = 0.05$ and $1/c = 10$. The stopping criterion is set to $50 \times s$ generations, where s denotes the dimension of tested problem. The error values, which are the distance between found fitness and the global optimum, are given in Table I. The SR in the table denotes the success rate of each algorithm converged to the global optimum. It can be found that JADE with pbest failed to find the global optimum in 3 and 6 out of the 100 runs for the 50 and 100 dimension problems, respectively. In comparison, JADE with lpbest is more robust as it appears able to converge to the

global optimum consistently.

$$f_8(\mathbf{x}) = -\sum_{i=1}^{n} \left(x_i \sin(\sqrt{|x_i|})\right), -500 \le x_i \le 500 \quad (9)$$

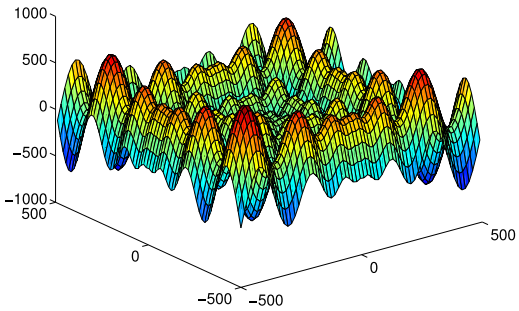

Fig. 1. The two-dimensional version of f_8.

TABLE I

SIMULATED RESULTS OF JADE WITH PBEST AND LPBEST ON THE GENERALIZED SCHWEFEL'S PROBLEM 2.26. THE RESULTS OF 100 INDEPENDENT RUNS ARE SORTED FROM THE 1ST TO 100TH.

# of runs	JADE(50-D)		JADE(100-D)	
	pbest	lpbest	pbest	lpbest
1st	1.82e-11	1.82e-11	1.09e-10	1.09e-10
12nd	1.82e-11	1.82e-11	1.09e-10	1.09e-10
23rd	1.82e-11	1.82e-11	1.09e-10	1.09e-10
34th	1.82e-11	1.82e-11	1.09e-10	1.09e-10
45th	1.82e-11	1.82e-11	1.09e-10	1.09e-10
56th	1.82e-11	1.82e-11	1.09e-10	1.09e-10
67th	1.82e-11	1.82e-11	1.09e-10	1.09e-10
78th	1.82e-11	2.18e-11	1.09e-10	1.09e-10
89th	1.82e-11	2.55e-11	1.09e-10	1.09e-10
100th	1.18e+02	7.64e-11	1.18e+02	1.09e-10
SR	97/100	100/100	94/100	100/100
Mean	3.55e+00	2.05e-11	7.11e+00	1.09e-10
Std	2.03e+01	6.97e-12	2.83e+01	0.00e+00

IV. EXPERIMENTAL STUDIES

A. Experimental Setup

The performance of the proposed JACC-G algorithm will be evaluated on both a set of classical benchmark functions [21] and a new suite of test functions provided by CEC'2005 special session [22]. The dimensions of all test functions are set to 1000. The algorithms used for comparison are JACC-G, DECC-G and the non-CC algorithm JADE. The DECC-G is an implementation of EACC-G framework by using a DE variant, SaNSDE [10] , as subcomponent optimizer. In order to make the comparisons as fair as possible, the same number of fitness evaluations (FEs) will be used for all algorithms as the stopping criterion, which is set to 5,000,000. The population size of all the algorithms are set to 100, and the subcomponent dimensions of JACC-G and DECC-G are set to 100.

B. Simulation Results

We first test JACC-G's performance on 13 classical benchmark functions. Among the 13 functions, f_1–f_7 are unimodal functions and functions f_8–f_{13} are multimodal functions where the number of local minima increases exponentially as the problem dimension increases. Functions f_4 and f_5 are nonseparable, while others are separable. Details of these functions can be found in the appendix of [21]. In each cycle, $10,000$ FEs are assigned to JADE to optimize each of the subcomponents. The average results over 25 independent runs on these functions are summarized in Table II.

