International Journal of Control and Automation
Vol.8, No.8 (2015), pp.261-272
http://dx.doi.org/10.14257/ijca.2015.8.8.26

Self-adaptive Based Cooperative Coevolutionary Algorithm for Large-scale Numerical Optimization

Qianli Zhang[1], Yu Xue[2], Xueliang Zhao[1,3], Xiangang Shang[3] and Qiqiang Li[1*]

[1]*School of Control Science and Engineering, Shandong University, Jinan 251000, P. R. China*
[2]*School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing 210044, P. R. China*
[3]*College of Information and Engineering, Taishan Medical University, Taian, 271016, P. R. China*
** Corresponding Author e-mail: qqli@sdu.edu.cn.*

Abstract

The scalability performance of the traditional evolutionary algorithms (EAs) deteriorates rapidly as the dimensionality of the optimization problems increases. Therefore, cooperative coevolutionary (CC) framework is proposed to overcome the defect. Different from existing CC algorithms, a novel self-adaptive based cooperative coevolutionary (SaCC) algorithm is presented in this paper. The SaCC employs three algorithms which with self-adaptive mechanism as sub-algorithms. The focus of this paper is on investigating two different cooperative coevolutionary manners. In the first manner, SaCC executes its sub-algorithms in parallel during evolve process and the corresponding algorithm is termed as SaCC-M_1. In the second manner, SaCC executes its sub-algorithms in serial and the corresponding algorithm is termed as SaCC-M_2. 26 test functions with 1000 dimensionalities are employed to verify the validity of SaCC-M_1 and SaCC-M_2. Experiment results demonstrate that SaCC-M_2 outperforms its sub-algorithms and SaCC-M_1. Besides, the results indicate that serial manner is another simple yet efficient manner for CC algorithms to solve large-scale global optimization problems.

Keywords: *Self-adaptive, cooperative coevolution, optimization algorithm, large-scale, optimization*

1. Introduction

Since the success of the evolutionary algorithm (EA) in solving one specific low dimensional optimization problem mainly depends on the selection of the suitable candidate solution generation strategy (CSGS) and its parameters, scientists and engineers usually employ a trial-and-error scheme to find the suitable CSGS and its parameters for an algorithm to ensure the success of the algorithm in solving a real-world optimization problem. However, it is time consuming to use the trial-and-error scheme to find the most suitable CSGS. Moreover, the trial-and-error scheme is infeasible sometimes due to the fitness landscape of the practical scientific and engineering problems can't be analyzed beforehand and it is difficult to accurately verify the characteristic of the CSGS. This prompted the researchers to introduce the self-adaptive mechanism into the EAs. For example, Qin *et al*. proposed a self-adaptive differential evolution (SaDE) algorithm in [1]. Successively, SaDE was developed in [2]. Besides, a self-adaptive learning based particle swarm optimization (SLPSO) [3] is proposed by Wang *et al*. to improve the performance of particle swarm optimization (PSO) in dealing with diverse problems with different characteristics. Moreover, a self-adaptive learning based immune algorithm (SALIA) is proposed in [4] to improve the performance of immune based algorithms

ISSN: 2005-4297 IJCA

(IBAs) in tackling with diverse problems especially complex multi-modal and ill-conditioned problems. Self-adaptive search method is effective and efficient in improving the performance of the evolutionary algorithms which has been demonstrated by many relevant research works [5-8].

Reference [9] demonstrates that the scalability of EAs is unsatisfactory, the performance of these algorithms deteriorates rapidly as the dimension of the search space increases. It is important to point out that, although the universality of the self-adaptive algorithms have been enhanced by the self-adaptive mechanism, they also have the "curse of dimensionality" problem [10] when the dimensionality of the search space increases. Global optimization of high-dimensional problems is a major challenge to the research community of EAs. The weakness of randomization-based EAs in searching high-dimensional spaces is demonstrated in [11]. Therefore, to enhance the scalability performance of the self-adaptive algorithms, the algorithm structure should be developed.

Cooperative coevolutionary (CC) is a new algorithm designing model and has been proved to be efficiently in solving large and complex problems [9, 12]. In this paper, we attempt to investigate the different algorithm portfolios manners for the SaDE, SLPSO and SALIA to enhance the scalability of self-adaptive algorithms and a new self-adaptive based cooperative coevolutionary algorithm is proposed.

