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

A Hybrid Adaptive Coevolutionary Differential Evolution Algorithm for Large-scale Optimization

Sishi Ye, Guangming Dai*, Lei Peng, Maocai Wang
School of Computer Science
China University of Geosciences
Wuhan, China
gmdai@cug.edu.cn

***Abstract*—In this paper, we propose a new algorithm, named HACC-D, for large scale optimization problems. The motivation is to improve the optimization method for the subcomponents in the cooperative coevolution framework. In the new HACC-D algorithm, an algorithm selection method named hybrid adaptive optimization strategy is used. It is aimed to hybridize the superiority of two very efficient differential evolution algorithms, JADE and SaNSDE, as the subcomponent optimization algorithm of the cooperative coevolution. In the beginning stage, the novel strategy evolves the initial population with JADE and SaNSDE as the subcomponent optimization algorithm for a certain number of iterations separately. Then the one obtained better fitness value will be chosen to be the subcomponent optimization algorithm for the following evolution process. In the later stage of evolution, the selected algorithm may be trapped in a local optimum or lose its ability to make further progress. So it exchanges the subcomponent optimization algorithm with the other one when there is no improvement in the fitness every certain number of iterations. The proposed HACC-D algorithm is evaluated on CEC'2010 benchmark functions for large scale global optimization.**

Keywords—hybrid adaptive optimization; differential evolution; cooperative coevolution; large scale global optimization

I. INTRODUCTION

Many Evolution Algorithm (EAs) have been used for large scale optimization problems. But the performance of these algorithms deteriorate rapidly as the dimensionality of problem increases. It is chiefly derived from the exponential growth in the size of search space of the problems [1]. As EAs are applied to increasingly large and complex problems, their scalability has become one of the most urgent challenges. The problem of finding the global optimum becomes even more difficult when some or all of the decision variables have interaction among themselves. This kind of problems is classified as non-separable problems. Variable interaction in large scale problems drastically increases the total number of function evaluation in order to find a reasonable solution. Recently, there are two solutions to solve large scale optimization problems. The first one is to utilize a decomposition strategy, and the second one is to apply a hybridization approach.

The first attempt of decomposition was the Cooperative Coevolution (CC) method proposed by Potter et al. [1]. CC is a popular technique in EAs for large scale optimization. It uses a divide-and-conquer approach to divide the decision variables into low dimensional subcomponents, each of which is optimized with a certain EA in a round robin fashion. CC strategy is very successful with separable optimization problems but lose its efficiency with non-separable ones. In order to improve the performance of CC with non-separable optimization problems, the variable interaction among the subcomponents should be minimized. This emphasizes the importance of decomposition strategy in CC. Some kinds of grouping approaches were developed recently, such as Random Grouping (RG) technique [2], correlation based Adaptive Variable Partitioning (AVP) [3], Delta Grouping [4], and Variable Interactive Learning (VIL) [5].

The second solution is in hybrid manner. The hybridization of the EAs with other technique has been proven to enhance the performance of the optimization algorithms when solving large scale optimization problem. Memetic Algorithms (MAs) has become a popular one for large scale optimization [6, 7]. It is a hybridization of EA and Local Search (LS). Exploration and exploitation are two major issues when designing a global search method. MAs attempt to accomplish the compromise by putting together two specialized components: an EA that may assume the task of exploring the search space, and an LS algorithm that refines promising individuals being evolved by the EA. The ratio of local search and global search has a significant influence on the performance of the algorithm. In this paper, we focus on the first solution to solve large scale optimization problems.

Differential Evolution (DE) is a popular algorithm as the subcomponent optimization algorithm of CC. There are many sorts of adaptive and self-adaptive DE variants [8-11]. Nearly all of the algorithms for large scale optimization problems only use a single DE variant as the subcomponent optimization algorithm [2,4,12,13]. Self-adaptive Differential Evolution with Neighborhood Search (SaNSDE) [10] has the highest adoption rate [2,4,12]. However, there are several

This work was supported by "Twelve Five-Year Plan" Civil Aerospace Professional and Technical Pre-Research Project, the National Natural Science Foundation of China under Grant No. 61103144, 60873107 and China Postdoctoral Science Foundation Funded Project No. 2011M501260, 2012T50681, 2012M511301 and Hubei Natural Science Foundation under grant No. 2010CDB04104, 2011CDB348 and the Fundamental Research Funds for the Central Universities, China University of Geosciences(Wuhan) No.CUG120114.