TABLE II

EXPERIMENTAL COMPARISON ON THE 13 CLASSICAL FUNCTIONS. ALL RESULTS HAVE BEEN AVERAGED OVER 25 INDEPENDENT RUNS.

Test Func	JADE Mean (Std)	DECC-G Mean (Std)	JACC-G Mean (Std)
f_1	9.4e-03 (2.8e-02)	2.2e-25 (2.7e-26)	**2.7e-80 (5.2e-80)**
f_2	1.1e+02 (3.5e+02)	5.4e-14 (1.3e-14)	**2.3e-20 (4.5e-20)**
f_3	8.2e+00 (2.5e+01)	3.1e-03 (1.6e-03)	**2.4e-10 (4.3e-10)**
f_4	4.3e+01 (2.3e+00)	1.0e-01 (4.3e-02)	**8.0e-05 (2.7e-05)**
f_5	2.0e+03 (1.6e+02)	987.33 (3.2e-01)	**983.04 (3.7e-01)**
f_6	2.6e+04 (3.5e+03)	0.0e+00 (0.0e+00)	0.0e+00 (0.0e+00)
f_7	5.3e+00 (1.3e+00)	8.4e-03 (7.6e-04)	**1.2e-03 (3.9e-04)**
f_8	-418225 (5.2e+02)	-418983 (2.1e-08)	-418983 (2.4e-10)
f_9	1.1e+00 (1.8e+00)	3.6e-16 (8.4e-16)	**0.0e+00 (0.0e+00)**
f_{10}	9.3e+00 (4.8e-01)	2.2e-13 (2.4e-14)	**1.4e-14 (1.3e-14)**
f_{11}	4.8e-01 (5.0e-01)	1.0e-15 (1.3e-16)	**0.0e+00 (0.0e+00)**
f_{12}	1.4e+00 (5.2e-01)	6.9e-25 (8.1e-26)	**1.4e-32 (7.2e-33)**
f_{13}	6.4e+01 (7.5e+01)	2.6e-21 (5.4e-21)	**1.3e-32 (5.6e-48)**

As shown in Table II, both DECC-G and JACC-G obtain much better results than the non-CC algorithm JADE. This confirms the advantages of CC algorithms over non-CC ones for large scale optimization problems. Comparing to DECC-G, the JACC-G algorithm performs better on all of the tested functions, except they show similar performance on functions f_6 and f_8. The differences are most significant on functions f_1, f_2 and f_4. Since DECC-G and JACC-G have adopted a very similar CC framework, it can be inferred that the advanced subcomponent optimizer JADE and the improved adaptive weighting strategy are the most contributing mechanisms of JACC-G.

To evaluate the JACC-G algorithm further, experiments are also conducted on a new set of benchmark functions, which was provided by the CEC'2005 Special Session. It includes 25 functions with varying complexity. Among them, functions f_{cec1}–f_{cec5} are unimodal while others multimodal. Detailed descriptions of them can be found in [22]. Many of them are the shifted, rotated, expanded and/or combined variants of the classical functions. Some of these changes cause them to be more resistant to simple search tricks. Other changes, such as rotation, transfer separable functions into nonseparable ones, which will be particularly challenging in large scale optimization. Previously, eight representative functions (out of 25) were used to evaluate DECC-G [1], so we still use these functions in this paper, including two separable functions (f_{cec1} and f_{cec9}) and six nonseparable

functions. To capture better the variable interactions of these nonseparable functions, $5,000$ FEs are assigned to JADE to optimize each subcomponent in each cycle. The averaged error values, which are the distances to optimum, over 25 independent runs are given in Table III. The evolution curves are also given in Fig. 2.

TABLE III
EXPERIMENTAL COMPARISON ON THE 8 CEC'2005 FUNCTIONS. ALL RESULTS HAVE BEEN AVERAGED OVER 25 INDEPENDENT RUNS.