2. Previous Work Related to SaCC

Different CSGS perform differently. This motivates the researchers proposed some self-adaptive based EAs. The most important components of the self-adaptive based algorithms are the strategy pool and the self-adaptive mechanism. The strategy pool and the self-adaptive mechanism of SaDE, SLPSO and SALIA are described as follows:

2.1. Strategy Pool

2.1.1. The four CSGSes employed by SaDE

The four CSGS employed by SaDE are described as follows [2]:

(1) DE/rand/1/bin:

$$u_{i,j} = \begin{cases} x_{r_1,j} + F \cdot (x_{r_2,j} - x_{r_3,j}), \text{ if } rand[0,1) < CR \text{ or } j = j_{rand} \\ x_{i,j}, \qquad \text{otherwise} \end{cases} \tag{1}$$

(2) DE/rand-to-best/2/bin:

$$u_{i,j} = \begin{cases} x_{i,j} + F \cdot (x_{best,j} - x_{i,j}) + F \cdot (x_{r_1,j} - x_{r_2,j}) + F \cdot (x_{r_3,j} - x_{r_4,j}), \text{ if } rand[0,1) < CR \text{ or } j = j_{rand} \\ x_{i,j}, \qquad \text{otherwise} \end{cases} \tag{2}$$

(3) DE/rand/2/bin:

$$u_{i,j} = \begin{cases} x_{r_1,j} + F \cdot (x_{r_2,j} - x_{r_3,j}) + F \cdot (x_{r_4,j} - x_{r_5,j}), \text{ if } rand[0,1) < CR \text{ or } j = j_{rand} \\ x_{i,j}, \qquad \text{otherwise} \end{cases} \tag{3}$$

(4) DE/current-to-rand/1:

$$\boldsymbol{U}_{i,G} = \boldsymbol{X}_{i,G} + k \cdot (\boldsymbol{X}_{r_1,G} - \boldsymbol{X}_{i,G}) + F \cdot (\boldsymbol{X}_{r_2,G} - \boldsymbol{X}_{r_3,G}) \tag{4}$$

where the first CSGS and the second CSGS are commonly used in many DE literatures. The third CSGS has better exploration capability while the fourth CSGS is efficient for rotated problems.

2.1.2. The Four CSGS Employed by SLPSO

The difference based velocity updating strategy (DbV), the estimation based velocity updating strategy (EbV), the comprehensive learning PSO (CLPSL) [13] strategy and the PSO-CL-pbest strategy, which are employed by SLPSO, are described as follows [3]:
(1) Difference based velocity updating strategy

$$Viad_i^d \leftarrow X_k^d - X_j^d \tag{5}$$

$$c = N(0.5,0.2) \tag{6}$$

$$V_i^d \leftarrow c \times Viad_i^d + c \times (pbest_i^d - X_i^d) \tag{7}$$

(2) Estimation-based velocity updating strategy

$$c = \frac{(D-1)N(0,1)}{D} + \frac{C(0,1)}{D} \tag{8}$$

$$V_i^d \leftarrow (mean_i^d - X_i^d) + \frac{c}{\sqrt{3}} \sqrt{(pbest_i^d - mean_i^d)^2 + (X_i^d - mean_i^d)^2 + (X_k^d - mean_i^d)^2} \tag{9}$$

(3) CLPSO strategy

$$V_i^d \leftarrow w \times V_i^d + c \times rand_i^d \times (pbest_{fi(d)}^d - X_i^d) \tag{10}$$

(4) PSO-CL-pbest strategy

$$V_i^d \leftarrow w \times V_i^d + 0.5 \times c \times rand_i \times (pbest_{fi(d)}^d - X_i^d + pbest_i^d - X_i^d) \tag{11}$$

2.1.3. The Four CSGS Employed by SALIA

The four CSGS used in SALIA are described as follows [4]:
(1) Modified DE/rand/1 mutation strategy (Mr1)

$$F = \mathrm{N}(0.5, 0.2) \tag{12}$$

$$\boldsymbol{A}_i^{'} \leftarrow \boldsymbol{A}_{r_1} + F \times (\boldsymbol{A}_{r_2} - \boldsymbol{A}_{r_3}) \tag{13}$$

(2) Modified DE/rand/2 mutation strategy (Mr2)

$$\boldsymbol{A}_i^{'} \leftarrow \boldsymbol{A}_{r_1} + rand(j) \times (\boldsymbol{A}_{r_2} - \boldsymbol{A}_{r_3}) + F \times (\boldsymbol{A}_{r_4} - \boldsymbol{A}_{r_5}) \tag{14}$$

(3) Modified DE/current-to-rand/1 mutation strategy (Mcr1)

$$F = U(0.6,1) \tag{15}$$

$$\boldsymbol{A}_i^{'} \leftarrow \boldsymbol{A}_i + rand(j) \times (\boldsymbol{A}_{r_1} - \boldsymbol{A}_{r_2}) + F \times (\boldsymbol{A}_{r_3} - \boldsymbol{A}_{r_4}) \tag{16}$$