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

proposed DE variants with different search preference have shown great performance as well. Recently in a newly proposed algorithm named JACC-G [13], a new variant of DE called JADE [5] has been applied to CC as its subcomponent optimization algorithm. JADE is an adaptive DE algorithm with a novel mutation strategy, called "*DE/current-to-pbest"*, and an optional external archive. Different DE variants might provide different searching manner toward the global optimum and improve the diversity of the population as well.

In order to utilize different search bias in the subcomponents evolution process, we proposed a algorithm selection method, named *hybrid adaptive optimization strategy*, to synthesize the superiority of different DE variants serving as the subcomponent optimization algorithm of CC. Among the DE variants, JADE and SaNSDE are the two most efficient ones [4,5]. Both of them are adaptive and have show great performance in solving large scale optimization problems [2,4,12,13]. It is worth the effort to explore the possibility of hybridizing their efficiency and searching bias to tackle large scale optimization problems.

The *hybrid adaptive optimization strategy* composites the searching preferences of JADE and SaNSDE in a two-level adaptive manner. Considering that it is difficult to distinguish which DE candidate is more suitable for the current problem without any prior knowledge, there needs a way to find out the better one in the early stage of the evolution process. Wrong selection might lead to small progress at the beginning of evolution and a waste of fitness evaluation number, especially when the selected one is almost unable to make any progress. In the later period of evolution, the selected DE candidate might be trapped in a local optimum or lose its ability towards the global optimum further more. However, the unselected DE algorithm might be more efficient than the selected one. The exchange will infuse new searching ability and direction into the evolution process. Experimental analysis reveals that this new technique successfully integrated the search superiority of these two DE algorithms for solving large scale optimization problems.

The organization of the rest of this paper is as follows. Section II gives a briefly explanation of the preliminaries and background information. Section III describes the proposed hybrid adaptive optimization strategy in details. Section IV demonstrates and analyzes the experimental results. Finally Section V concludes this paper and give directions for further potential improvements.

II. PRELIMINARIES

A. Cooperative Coevolution

Based on a divide-and-conquer manner, CC has been proved to be a promising framework for tackling those large scale optimization problems. The original CC proposed by Potter and De Jong decomposes the decision variables into smaller subcomponents each of which is optimized by a certain EA separately. This algorithm is named as Cooperative Coevolution Genetic Algorithm (CCGA) [1]. They successfully incorporated CC into Genetic Algorithm for function optimization. It is a significant inspiration for the incorporation of CC with various kinds of EA such as Evolutionary Programming [14], Evolutionary Strategies [15], Particle Swarm Optimization [16], and Differential Evolution [2,4,7].

Liu et al. made the first attempt of applying CC to solve large scale optimization problems. They combined Fast Evolutionary Programming with Cooperative Coevolution (FEPCC) [18] for tackling problems with up to 1000 dimensions. However, it is inefficient in dealing with non-separable functions. It demonstrated that CC loses its ability in variables interaction.

The first combination of CC with PSO which was implemented by Van Ben Bergh and Englbrecht led to the formation of Cooperative Particle Swarm Optimization (CPO) [16]. It adopts a statistic grouping method that the arrangement of variables stays the same during the evolution process. By concatenating the variables of every individual with the best-fit individuals of other subcomponents which forms a *context vector* [14], the individuals in each of the subcomponents is evaluated. Then the context vector is fed into fitness function for evaluation.

Shi et al. [17] proposed a decomposition strategy which applied DE into CC. They used a splitting-in-half strategy that divides the decision variables into halves and each of them is optimized by DE. With the increasing dimension of the halves, this strategy can hardly be scaled up in an effective way.

Yang et al. [2] took the first step to develop a more systematic way of dealing with variable interaction by random grouping the decision variables into different subcomponents. In Random Grouping approach, every decision variable is randomly arranged into any of the subcomponents with equal probability and it is repeated at the beginning of every *cycle* [2]. Despite its success in increasing the probability of grouping two interacting variables into the same subcomponent, the efficiency drops considerably when there are more than two interacting variables.

Based on the shortcoming of Random Grouping, Omidvar et al. proposed a more systematic strategy named Delta Grouping [4] to capture the interaction among variables. Unlike the blind mechanism of random grouping, Delta Grouping calculates the amount of change (*delta value*) in each of the decision variables in every iteration to identify the interacting variables. Eventually, the grouping of decision variables is decided on the basis of the sorted *delta values*. The Delta Grouping method showed its good performance in grouping up to fifty interacting variables in to the same subcomponent. In this paper, we adopt the Delta Grouping approach as the problem decomposition method in HACC-D algorithm.

