Differential Evolution for High-Dimensional Function Optimization

Zhenyu Yang, Ke Tang and Xin Yao

***Abstract*— Most reported studies on differential evolution (DE) are obtained using low-dimensional problems, e.g., smaller than 100, which are relatively small for many real-world problems. In this paper we propose two new efficient DE variants, named DECC-I and DECC-II, for high-dimensional optimization (up to 1000 dimensions). The two algorithms are based on a cooperative coevolution framework incorporated with several novel strategies. The new strategies are mainly focus on problem decomposition and subcomponents cooperation. Experimental results have shown that these algorithms have superior performance on a set of widely used benchmark functions.**

I. Introduction

Differential evolution (DE) is a simple yet effective algorithm for global optimization [1]. It has shown superior performance in both widely used benchmark functions [2] and real-world applications [3], [4]. The crucial idea behind DE is a new scheme of mutation. DE executes its mutation by adding a weighted difference vector between two individuals to a third individual. And then the mutated individuals will do discrete crossover and greedy selection with corresponding individuals of last generation to produce offspring. There are three strategy parameters in DE, i.e., population size NP, crossover rate CR and scaling factor F. Many works have been done to study the suitable setting of these control parameters [5]. Zaharie has analyzed the relationship between these control parameters and population diversity [6]. Several DE schemes have been proposed based on different mutations [1]. Self-adaptive strategy has also been investigated to adapt these parameters [7], as well as different schemes of DE [8].

However, most analytical and experimental results on DE are obtained using low-dimensional problems, e.g., smaller than 100. The reported studies on the scalability of DE derivative algorithms are scarce. In contrast, other evolutionary algorithms such as evolutionary programming (EP), have been studied in the domain of high-dimensional problems (up to 1000-dimensions) [9]. In terms of optimizing high-dimensional problems, cooperative coevolution with the following divide-and-conquer strategy is the usual and effective choice:

1) Problem decomposition: Splitting the object vectors into some smaller subcomponents.
2) Optimize subcomponents: Evolve each subcomponent with a certain optimizer separately.

The authors 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. (emails: zhyuyang@mail.ustc.edu.cn, ketang@ustc.edu.cn, x.yao@cs.bham.ac.uk).

3) Cooperative combination: Combine all subcomponents to form the whole system.

We can conclude that problem decomposition, optimizer selection and subcomponents cooperation are the crucial issues for the cooperative coevolution framework. Cooperative coevolution architecture was firstly proposed by Potter for genetic algorithm, called CCGA [10], and had been successfully applied to other evolutionary algorihtms such as evolutionary programming, called FEPCC [9], evolution strategy, called CCES [11], and particle swarm optimization, called CPSO [12]. In the context of DE, the cooperative coevolution framework has also been introduced to propose the corresponding CCDE [13]. CCDE only utilized simple problem decomposition and subcomponents cooperation methods. Moreover, CCDE still use the original DE as subcomponent optimizer, while many more efficient DE variants have been proposed recently. We might get a better scratch line for optimizing high-dimensional problems with these advanced DE variants. Besides, CCDE only extended the problem domain up to 100 dimensions, which are still relatively small for many real-world problems.

In the present paper, we propose two new efficient DE variants, named DECC-I and DECC-II, for high-dimensional optimization (up to 1000 dimensions). Cooperative coevolution framework with several novel strategies are used in the two algorithms. We mainly focus on how to decompose and cooperate subcomponents more suitably, rather than applying the original straightforward framework to DE directly. We choose a relatively new and efficient DE variant SaNSDE as the basic subcomponent optimizer, but any other DE variant and even other EAs can be introduced into DECC easily.

II. Preliminaries

A. Differential Evolution (DE)

Individuals in DE are represented by D-dimensional vectors $\mathbf{x_i}, \forall i \in \{1, \cdots, NP\}$, where D is the number of optimization parameters and NP is the population size. According to the description by Storn and Price [1], the evolutionary operations of classical DE can be summarized as follow:

1) Mutation:

$$\mathbf{v_i} = \mathbf{x_{i_1}} + F \cdot (\mathbf{x_{i_2}} - \mathbf{x_{i_3}}) \tag{1}$$

with $i, i_1, i_2, i_3 \in [1, NP]$, are integers and mutually different, and $F > 0$, is a real constant factor to control the differential variation $\mathbf{d_i} = \mathbf{x_{i_2}} - \mathbf{x_{i_3}}$.

2) Crossover:

$$\mathbf{u_i}(j) = \begin{cases} \mathbf{v_i}(j), & if\ U_j(0,1) < CR \\ \mathbf{x_i}(j), & otherwise. \end{cases} \tag{2}$$

with $U_j(0,1)$ stands for the uniform random between 0 and 1, and $CR \in (0,1)$ is the crossover rate.