(4) Immune secondary response based mutation strategy (ISR)

$$c = U(0,2) \tag{17}$$

$$\boldsymbol{A}_i^{'} \leftarrow \boldsymbol{A}_i^{'} + c \times (\boldsymbol{A}_{r_1} - \boldsymbol{A}_i) \tag{18}$$

After $\boldsymbol{A}_i^{'}$ is generated in Eqs. (12)~(18), the following Eqs. (19) and (20) are used to generate new solution, respectively.

$$\boldsymbol{A}_i^d = \begin{cases} \boldsymbol{A}_i^{'d}, & \text{if } d \in R(D) \\ \boldsymbol{A}_i^d, & \text{otherwise} \end{cases} \tag{19}$$

$$\boldsymbol{A}_i^d = \begin{cases} \min(2l_i^d - \boldsymbol{A}_i^d, u_i^d), & \text{if } \boldsymbol{A}_i^d < l_i^d \\ \max(2u_i^d - \boldsymbol{A}_i^d, l_i^d), & \text{if } \boldsymbol{A}_i^d > u_i^d \end{cases} \tag{20}$$

2.2. Self-adaptive Mechanism

2.2.1. The Self-adaptive Mechanism used in SaDE

Reference [2] uses $\mathrm{ns}_{k,G}$ (k=1,…,4) to record the number of solutions generated by the kth strategy and successfully enter the next generation in generation G while the number of solution generated by the kth strategy and discarded in the next generation is recorded by $\mathrm{nf}_{k,G}$. After learning period (LP) generations, the following Eqs. are used to calculate the *SSP*:

$$S_{k,G} = \left(\sum_{g=G-LP}^{G-1} \mathrm{ns}_{k,g}\right) \Big/ \left(\sum_{g=G-LP}^{G-1} \mathrm{ns}_{k,g} + \sum_{g=G-LP}^{G-1} \mathrm{nf}_{k,g}\right) + \varepsilon \tag{21}$$

$$p_{k,G} = S_{k,G} \Big/ \sum_{k=1}^{K} S_{k,G} \tag{22}$$

where $K = 4$, p_k is the *SSP* of the kth strategy.

2.2.2. The Self-Adaptive Mechanism used in SLPSO

In SLPSO, the idea of self-adaptive framework is similar to that in [2], but a learning rate α is used to control the updating speed of the *SSP* during the evolution procedure. Moreover, the *SSP* is updated based on the feedback of solution quality. The strategy selection probabilities $proSTR_{ij}$ and the accumulators of the four strategies S_{ij} (i=1,2,…,ps; j=1,…,4) is initially assigned to 0. Then, the ith best individual is assigned a weight $w_i = \log(ps - j + 1)/(\log(1) + \cdots + \log(ps))$. Finally, the weight is added to accumulator S_{ij}, where j is the corresponding updating strategy. After a fixed number of generations GS, the *SSP* is updated as follows [3]:

$$proSTR_{ij}^{'} = (1-\alpha)\, proSTR_{ij} + \alpha \cdot (S_{ij}/GS) \tag{23}$$

$$proSTR_{ij} = proSTR_{ij}^{'} \Big/ \sum_{j=1}^{4} proSTR_{ij}^{'} \tag{24}$$

2.2.3. Self-adaptive Mechanism used in SALIA

In SALIA, the fundamental idea of updating the *SSP* is similar to that in [2] and [3]. The main difference is that, [2] updates the *SSP* according to the effectiveness of the strategies only, [3] according to the fitness of the individuals only, whereas [4] accroding to both of the effectiveness of the strategies and the fitness of the individuals.

In SALIA, Vector $\boldsymbol{S}_i = [s_{i1}, s_{i2}, ..., s_{iM}]$ (i=1,2,…,ps, j=1,2,…,M, M is the number of CSGS) denotes the *SSP* of the i-th individual. Besides, a flag vector $\boldsymbol{f}_i = [f_{i1}, f_{i2}, ..., f_{iM}]$ is employed to record weather the individual is improved. f_{ij} ($j = 1, 2, ..., M$) is initially assigned to 0, $f_{ij} = 1$ if the individual is improved by the j-th strategy, otherwise $f_{ij} = 0$. s_{ij} is initially assigned to 0.25 since SALIA employs four CSGS. Moreover, a temporary