CEC05 Func	JADE Mean (Std)	DECC-G Mean (Std)	JACC-G Mean (Std)
f_{cec1}	3.5e+02 (9.3e+02)	6.8e-13 (3.9e-14)	**5.6e-13 (1.3e-13)**
f_{cec3}	**1.2e+08 (1.4e+07)**	8.1e+08 (5.6e+07)	1.6e+08 (1.3e+07)
f_{cec5}	2.4e+05 (9.0e+03)	2.2e+05 (6.9e+03)	**2.1e+05 (9.6e+03)**
f_{cec6}	7.9e+06 (1.9e+07)	2225.12 (7.3e+02)	**2062.83 (2.4e+02)**
f_{cec8}	**21.26 (4.6e-01)**	21.59 (5.4e-03)	21.59 (6.8e-03)
f_{cec9}	9.6e-01 (3.2e+00)	6.3e+02 (2.3e+01)	**1.5e-12 (1.5e-13)**
f_{cec10}	1.1e+04 (5.8e+02)	9.7e+03 (3.5e+03)	**9.5e+03 (1.9e+03)**
f_{cec13}	**1.5e+02 (6.6e+00)**	3.6e+02 (1.7e+01)	2.4e+02 (1.2e+01)

Different than classical test functions, non-CC algorithm JADE is also competitive on the CEC'2005 benchmark functions. For example, it achieved the best results on functions f_{cec3}, f_{cec8} and f_{cec13}. That is mainly because: (1) Most of these functions (except f_{cec1} and f_{cec9}) are completely nonseparable, which means interactions exist in any two variables of the object vector. It is very challenging for decomposition based methods such as CC based algorithms; (2) JADE utilized an external archive to store former good solutions, which make it able to explore a large search space better [2]. However, the CC based algorithms also obtained better results on the other 5 functions. In the comparison of CC based algorithms, JACC-G outperformed DECC-G on all the tested functions except f_{cec8}, on which CC based algorithms are not likely to make much progress. The results on CEC'2005 functions also confirmed the efficacy of improvements in JACC-G over DECC-G.

V. CONCLUSIONS

In this paper, we proposed a JACC-G algorithm for large scale optimization, which is a further improvement to our previous work on cooperative coevolution with random grouping and adaptive weighting (DECC-G). The improvements are presented in the new JACC-G algorithm: (1) A most recent and efficient DE variant, JADE [2], is utilized as the subcomponent optimizer to seek for a better performance; (2) A detection module is added to the adaptive weighting mechanism to prevent wasting FEs on ineffective adaptive weighting processes. (3) JADE is also used to optimize the weight vector in adaptive weighting process instead of using a outdated DE algorithm in previous DECC-G. The performance of the proposed JACC-G algorithm is evaluated and discussed on both a set of 13 classical test functions [21], and a new set of 8 benchmark functions provided by CEC'2005 special session [22]. The results confirmed the efficacy of the proposed improvements.

ACKNOWLEDGMENT

This work is partially supported by the Fund for International Joint Research Program of Anhui Science & Technology Department (No. 08080703016), the National Science Foundation (Grant No. IIS-0329837), the Center for Automation Technologies and Systems (CATS) under a block grant from the New York State Office of Science, Technology, and Academic Research (NYSTAR). Xin Yao's work is also partially supported by an EPSRC grant (EP/G002339/1) on "Cooperatively Coevolving Particle Swarms for Large Scale Optimisation."

REFERENCES

[1] Z. Yang, K. Tang, and X. Yao, "Large Scale Evolutionary Optimization Using Cooperative Coevolution," *Information Sciences*, vol. 178, no. 15, pp. 2985–2999, 2008.

[2] J. Zhang and A. C. Sanderson, "JADE: Adaptive Differential Evolution with Optional External Archive," *IEEE Transactions on Evolutionary Computation*, in press, 2008.