3) Selection:

$$\mathbf{x}'_\mathbf{i} = \begin{cases} \mathbf{u_i}, & if\ f(\mathbf{u_i}) < f(\mathbf{x_i}) \\ \mathbf{x_i}, & otherwise. \end{cases} \quad (3)$$

where $\mathbf{x}'_\mathbf{i}$ is the offspring of $\mathbf{x_i}$ for the next generation.

There are several variants of DE, but this DE scheme which can be classified using notation $DE/rand/1/bin$ is the most often used in practice [1], [8].

B. Differential Evolution with Neighborhood Search (NSDE)

Evolutionary programming (EP) [14], [15] is another major branch of evolutionary computation. Neighborhood search (NS) is a main strategy underpinning EP. Characteristics of several NS operators' have been investigated in EP literatures [15], [16]. Although DE might be similar to the evolutionary process in EP, it lacks relevant concept of neighborhood search. Based on the generalization of NS strategy, we have proposed a neighborhood search differential evolution (NSDE) in [17]. NSDE is the same as the DE described in Section II.A except for Eq. (1) which is replaced by the following:

$$\mathbf{v_i} = \mathbf{x_{i_1}} + \begin{cases} \mathbf{d_i} \cdot N(0.5, 0.5), & if\ U(0,1) < 0.5 \\ \mathbf{d_i} \cdot \delta, & otherwise. \end{cases} \quad (4)$$

where $\mathbf{d_i} = \mathbf{x_{i_2}} - \mathbf{x_{i_3}}$ is the differential variation, $N(0.5, 0.5)$ denotes a Gaussian random number with mean 0.5 and standard deviation 0.5, and δ denotes a Cauchy random variable with scale parameter $t = 1$.

The advantages of NS strategy in DE have been studied in [17]. Experimental results have shown that NSDE has significant advantages over conventional DE on a broad range of different benchmark functions. It has been found that NS operators will improve the diversity of DE's search step size and population. This feature will be beneficial to escaping from local optima when searching in environments without prior knowledge about what kind of search step size will be preferred.

C. Self-adaptive NSDE (SaNSDE)

As stated in [5], [8], the control parameters and learning strategies involved in DE are highly problem dependent. It might be time consuming to try through various strategies and fine-tune the corresponding parameters for a specific task. Hence a self-adaptive DE algorithm (SaDE) [8] has been proposed to solve this dilemma. In SaDE, two DE's learning strategies are selected as candidates due to their good performance. Two out three critical parameters, crossover rate CR and scaling factor F, are adaptively changed instead of taking fixed empirical settings. The suitable learning strategy and parameter settings are gradually self-adapted according to the learning experience during evolution. The performance of SaDE is reported on the set of 25 benchmark functions provided by CEC2005 special session [8].

Comparing to NSDE, SaDE has quite different emphasis on improving classical DE. SaDE pay special attention to self-adaption between different DE variants, while NSDE intends to mix search biases of different NS operators through the scaling factor F. Since they have relatively irrelated focuses, it's possible to merge SaDE and NSDE to produce a more considered self-adaptive NSDE (SaNSDE). We keep the SaNSDE same to NSDE except:

1) Introducing the self-adaptive mechanism between different DE variants of SaDE.
2) Following the strategy in SaDE to dynamically adapt the value of CR.
3) Using the same self-adaptive strategy in SaDE on adapt different variants to adapt F using Gaussian and Cauchy operators.

Due to the significant successes of NSDE and SaDE, SaNSDE is promising to be more powerful for function optimization.

III. Scaling Up Differential Evolution

A. Differential Evolution with Cooperative Coevolution

As suggested in [9], [10], [13], we still follow the cooperative coevolution framework in DE. The main approach is to decompose the problem into subcomponents and evolve these subcomponents cooperatively for a predefined number of *cycles*. Here one complete evolution of all subcomponents is called a *cycle*. Based on the idea, the main steps of differential evolution with cooperative coevolution (DECC) would be:

1) Decompose high-dimensional object vector into m low-dimensional subcomponents.
2) i = 1. Optimize subcomponent 1 with SaNSDE for a predefined number of fitness evaluations (FEs).
3) i = i + 1. Optimize subcomponent i with SaNSDE for a predefined number of FEs.
4) if $i \leq m$, go to step 3.
5) Execute a cooperative operation among all subcomponents.
6) Stop if the halting criterion is satisfied; otherwise go to step 2 for the next cycle.

Given SaNSDE as the basic subcomponent optimizer, we need to further state the problem decomposition and cooperation issues to form algorithms as follow.