Vector $\boldsymbol{t}_i = [t_{i1}, t_{i2}, ..., t_{iM}]$ is used in SALIA. t_{ij} is initially assigned to 0.25 too. At each generation t_{ij} is calculated as follows:

$$t_{ij} = \begin{cases} t_{ij} + m_i \times w_i, \text{if } f_{ij} = 1 \\ t_{ij}, \qquad \text{otherwise} \end{cases} \tag{25}$$

$$t_{ij} = t_{ij} \Big/ \sum_{j=1}^{M} t_{ij} \tag{26}$$

$$s_{ij} = t_{ij} \tag{27}$$

where m_i is a flag integer, $m_i = 1$ if the fitness of the new individual is better than the original one, otherwise $m_i = 0.1$. w_i is the weight value of the *i*-th individual, it is calculated as follows:

$$(ps - s_i^{'} + 1) \Big/ \sum_{i=1}^{ps} (ps - s_i^{'} + 1) \tag{28}$$

where $s_i^{'}$ is the sorting sequence number of the *i*-th individual.

After a fixed *LP* generations, the following equation is employed to update the *SSP*:

$$s_{ij} = \frac{1}{N} \times \sum_{i=1}^{N} t_{ij} \tag{29}$$

Then, s_{ij} is normalized as follows:

$$s_{ij} = s_{ij} \Big/ \sum_{j=1}^{M} s_{ij} \tag{30}$$

3. Self-adaptive based Cooperative Coevolutionary Algorithms

The core idea behind the proposed SaCC algorithm is elucidated as follows: SaDE, SLPSO and SALIA are used as sub-algorithms and the whole population is divided into three subpopulations. Relevant literatures have shown that the advantages of hyper-search are mostly credited to the synergy between constituent algorithms [14-16]. Thus, the work of this paper mainly focuses on investigating the efficient algorithm portfolio manners for the sub-algorithms.

Reference [14] indicates that parallel is an efficient manner for low-level heuristics constitute to hyper-heuristics and combining several algorithms is advantageous over using a single one. Moreover, CC has been proved to be efficiently in solving large and complex problems [9, 12]. Motivated by these works, we developed the first algorithm portfolio manner (SaCC-M_1). In SaCC-M_1, the information exchange manner as designed as follows: In every generation, the worst individual of the sub-population will be replaced by the best individual of the current whole population if the sub-population does not include the best individual of the current whole population [6]. The framework of SaCC-M_1 is presented as follows:

Algorithm 1. SaCC-M_1

1 Initialize the output variables, the number of fitness evaluation, the max number of fitness evaluation (NFE), and initialize parameters of SaDE, SLPSO and SALIA;
2 WHILE stopping criterion is not satisfied DO:
3 Execute SaDE only one generation on sub-population 1;
4 Update necessary information of SaDE;
5 Execute SLPSO only one generation on sub-population 2;
6 Update necessary information of SLPSO;
7 Execute SALIA only one generation on sub-population 3;
8 Update necessary information of SALIA;
9 Find the best solution among the sub-populations and replace the worst solution of the

sub-population (s) which do not include the best solution;
10 END WHILE
11 Output the results.

In this paper, we have designed another algorithm portfolio manner (SaCC-M_2), without modifying SaDE, SLPSO and SALIA, we make them execute on the same population in serial manner, in other words, SaDE, SLPSO and SALIA are executed successively. Our preliminary experimental results demonstrate that the difference is little between executing the sub-algorithms in order of randomly or successively. Thus, in SaCC-M_2, we randomly select the sub-algorithm to evolve the population. The framework of SaCC-M_2 is described as follows:

Algorithm 2. SaCC-M_2

1 Initialize the output variables, the number of fitness evaluation, the max number of fitness evaluation (NFE), and initialize the public population, public local best population;
2 WHILE stopping criterion is not satisfied DO:
3 Select one sub-algorithm from SaDE, SLPSO and SALIA randomly;
4 Give the public population and the public local best population to the selected algorithm;
5 Execute the selected algorithm on the public population;
6 Update relevant information of the selected algorithm;
7 Give the evolved population and the updated local best population back to the public population and public local best population;
8 END WHILE
9 Output the results.

4. Experimental Study and Results

4.1. Test Functions and Parameter Settings

The test functions employed in this paper are similar to that used in [3, 4]. The difference is that the vector $\boldsymbol{o}$ in $\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{o}$ is updated and the product subcomponent is deleted from the function Shifted Schwefel 2.22 because the dimensions of the functions are extended to 1000. The dimension sizes of the 26 test functions were set to 1000. The number of function evaluations (NFE) was set to 300000 as the termination criterion for all the algorithms. 30 times independent experiments were conducted for each algorithm on each test function. Then, the mean value (*mean*) and standard deviation (*std*) of the results obtained by the 30 times run are calculated for measuring the performance of each algorithm. Since SaDE, SLPSO and SALIA are employed as sub-populations in SaCC, so their main parameter settings are list as follows:

Algorithm	Parameter settings
SaDE	Population is set to 1000; $F = N(0.5,0.3)$; initial $SSP = [0.25,0.25,0.25,0.25]$, $LP = 3$.
SLPSO	Population is set to 1000; $c = normrand(0.5,0.2)$ in the fist CSGS, $c = 1.49445$ in the third and the fourth CSGS ; $w = 0.9 - 0.5 \times (Cur\_Gen / Max\_Gen)$ where Cur_Gen is the number of current generations, Max_Gen is the number of maximum generations; $\alpha = 1/10$; initial $SSP = [0.25,0.25,0.25,0.25]$, $LP = 3$.
SALIA	Population is set to 1000; $cloneScale = 100$; $cloneSize = 100$; $pd = 0.5$; initial $SSP = [0.25,0.25,0.25,0.25]$, $LP = 3$.

Note: Detailed descriptions of the parameter can be found in Refs. [2-4].

4.2. Experimental Results

We have conducted experiments to test the performance of SaDE, SLPSO, SALIA, SaCC-M_1 and SaCC-M_2 on the 26 test functions with 1000 dimensions. Table 1 and Table 2 present the numerical experimental results gathered by allowing all of the algorithms tested to run for a fixed number of function evaluations, *i.e.*, 300000.

Table 1. The Mean and Standard Deviation of the Solution Quality Obtained by SaDE, SLPSO, SALIA, SaCC-M1 and SaCC-M2 on f_1 to f_{12} (D=1000)

Algorithm	f_1		f_2		f_3	
	Mean	Std	Mean	Std	Mean	Std
SaDE	2.02E+006	8.38E+005	6.59E+006	1.22E+005	9.77E+002	8.08E+000
SLPSO	1.56E+006	6.26E+003	1.70E+007	1.03E+005	1.47E+003	1.45E+003
SALIA	1.36E+006	2.63E+004	7.28E+006	8.68E+005	1.80E+003	5.47E+000
SaCC-M_1	2.58E-004	8.14E-005	3.07E-005	4.07E-005	**2.38E-004**	2.44E-004
SaCC-M_2	**7.29E-005**	1.76E-005	**2.19E-005**	2.53E-005	5.86E-004	4.70E-004
Algorithm	f_4		f_5		f_6	
	Mean	Std	Mean	Std	Mean	Std
SaDE	2.53E+006	7.87E+005	1.38E+007	1.66E+006	1.61E+001	9.37E-002
SLPSO	**1.67E+006**	8.38E+003	6.80E+006	5.50E+005	1.87E+001	3.47E-002
SALIA	1.17E+007	9.40E+005	7.47E+009	9.72E+008	1.93E+001	1.38E-001
SaCC-M_1	4.11E+006	3.40E+004	**0.00E+000**	0.00E+000	9.09E-003	4.94E-003
SaCC-M_2	4.96E+006	2.77E+005	**0.00E+000**	0.00E+000	**5.13E-003**	1.56E-003
Algorithm	f_7		f_8		f_9	
	Mean	Std	Mean	Std	Mean	Std
SaDE	5.51E+004	5.25E+003	1.55E+004	1.69E+002	1.47E+004	5.85E+002
SLPSO	3.61E+004	2.93E+002	1.20E+004	1.84E+002	1.24E+004	2.88E+002
SALIA	5.06E+004	5.12E+002	1.30E+004	2.75E+002	1.31E+004	2.71E+002
SaCC-M_1	**1.91E+001**	3.33E-004	1.19E-004	1.58E-004	1.48E-005	2.08E-005
SaCC-M_2	**1.91E+001**	4.53E-004	**4.45E-005**	6.29E-005	**4.34E-006**	5.57E-006
Algorithm	f_{10}		f_{11}		f_{12}	
	Mean	Std	Mean	Std	Mean	Std
SaDE	9.57E-001	5.81E-002	5.16E+002	1.20E+002	3.28E+015	1.31E+015
SLPSO	9.32E-001	8.57E-002	1.95E+002	1.87E+001	7.91E+015	1.97E+014
SALIA	5.48E+006	1.20E+006	6.35E+007	2.24E+007	7.27E+018	3.63E+017
SaCC-M_1	**1.17E-005**	1.40E-005	1.33E-005	1.73E-005	**0.00E+000**	0.00E+000
SaCC-M_2	5.15E-005	6.89E-005	**6.93E-006**	2.99E-006	**0.00E+000**	0.00E+000