[3] R. Sarker, M. Mohammadian, and X. Yao, *Evolutionary Optimization*. Kluwer Academic Publishers Norwell, MA, USA, 2002.

[4] F. van den Bergh and A. P. Engelbrecht, "A Cooperative Approach to Particle Swarm Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 225–239, 2004.

[5] M. Potter, "The Design and Analysis of a Computational Model of Cooperative Coevolution," Ph.D. dissertation, George Mason University, 1997.

[6] M. Potter and K. De Jong, "Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents," *Evolutionary Computation*, vol. 8, no. 1, pp. 1–29, 2000.

[7] Y. Liu, X. Yao, Q. Zhao, and T. Higuchi, "Scaling up Fast Evolutionary Programming with Cooperative Coevolution," in *Proceedings of the 2001 Congress on Evolutionary Computation*, 2001, pp. 1101–1108.

[8] M. Potter and K. De Jong, "A Cooperative Coevolutionary Approach to Function Optimization," in *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, vol. 2, 1994, pp. 249–257.

[9] Y. Shi, H. Teng, and Z. Li, "Cooperative Co-evolutionary Differential Evolution for Function Optimization," in *Proceedings of the First International Conference on Natural Computation*. Springer-Verlag, 2005, pp. 1080–1088.

[10] Z. Yang, K. Tang, and X. Yao, "Self-adaptive Differential Evolution with Neighborhood Search," in *Proceedings of the 2008 IEEE Congress on Evolutionary Computation*, 2008, pp. 1110–1116.

[11] A. K. Qin and P. N. Suganthan, "Self-adaptive differential evolution algorithm for numerical optimization," in *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, vol. 2, 2005, pp. 1785–1791.

[12] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Žumer, "Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 646–657, 2006.

[13] Z. Yang, J. He, and X. Yao, "Making a Difference to Differential Evolution," in *Advances in Metaheuristics for Hard Optimization*, Z. Michalewicz and P. Siarry, Eds. Springer, 2008, pp. 397–414.

[14] J. Zhang and A. C. Sanderson, "JADE: Self-adaptive differential evolution with fast and reliable convergence performance," in *Proceedings of the 2007 IEEE Congress on Evolutionary Computation*, 2007, pp. 2251–2258.

[15] U. K. Chakraborty, S. Das, and A. Konar, "Differential Evolution with Local Neighborhood," in *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, 2006, pp. 2042–2049.

[16] K. Price, R. Storn, and J. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*. Springer-Verlag, ISBN:3-540-20950-6, 2005.

[17] U. K. Chakraborty (Ed.), *Advances in Differential Evolution*. Springer-Verlag Berlin, 2008.

[18] J. Zhang, V. Avasarala, and R. Subbu, "Evolutionary optimization of transition probability matrices for credit decision-making," *Under 2nd review of European Journal of Operational Research*, 2008.