B. Problem Decomposition by Grouping and Weighting

As we already know, DE is effective and efficient on low-dimensional (up to 100 dimensions) problems [1], [7]. Obviously, for high-dimensional (up to 1000 dimensions) problems we can split them into several groups of low-dimensional subcomponents, and then evolve each of these groups. For cooperation, we can apply a weight to each of these groups, and evolve the weight vector after each cycle with a certain optimizer (also DE in the paper). The key points of this strategy can be summarized as follows:

1) Decomposition: Group the n-dimensional object vector into m s-dimensional subcomponents randomly, where

assume $n = m * s$. Here "randomly" means each variable has the same chance to be assigned into any of the groups.

2) Cooperation: Apply a weight to each of the groups after each complete circle; Evolve the weight vectors for the best, the worst and a random members in current population.

The only parameter is the dimension of subcomponents s, which can be specified by users (usually between 30–100 dimensions). After combining with the basic process listed in Section III.A, we will immediately get a complete algorithm for high-dimensional problems. The algorithm will be denoted as DECC-I in this paper. The pseudocode of DECC-I is given in Fig. 1.

Algorithm III.1: DECC-I($s, cycles, FEs, wFEs$)

$pop(1:NP, 1:D) \leftarrow random\ population$
$wpop(1:NP, 1:D/s) \leftarrow define\ weight\ population$
$(best, best\_val) \leftarrow evalate(pop)$
$index(1:D) \leftarrow randperm(D)$
for $i \leftarrow 1$ **to** $cycles$
do
- **for** $j \leftarrow 1$ **to** D/s
  - **do**
    - $l \leftarrow (j-1) * s + 1$
    - $u \leftarrow j * s$
    - $subpop \leftarrow pop(:, index(l:u))$
    - $subpop \leftarrow SaNSDE(best, subpop, FEs)$
    - $wpop(:, j) \leftarrow random\ weight$
    - $pop(:, index(l:u)) \leftarrow subpop$
    - $(best, best\_val) \leftarrow evaluate(pop)$
- $(best, best\_index) \leftarrow findbest(pop)$
- $(rand, rand\_index) \leftarrow findrand(pop)$
- $(worst, worst\_index) \leftarrow findworst(pop)$
- $pop(best\_index, :) \leftarrow DE(best, wpop, wFEs)$
- $pop(rand\_index, :) \leftarrow DE(rand, wpop, wFEs)$
- $pop(worst\_index, :) \leftarrow DE(worst, wpop, wFEs)$
- $(best, best\_val) \leftarrow evaluate(pop)$

return $(best\_val)$

Fig. 1. The pseudocode of DECC-I.

C. Problem Decomposition by Evolving Random Subsets

In DECC-I, the structure of problem decomposition keeps fixed during evolutionary process. More generally, we can extend this problem decomposition idea to produce another algorithm as follows:

1) Select a subset with s variables randomly from the object vector to form a s-dimensional subcomponent.
2) Optimize the subset with SaNSDE for one cycle, while keeping the other variables constant.
3) Stop if the halting criterion is satisfied; otherwise go to step 1 for the next cycle.

Again, based on DE's scalability, we can set the parameter s between 30 to 100 dimensions. We denote the algorithm as DECC-II. The pseudocode of DECC-II is given in Fig. 2. Since the structure of subset changes dynamically, it's not easy to apply the weighting strategy in DECC-I to DECC-II. Although process of DECC-II is simple than DECC-I, it might be also promising for high-dimensional optimization.

Algorithm III.2: DECC-II($s, cycles, FEs$)

$pop(1:NP, 1:D) \leftarrow random\ population$
$(best, best\_val) \leftarrow evalate(pop)$
for $i \leftarrow 1$ **to** $cycles$
do
- $index(1:D) \leftarrow randperm(D)$
- $subpop \leftarrow pop(:, index(1:s))$
- $subpop \leftarrow SaNSDE(best, subpop, FEs)$
- $pop(:, index(1:s)) \leftarrow subpop$
- $(best, best\_val) \leftarrow evaluate(pop)$

return $(best\_val)$

Fig. 2. The pseudocode of DECC-II.

D. Discussions Toward Nonseparable Functions

The conventional one-dimension based problem decomposition methods have a reputation of inefficient on nonseparable functions [9], [10]. Here nonseparable means the object vector is consisted of interacting variables. It was always assumed that decomposition will not work on completely nonseparable functions, in which interaction exists between any two variables of the object vector. For these cases, we can only pursue a better near-optimum with decomposition based methods. However, it is more common and usual that interaction only exists in some subsets of all variables. If we happen to group these interacting variables together, we still have chance to find the optimum.

The proposed DECC-I and DECC-II have been designed to embody mechanisms that are more beneficial to nonseparable functions. Firstly, both DECC-I and DECC-II have extended the original one-dimensional decomposition methods. They optimize a subset of all variables at a time rather than only one variable. And this will increase the chance of optimizing some interacting variables together. Further, DECC-I has an additional weighting mechanisms to cooperate interacting variables of different groups, while DECC-II always changes the optimized subset dynamically to increase the probability of grouping interacting variables together. Although DECC-I and DECC-II might not consistently find the optimum for some functions, they do give a starting point for further refinements in the domain of high-dimensional optimization.