The best means obtained by the algorithms within 300000 NFE on the functions are typed in bold in Table 1 and Table 2. The results presented in Table 1 and Table 2 can be summarized as: (1) By comparing SaCC-M_1 and SaCC-M_2 with SaDE, SLPSO, SALIA, it can be seen that SaCC-M_1 and SaCC-M_2 significantly outperform SaDE, SLPSO and SALIA on most functions in terms of solution quality. (2) The performance of SaDE, SLPSO and SALIA are similar, and their performance relatively rapidly deteriorates. The performance of SaCC-M_2 is slightly more efficient than that of SaCC-M_1. (3) The results show that the performance of SaDE is better that other algorithms on 3 functions (f_{13}, f_{14}, f_{19}) in terms of mean value, SLPSO and SALIA outperform the other algorithms on 2 functions (f_4, f_{24}) and 1 function (f_{18}), respectively, whereas SaCC-M_1 and SaCC-M_2 outperform the other algorithms on 7 functions (f_3, f_{10}, f_{12}, f_{16}, f_{17}, f_{20}, f_{25}) and 10 (f_1, f_2, f_6, f_8, f_9, f_{11}, f_{12}, f_{15}, f_{22}, f_{26}) functions, respectively. Moreover, SaCC-M_1 and SaCC-M_2 can obtain the better solution quality on functions f_5, f_7 and f_{21} simultaneously. SLPSO, SaCC-M_1 and SaCC-M_2 can obtain the better solution quality on f_{23} simultaneously.

We can get some conclusions from the results presented in Table 1 and Table 2. First, the performance of SaDE, SLPSO and SALIA decreases significantly with the dimensionality of the test function increases. Second, it demonstrated that Manner 1 and Manner 2 are two efficient manners to enhance the performance of self-adaptive algorithms, and the constituent algorithms in Manner 1 and Manner 2 are complementary on most functions. Third, it seems that Manner 2 is more efficient than Manner 1 on the test functions. So, SaCC-M_2 is employed as SaCC finally.

Figure 1 shows a plot of the performance of the various algorithms over NFE on some classical functions. It can be seen that the convergence speed of SaCC-M_1 and SaCC-M_2 is faster than the other algorithms and they can obtain the optimal solutions on f_5, f_{12} while the other algorithms trapped in local search. SaCC-M_1 and SaCC-M_2 with better convergence performance and can obtain relatively better solutions on man functions. SaCC-M_1 and SaCC-M_2 can obtain better solutions but with good

Table 2. The Mean and Standard Deviation of the Solution Quality Obtained by SaDE, SLPSO, SALIA, SaCC-M1 and SaCC-M2 on f_{13} to f_{26} (D=1000)

Algorithm	f_{13}		f_{14}		f_{15}	
	Mean	Std	Mean	Std	Mean	Std
SaDE	**3.25E+004**	4.66E+002	**1.11E+007**	1.02E+006	6.58E+006	3.21E+005
SLPSO	6.47E+004	1.73E+003	5.84E+007	5.70E+005	1.96E+007	1.11E+006
SALIA	6.46E+004	4.03E+003	3.45E+007	2.38E+006	1.33E+007	1.68E+006
SaCC-M_1	1.16E+005	4.54E+003	1.63E+008	3.63E+006	5.74E-004	7.31E-004
SaCC-M_2	1.04E+005	9.14E+003	1.40E+008	2.10E+007	**2.60E-004**	3.68E-004
Algorithm	f_{16}		f_{17}		f_{18}	
	Mean	Std	Mean	Std	Mean	Std
SaDE	6.36E+005	2.00E+004	7.23E+005	8.16E+004	2.54E+007	2.43E+007
SLPSO	1.56E+006	3.13E+004	1.66E+006	2.95E+004	1.92E+007	2.12E+005
SALIA	1.37E+006	3.71E+004	1.41E+006	6.07E+004	**1.73E+007**	1.59E+005
SaCC-M_1	**1.02E-005**	8.60E-006	**1.05E-001**	3.46E-002	3.90E+007	1.60E+006
SaCC-M_2	1.79E-005	2.01E-005	1.31E-001	5.83E-002	3.70E+007	2.03E+006
Algorithm	f_{19}		f_{20}		f_{21}	
	Mean	Std	Mean	Std	Mean	Std
SaDE	**1.50E+019**	1.37E+019	9.68E+000	2.63E-001	1.53E+007	5.28E+006
SLPSO	4.27E+020	8.00E+019	7.63E+000	5.91E-002	6.84E+006	6.99E+005
SALIA	7.06E+019	9.70E+018	4.79E+001	3.52E+000	8.48E+009	1.52E+009
SaCC-M_1	8.14E+021	3.81E+021	**2.35E-003**	2.59E-003	**0.00E+000**	0.00E+000
SaCC-M_2	4.46E+021	8.98E+018	6.53E-003	5.32E-003	**0.00E+000**	0.00E+000
Algorithm	f_{22}		f_{23}		f_{24}	
	Mean	Std	Mean	Std	Mean	Std
SaDE	1.64E+001	4.45E-001	5.35E+004	7.85E+002	1.49E+004	7.15E+002
SLPSO	1.88E+001	7.74E-002	**1.91E+001**	3.50E-004	**3.12E-005**	1.73E-005
SALIA	1.93E+001	2.35E-002	5.06E+004	8.18E+002	1.31E+004	1.73E+002
SaCC-M_1	1.20E-002	7.83E-003	**1.91E+001**	4.69E-006	5.98E-005	6.12E-005
SaCC-M_2	**1.16E-003**	1.56E-003	**1.91E+001**	2.90E-003	8.35E-005	6.74E-005