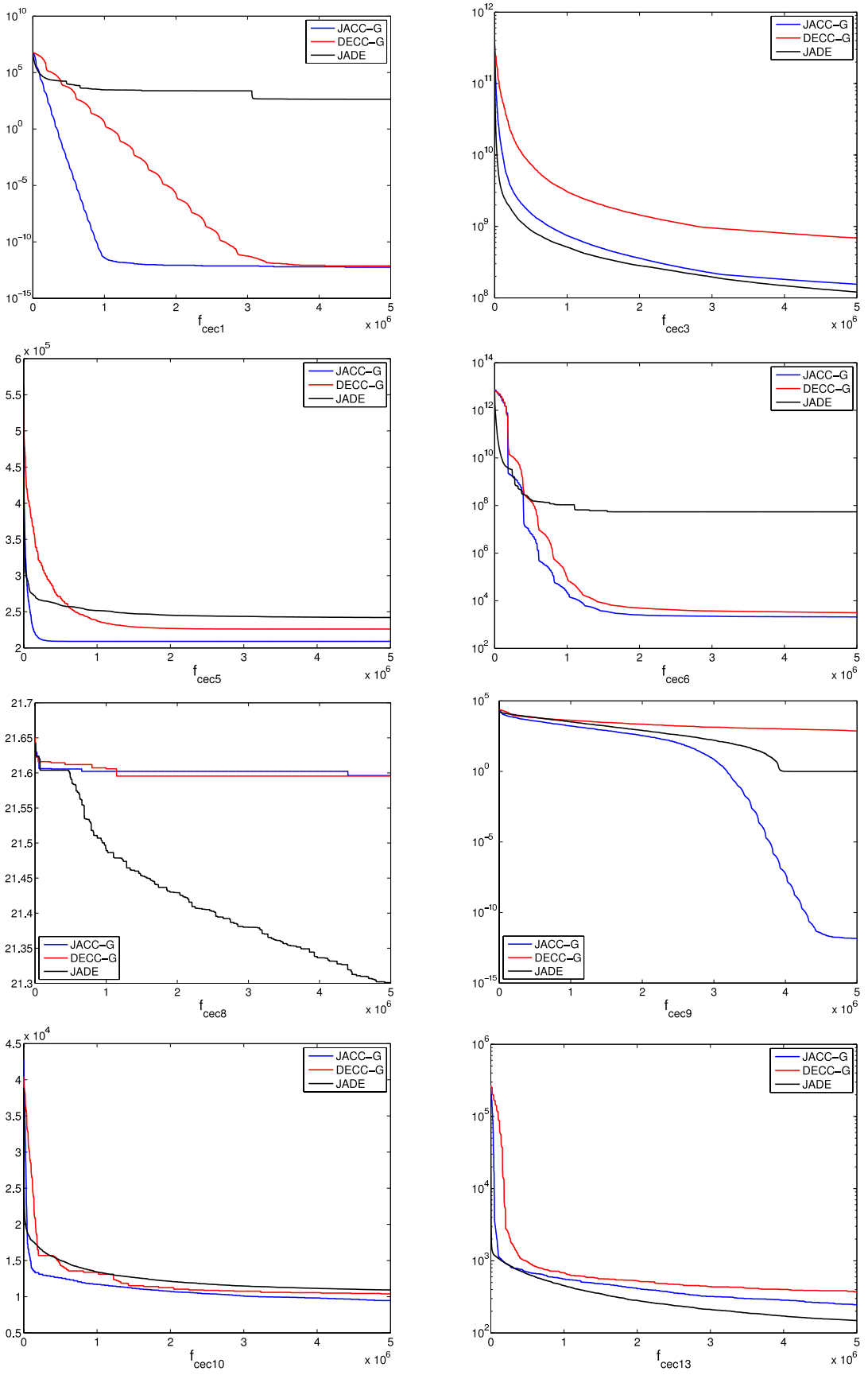

Fig. 2. The evolution curves for CEC'2005 functions. The vertical axes show the distance to optimum and the horizontal axes show the number of FEs.

[19] J. Zhang, R. Subbu, and J. Lizzi, "MONACO - Multi-Objective National Airspace Collaborative Optimization," *submitted as a book chapter of Computational Intelligence in Expensive Optimization Problems*, 2008.

[20] J. Zhang, V. Avasarala, A. C. Sanderson, and T. Mullen, "Differential evolution for discrete optimization: an experimental study on combinatorial auction problems," in *Proceeding of IEEE World Congress on Computational Intelligence*, 2008, pp. 2794–2800.

[21] X. Yao, Y. Liu, and G. Lin, "Evolutionary Programming Made Faster," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 82–102, 1999.

[22] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. P. Chen, A. Auger, and S. Tiwari, "Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization," *Technical Report, Nanyang Technological University, Singapore, http://www.ntu.edu.sg/home/EPNSugan*, 2005.