IV. Experimental Studies

A. Benchmark Functions

A new set of benchmark functions, which was provided by CEC2005 special session, was used in our experimental studies. It includes 25 functions with different complexity [18]. Functions $f_1 - f_5$ are unimodal while the remaining 20

functions are multimodal. All of these functions are scalable. Detailed descriptions of these functions can be found in [18]. Functions $f_{15} - f_{25}$ are hybrid composition functions, which are very time consuming for fitness evaluation. So we only used the first 14 functions of the set. Many of them are the shifted, rotated, expanded or combined variants of classical functions listed in appendix of [15]. Some of these changes cause these functions more insensitive to simple search tricks. Other changes such as rotation will transfer them from separable to nonseparable, which might be a particular challenge for the cooperative coevolution framework. Hence, this suite of benchmark functions are more applicable for the experimental evaluations on DECC-I and DECC-II.

B. SaNSDE on Low-dimensional Problems

First we will evaluate SaNSDE's effectiveness for low-dimensional problems (30 dimensions in our experiment). The algorithms used for comparison are classical DE and NSDE. The experimental results of them are taken from [17], [19]. The average results of 25 independent runs are summarized in Tables 1 and 2.

TABLE I
COMPARISON BETWEEN SANSDE, NSDE AND DE ON UNIMODAL FUNCTIONS $f_1 - f_5$, WITH DIMENSION $D = 30$. ALL RESULTS HAVE BEEN AVERAGED OVER 25 INDEPENDENT RUNS.

Test Func	SaNSDE Mean	NSDE Mean	DE Mean	SaNSDE t-test	NSDE t-test
f_1	0.00e+00	0.00e+00	0.00e+00	0	0
f_2	1.66e-13	1.27e-06	3.33e-02	-3.40†	-3.40†
f_3	9.18e+04	5.19e+05	6.92e+05	-14.14†	-2.64†
f_4	7.15e-05	5.28e+01	1.52e+01	-4.20†	1.48
f_5	2.24e+00	1.15e+03	1.70e+02	-4.55†	10.54†

† The value of t-test is significant at $\alpha = 0.05$ by a two-tailed test.

For unimodal functions $f_1 - f_5$, NSDE has comparable performance with DE, while SaNSDE is better than both of them. SaNSDE outperforms DE significantly on almost all functions except the three algorithms have exactly same result on the shifted Sphere function f_1. The success of SaNSDE may because the introduced self-adaptive and neighborhood search strategies both worked well.

TABLE II
COMPARISON BETWEEN SANSDE, NSDE AND DE ON MULTIMODAL FUNCTIONS $f_6 - f_{14}$, WITH DIMENSION $D = 30$. ALL RESULTS HAVE BEEN AVERAGED OVER 25 INDEPENDENT RUNS.

Test Func	SaNSDE Mean	NSDE Mean	DE Mean	SaNSDE t-test	NSDE t-test
f_6	3.19e-01	1.51e+01	2.51e+01	-4.27†	-1.41
f_7	1.03e-02	4.00e-03	2.96e-03	2.87†	0.67
f_8	2.06e+01	2.09e+01	2.10e+01	-20.83†	-0.71
f_9	0.00e+00	7.96e-01	1.85e+01	-17.79†	-17.01†
f_{10}	3.26e+01	5.30e+01	9.69e+01	-3.90†	-2.57†
f_{11}	2.76e+01	2.21e+01	3.42e+01	-3.18†	-4.62†
f_{12}	2.90e+03	5.94e+03	2.75e+03	0.19	2.49†
f_{13}	1.79e+00	3.24e+00	3.23e+00	-8.41†	0.046
f_{14}	1.24e+01	1.33e+01	1.34e+01	-12.28†	-2.73†

† The value of t-test is significant at $\alpha = 0.05$ by a two-tailed test.

For multimodal functions $f_6 - f_{14}$, the advantages of introducing self-adaptive and neighborhood search strategies are still significant. Although DE performs better than NSDE on f_7, f_{12} and f_{13}, it was outperformed by NSDE on the other 6 functions. SaNSDE's results are even better. It has superior performance on 7 out of the 9 multimodal functions.

Based on the experimental results obtained on low-dimensional problems, we can conclude that SaNSDE is more suitable to be a basic subcomponents optimizer for high-dimensional optimization.