Algorithm	f_{25}		f_{26}	
	Mean	Std	Mean	Std
SaDE	7.70E+006	1.77E+006	2.22E+008	2.52E+008
SLPSO	1.26E-004	1.56E-004	2.46E-001	3.39E-001
SALIA	1.46E+007	2.60E+006	3.90E+010	1.50E+009
SaCC-M_1	**1.85E-005**	2.30E-005	5.29E-002	2.06E-002
SaCC-M_2	9.39E-005	1.32E-004	**2.50E-002**	1.58E-003

convergence speed on f_{14} and f_{18}. Overall, Figure 1 shows that the SaCC-M_1 and SaCC-M_2 with better convergence speed and they can obtain better solutions on most test functions whereas perform worse on very few functions.

5. Conclusion

Through comparing the results of SaCC and its constituent algorithms we have known that SaCC obviously outperforms its constituent algorithm on the test functions. Therefore, combining algorithm is an efficient manner to significantly improve the scalability performance of the evolutionary algorithms on large-scale optimization problems. In our further research, more test functions would be employed and more algorithms would be compared.

Acknowledgements

We would like to thank the National Natural Science Foundation of China (61403206), the Natural Science Foundation of Jiangsu Province (BK20141005), the Natural Science Foundation of the Jiangsu Higher Education Institutions of China (14KJB520025), the Priority Academic Program Development of Jiangsu Higher Education Institutions, the Research Foundation of Nanjing University of Information Science and Technology (2013x034) and the Student Innovation Training Program (201510300103).

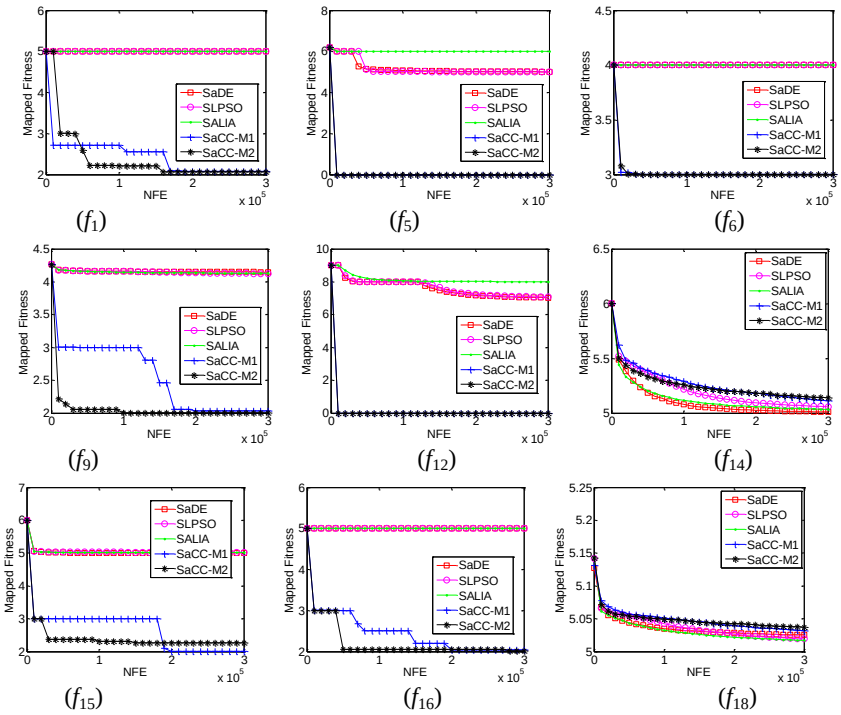

Figure 1. Fitness Change Curves of the Compared Algorithms on some Functions

References

[1] Qin, P. Suganthan, Self-adaptive differential evolution algorithm for numerical optimization, in: IEEE Congress on Evolutionary Computation, IEEE, Edinburgh, Scotland, **2005**, pp. 1785-1791
[2] A. K. Qin, V.L. Huang, P.N. Suganthan, Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization, Ieee Transactions on Evolutionary Computation, 13 (**2009**) 398-417.
[3] Y. Wang, B. Li, T. Weise, J.Y. Wang, B. Yuan, Q.J. Tian, Self-adaptive learning based particle swarm optimization, Information Sciences, 181 (**2011**) 4515-4538.