C. Performance of DECC on High-dimensional Problems

In this section we evaluate the performance of proposed DECC-I, DECC-II on 500 and 1000 dimensional problems. The classical DE is also scaled directly for comparison. We set the population size NP to 100 for all algorithms, and the dimensions of subcomponent s to 100 for DECC-I and DECC-II. The number of fitness evaluations for 500 and 1000 dimensional problems are 2500000 and 5000000, respectively[9]. The fitness of an individual in DECC was estimated by combining it with the current best individuals from other subcomponents to form a vector, and applying the vector to the target function [10], [9]. Since the fitness evaluation of high-dimensional functions are very time consuming, and even the values of some functions are too big to represent by computer, we only used 8 functions for experiments, including two separable functions (f_1, f_9), and six nonseparable functions [18]. The average results of 25 independent runs are summarized in Tables 3 and 4.

TABLE III
COMPARISON BETWEEN DECC-I, DECC-II AND DE ON UNIMODAL FUNCTIONS, WITH DIMENSION $D = 500$ AND 1000. ALL RESULTS HAVE BEEN AVERAGED OVER 25 INDEPENDENT RUNS.

Test Func	# of Dim's	DE Mean	DECC-I Mean	DECC-II Mean
f_1	500	1.56e+06	**2.18e-13**	4.07e-13
	1000	3.36e+06	**4.91e-13**	6.64e-13
f_3	500	9.25e+10	**1.75e+08**	2.29e+08
	1000	2.20e+11	**3.85e+08**	4.10e+08
f_5	500	2.39e+05	**9.65e+04**	1.75e+05
	1000	3.94e+05	**1.98e+05**	2.40e+05

For unimodal functions, both DECC-I and DECC-II outperformed the classical DE. DECC-I and DECC-II not only have essential improvement over DE on separable function f_1, but also show great efficiency on nonseparable f_3 and f_5. The evolution processes are shown in Figures 4 and 5.

Similar conclusions can be drawn for multimodal functions from Table 4. DECC-I and DECC-II performs significantly better than DE on almost all functions except for f_8. The three algorithms have exactly same results on function f_8. The fitness landscape of f_8 in Fig. 3 shows that it belongs to a class of deceptive problems, which are often regarded as being difficult to optimize. For the other functions, Figures 6 and 7 show that DE stagnates rather early in search and makes little progress thereafter, while DECC-I and DECC-II keep finding better function values throughout the evolu-

TABLE IV

COMPARISON BETWEEN DECC-I, DECC-II AND DE ON MULTIMODAL FUNCTIONS, WITH DIMENSION $D = 500$ AND 1000. ALL RESULTS HAVE BEEN AVERAGED OVER 25 INDEPENDENT RUNS.

Test Func	# of Dim's	DE Mean	DECC-I Mean	DECC-II Mean
f_6	500	1.68e+12	**4.91e+02**	8.82e+02
	1000	1.68e+12	**9.86e+02**	2.25e+03
f_8	500	2.16e+01	2.16e+01	2.16e+01
	1000	2.16e+01	2.16e+01	2.16e+01
f_9	500	8.85e+03	**2.22e+03**	2.91e+03
	1000	1.81e+04	**4.59e+03**	5.22e+03
f_{10}	500	1.38e+04	**2.58e+03**	5.88e+03
	1000	2.93e+04	**6.85e+03**	9.40e+03
f_{13}	500	7.90e+02	**2.14e+02**	2.94e+02
	1000	2.08e+03	**4.14e+02**	5.32e+02

tion. It's interesting to see that the cooperative coevolution framework works well even for nonseparable multimodal functions, which are more difficult for the method in [10]. Although some of the improvements might come from the import of SaNSDE, the great difference of optimization ability between DECC-I/DECC-II and DE make us believe the proposed cooperative coevolution framework is more suitable for high-dimensional optimization.

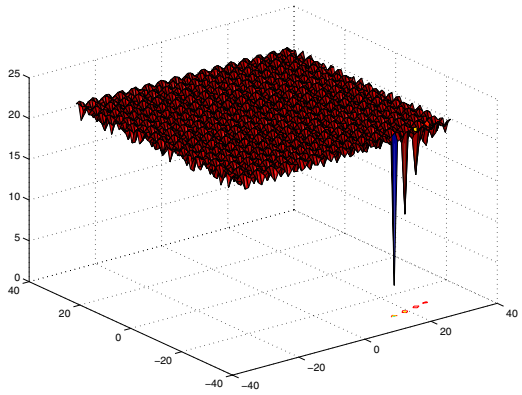

Fig. 3. The fitness landscape of two-dimensional version of f_8.

V. CONCLUSIONS

In this paper we propose two new variants of DE, named DECC-I and DECC-II, for high-dimensional optimization. They follow the cooperative coevolution framework proposed by Potter [10], but are extended with novel strategies on problem decomposition and subcomponents cooperation. DECC-I groups the object vector into several low-dimensional subcomponents that a basic optimizer can handle, and optimizes all subcomponents in each cycle, and then cooperates all subcomponents by optimizing the weights of them. On the other hand, DECC-II selects a low-dimensional subset randomly in each cycle, and then optimize the subset with a basic optimizer.

Experimental evidence is also provided to evaluate the effectiveness and efficiency of DECC-I and DECC-II on a set of widely used benchmark functions. The set consists of separable unimodal, nonseparable unimodal, separable multimodal, and nonseparable multimodal functions. These functions are scaled to 500 and 1000 dimensions. It was found that DECC-I and DECC-II worked well on all kinds of functions, even for the most difficult one (nonseparable multimodal). Comparing to the results got by directly scaled DE, DECC-I and DECC-II are definitely more suitable for high-dimensional problems.

In the experimental studies for high-dimensional functions, we have to fling away several functions for the difficulties of their fitness evaluation. One of the future improvements to DECC would be to apply advanced fitness evaluation methods [20].

ACKNOWLEDGMENT

The authors would like to thank P. N. Suganthan, A. K. Qin and V. L. Huang for they kindly shared some codes of DE variants. This work is partially supported by the National Natural Science Foundation of China (Grant No. 60428202 and 60573170).

REFERENCES

[1] R. Storn and K. Price, "Differential Evolution - A Simple and Efficient Heuristic Strategy for Global Optimization over Continuous Spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.

[2] J. Vesterstrom, R. Thomsen, "A Comparative Study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on Numerical Benchmark Problems," *Evolutionary Computation*, vol. 2, pp. 1980–1987, 2004.

[3] R. Storn, "System Design by Constraint Adaptation and Differential Evolution," *IEEE Transactions on Evolutionary Computation*, vol. 2, pp. 82–102, 1999.

[4] R. Thomsen, "Flexible ligand docking using differential evolution," *Proc. of the 2003 Congress on Evolutionary Computation*, vol. 4, pp. 2354–2361, 2003

[5] R. Gämperle, S. D. Müller and P. Koumoutsakos, "A Parameter Study for Differential Evolution," *Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, pp. 293–298, 2002

[6] D. Zaharie, "Critical values for the control parameters of differential evolution algorithms," *Proc. of the 8th International Conference on Soft Computing*, pp. 62–67, 2002

[7] Janez Brest et. al, "Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems," *IEEE Transactions on Evolutionary Computation*, vol. 2, pp. 82–102, 2006.

[8] A.K. Qin and P.N. Suganthan, "Self-adaptive Differential Evolution Algorithm for Numerical Optimization," *Proc. of the 2005 Congress on Evolutionary Computation*, vol. 2, pp. 1785–1791, 2005.

[9] Y. Liu, X. Yao, Q. Zhao and T. Higuchi, "Scaling Up Fast Evolutionary Porgramming with Cooperative Coevolution," *Proc. of the Congress on Evolutionary Computation*, pp. 1101–1108, 2001

[10] A. M. Potter and K. A. De Jong, "A cooperative co-evolutionary approach to function optimization," *Proc. of the Third International Conference on Parallel Problem Solving from Nature*, pp. 249–257, Springer-Verlag, 1994.

[11] D. Sofge, K. A. De Jong, and A. Schultz, "A blended population approach to cooperative coevolution for decomposition of complex problems," *Proc. of the Congress on Evolutionary Computation*, pp. 413–418, 2002.

[12] Bergh, F.v.d. and A. P. Engelbrecht, "A Cooperative Approach to Particle Swarm Optimization," *IEEE Transactions On Evolutionary Computation*, vol. 3 pp. 225–239, 2004.