[4] B. Xu, Y. Zhuang, Y. Xue, Z. Wang, Self-adaptive learning based immune algorithm, Journal of Central South University, 19 (**2012**) 1021-1031.
[5] Y. Xue, Y. Zhuang, T.Q. Ni, J. Ouyang, Z. Wang, Enhanced self-adaptive evolutionary algorithm for zation, J. Syst. Eng. Electron., 23 (**2012**) 921-928.
[6] ong, Y. Zhuang, B. Xu, An ensemble algorithm with self-adaptive learning techniques onal numerical optimization, Applied Mathematics and Computation, 231 (**2014**) 329-
[7] ang, T.Q. Ni, S.R. Ni, X.Z. Wen, Self-adaptive learning based discrete differential hm for solving CJWTA problem, J. Syst. Eng. Electron., 25 (**2014**) 59-68.
[8] Chiam, A.A. Mamun, C.K. Goh, Balancing exploration and exploitation with adaptive utionary multi-objective optimization, Eur. J. Oper. Res., 197 (**2009**) 701-713.
[9] Z. Y. Yang, K. Zhang, K. Tang, X. Yao, A.C. Sanderson, An adaptive coevolutionary differential evolution algorithm for large-scale optimization, in: IEEE Congress on Evolutionary Computation, IEEE, New York, **2009**, pp. 102-109.
[10] F. van den Bergh, A.P. Engelbrecht, A cooperative approach to particle swarm optimization, IEEE Transactions on Evolutionary Computation, 8 (**2004**) 225-239.
[11] W. Chu, X.G. Gao, S. Sorooshian, A new evolutionary search strategy for global optimization of high-dimensional problems, Information Sciences, 181 (**2011**) 4909-4927.
[12] Z. Y. Yang, K. Tanga, X. Yao, Large scale evolutionary optimization using cooperative coevolution, Information Sciences, 178 (**2008**) 2985-2999.
[13] J. J. Liang, A.K. Qin, P.N. Suganthan, S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE Transactions on Evolutionary Computation, 10 (2006) 281-295.
[14] F. Peng, K. Tang, G.L. Chen, X. Yao, Population-based algorithm portfolios for numerical optimization, IEEE Trans. Evol. Comput., 14 (**2010**) 782-800.
[15] E. K. Burke, G. Kendall, E. Soubeiga, A tabu-search hyperheuristic for timetabling and rostering, Journal of Heuristics, 9 (**2003**) 451-470.
[16] E. K. Burker, M. Hyde, G. Kendall, G. Ochoa, E. Ö zcan, J.R. Woodward, A classification of hyper-heuristic approaches, Handbook of Metaheuristics, 146 (**2010**) 449-468.

Authors

Qianli Zhang, College Student in the School of Control Science and Engineering, Shandong University. His current research interests include modeling and control, measurement and control technology.

Yu Xue (S'10- M'13) received the B.S. and M.S. degrees from Taishan medical college, in 2006, and China West Normal University, in 2009. He received the Ph.D. degree from Nanjing University of Aeronautics & Astronautics, China, in 2013. He is a lecturer in the School of Computer and Software, Nanjing University of Information Science and Technology. He is also a post-doctor of Nanjing University of Information Science and Technology. His research interests include Computational Intelligence, Internet of Things.

Xueliang Zhao received his B.Sc. degree in electromachanics from Jinan University in 1998, and M. Sc. in detection technology and automatic equipment from Shandong Universtiy in 2006.Since 2011, he has been a Ph.D. degree candidate in control scicence and engineering from Shandong Universtiy. He is a lecturer at Taishan Medical University. His current research interests include modeling and control of PZT, applications of nanopositioning system.

Xiangang Shang received his M.Sc. degree in Control engineering and control theory from Shandong University of Science and Technology in 2005. He is a University Lecturer at Taishan Medical University. His research interests include digital image processing, pattern recognition, machine learning, etc.

Li Qiqiang, born in 1964, is a professor in the School of Control Science and Engineering, Shandong University. He received a B.Sc. and M.Sc. in Industrial Automation from Shandong University of Technology in 1985 and 1991 respectively, and received a Ph.D. in Industrial Automation at Institute of Industrial Process Control of Zhejiang University in 1998. His areas of research interest include modeling, simulation, control and optimization of complex processes