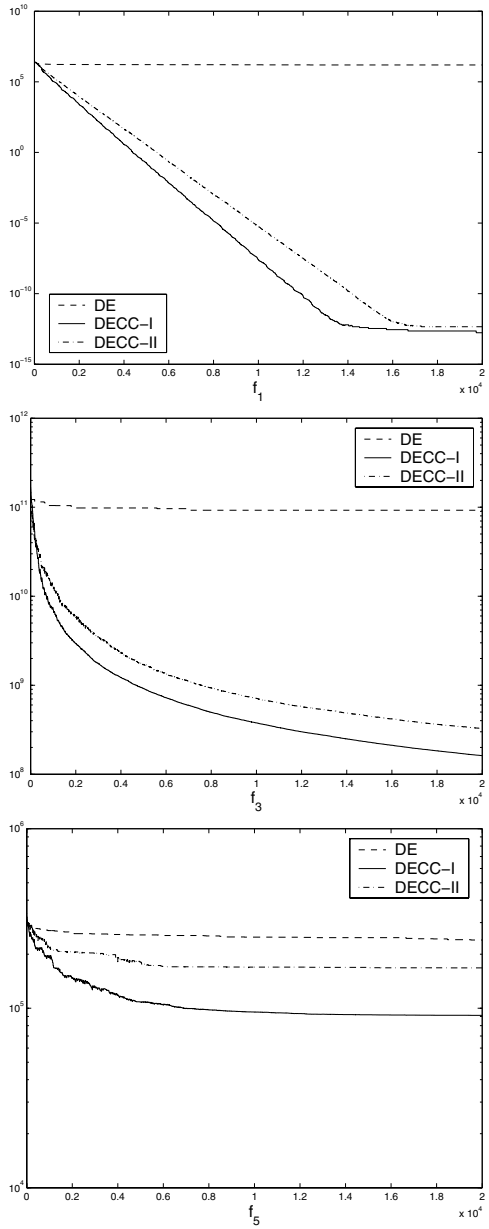

Fig. 4. The evolution process of the mean best values found for f_1, f_3 and f_5, with dimension $n = 500$. The results were averaged over 25 runs. The vertical axis is the function value and the horizontal axis is the number of generations.

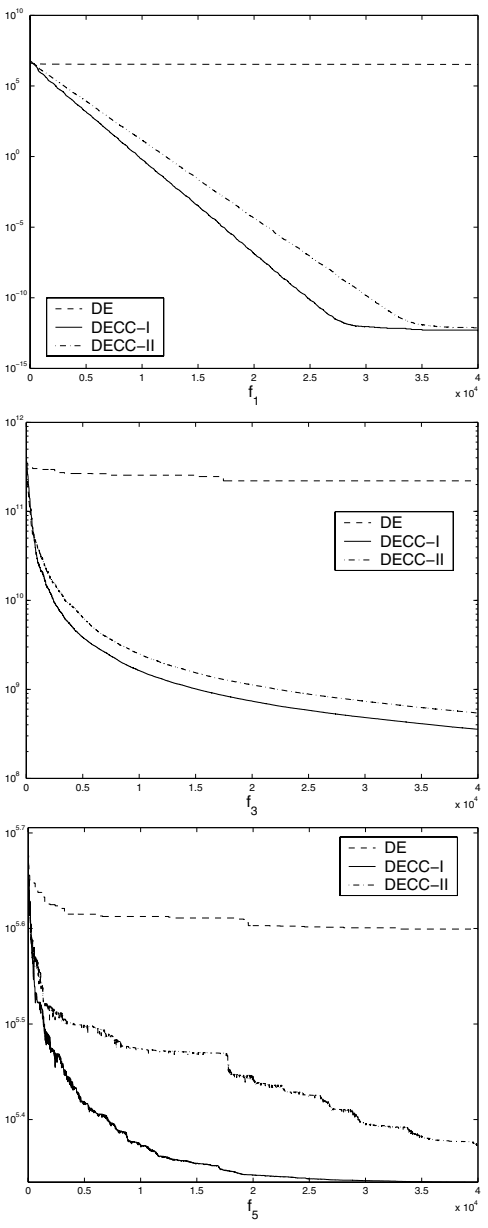

Fig. 5. The evolution process of the mean best values found for f_1, f_3 and f_5, with dimension $n = 1000$. The results were averaged over 25 runs. The vertical axis is the function value and the horizontal axis is the number of generations.

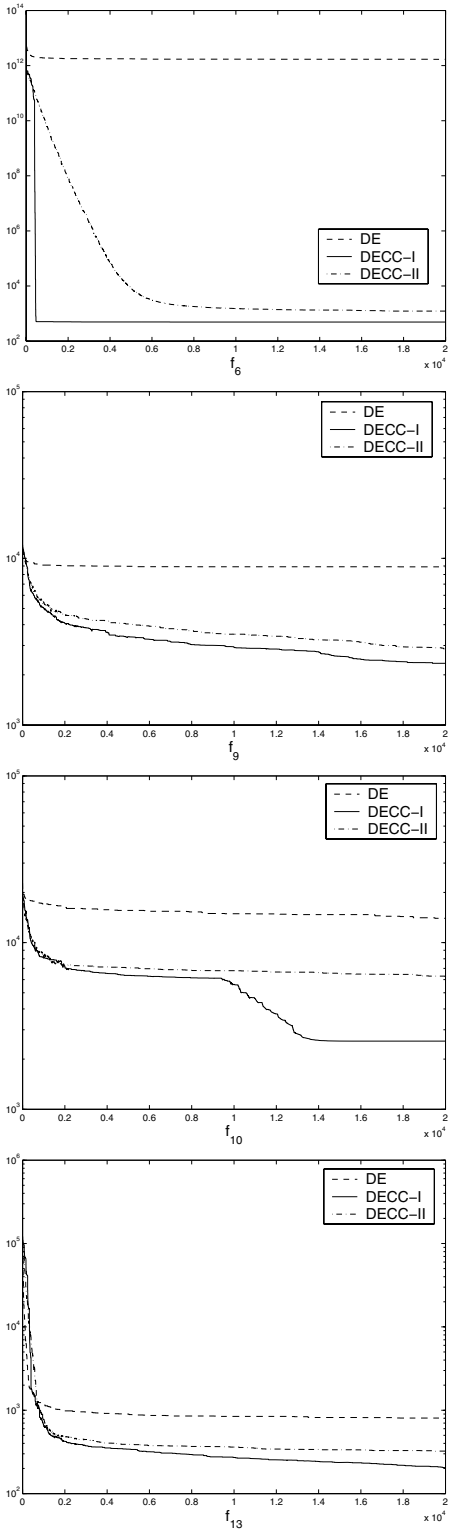

Fig. 6. The evolution process of the mean best values found for f_6, f_9, f_{10} and f_{13}, with dimension $n = 500$. The results were averaged over 25 runs. The vertical axis is the function value and the horizontal axis is the number of generations.

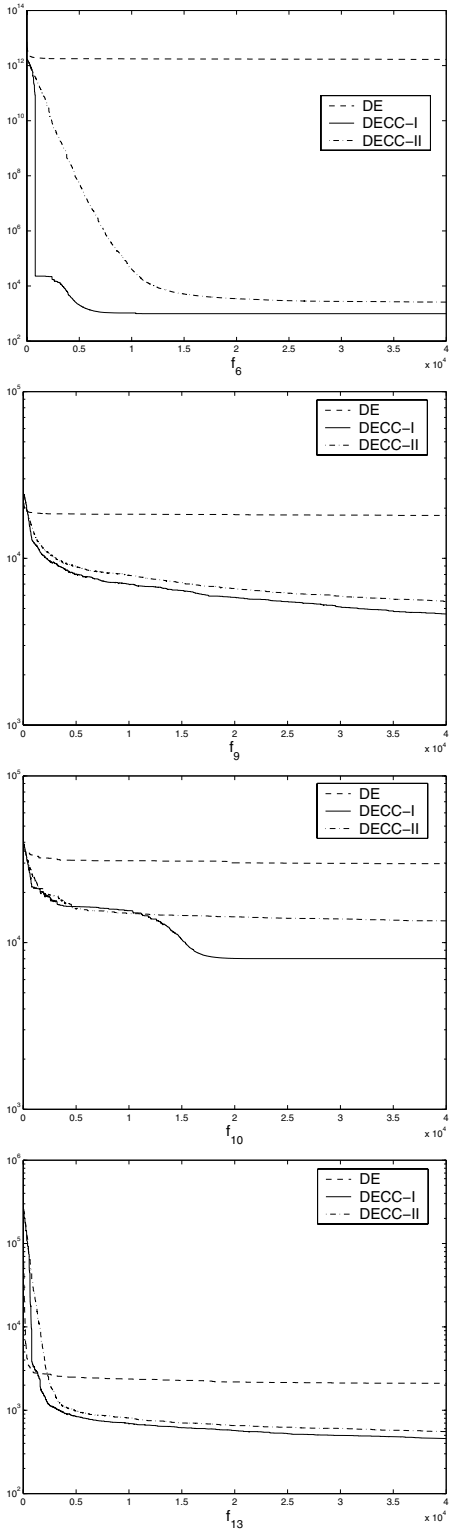

Fig. 7. The evolution process of the mean best values found for f_6, f_9, f_{10} and f_{13}, with dimension $n = 1000$. The results were averaged over 25 runs. The vertical axis is the function value and the horizontal axis is the number of generations.

[13] Y. Shi, H. Teng and Z. Li, "Cooperative Co-evolutionary Differential Evolution for Function Optimization," *Proc. of the First International Conference on Natural Computation*, pp. 1080–1088, 2005.
[14] T. Bäck, H. P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evolutionary Computation*, vol. 1 pp. 1–23, 1993.
[15] X. Yao, Y. Liu, G. Lin, "Evolutionary Programming Made Faster," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 82–102, 1999.
[16] C. Lee, X. Yao, "Evolutionary Programming Using Mutations Based on the Lévy Probability Distribution," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 1, pp. 1–13, 2004.
[17] Z. Yang, J. He, X. Yao, "Make a difference to differential evolution," *submitted to a book chapter*, 2006.
[18] P. N. Suganthan et al, "Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization," `http://www.ntu.edu.sg/home/EPNSugan`, 2005.
[19] J. Rönkkönen, S. Kukkonen, K. V. Price, "Real-Parameter Optimization with Differential Evolution," *Proc. of the 2005 Congress on Evolutionary Computation*, vol. 1, pp. 567–574, 2005.
[20] Q. F. Zhao, O. Hammami, K. Kuroda and K. Saito, "Cooperative co-evolutionary algorithm – how to evaluate a module?" *Proc. of the first IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*, pp. 150–157, 2000.