B. Algorithm Selection Method

Most problems can be solved by more than one algorithm. The choice of the algorithm can dramatically affect the quality of the solution and the time spent obtaining it. Algorithm selection is aimed to identify the best-performing algorithm from a set of candidate algorithms. Existing approaches for algorithm selection can be divided into two main categories,

i.e., the so-called inter-problem methods and the intra-problem ones.

An inter-problem approach usually focuses on selecting algorithm for a given problem class. For example, statistical racing [19] is a general-purpose tool to find an algorithm that performs as well as possible on a problem class. First, a number of problem instances are sampled and used as the training instances. Then, candidate algorithms are evaluated on the training instances. The algorithms that perform poorly will be discarded sequentially as soon as statistically sufficient evidence is gathered against them.

Differently from the inter-problem approaches, a typical intra-problem method aims to select the best algorithm for a single problem instance instead of a problem class. A representative method is "racing multiple algorithms on a single problem" approach proposed by Yuan and Gallagher [20], which is an extension of statical racing. For a given problem, it first executes all the candidate algorithms on the problem and compares the different algorithms with a pre-defined statistical test. This procedure is repeated until only one candidate is left or the time budget is used up. In this paper, the proposed *hybrid adaptive optimization strategy* is more like a intra-problem method.

III. Proposed Techniques

A. Hybrid Adaptive Optimization Strategy

Most of the recent proposed algorithms for large scale optimization are based on CC. Almost all of them only adopt a single DE variant as the subcomponent optimization algorithm while there are several DE variants with different search preferences and great efficiency. It is worth the effort to explore the possibility of hybridizing the search bias of different DE variants. Among all of them, SaNSDE and JADE are the two most efficient ones [10, 11].

SaNSDE integrates the advantages of SaDE [8] and NSDE [20]. It utilizes the self-adaptive mutation strategy of SaDE which composites two well performed mutation strategies to avoid the dilemma that the mutation strategies of DE are often highly dependent on the problems. In another aspect, SaNSDE makes use of the neighborhood searching method of NSDE to mix two *NS operators* with different searching preferences by the scaling factor *F*. The *NS operators* are Cauchy random number and Gaussian random number. The former one is more likely to provide long jumps in the early period of evolution which will make the population move fast forwards. The latter one is more feasible to produce small jumps when the current searching point is near the global optimum. In addition, the parameter *CR* of SaNSDE is based on a weighted self-adaptive manner to learn from the successful crossover operation.

JADE is an adaptive DE algorithm with a novel mutation strategy called "*DE/current-to-pbest*" and an optional external archive. The new mutation strategy in JADE is a generalization of the traditional "*DE/current-to-best*". It relies on the historical data instead of the best solutions in the current population. The best solution is usually less reliable and might lead to premature convergence [21,22]. The external archive of JADE stores the inferior solutions that fail in the selection process. Their difference from the current population might give a promising direction towards the global optimum when the evolution procedure is trapped in a local optimum and improve the diversity of the population as well.

The new proposed technique is aimed to integrate the preponderance of JADE and SaNSDE as subcomponent optimization algorithm of CC to tackle the large scale optimization problems. The *hybrid adaptive optimization strategy* hybridizes the DE candidates at two levels:

1) In the beginning stage of evolution, the initial population is evolves with JADE and SaNSDE as subcomponent optimization algorithm for an appropriate number of iterations separately. Then, the one gained better fitness value will go into the following evolution procedure and the worse one is abandoned.

The rationality behind it is that the two DE candidates have different superiority on different problems. Also, it is unclear which one is more suitable for the current problem without any available prior knowledge. Therefore, a number of iterations (called *js*) is spent in the early learning stage to identify either JADE or SaNSDE is more suitable for the current problem. The time (i.e. *js*) allocated for the early learning stage needs to be proper. Obviously, a large time budget might lead to a waste of fitness evaluation number and a small one is not enough to identify the appropriate DE candidate.

2) In the later period of evolution process, the selected subcomponent optimization algorithm might lead the evolution process be trapped in a local optimum or lose its efficiency to make further progress. Thus, it exchanges the subcomponent optimization algorithm when there is not any improvement in the fitness value every certain number of iterations (called *denum*). The different searching ability and bias of the new one might infuse the population with new energy and help to escape from a local optimum.

Considering that both JADE and SaNSDE use an adaptive mechanism to update their control parameters. The value of *denum* should not destroy the original exploration ability of them. The control parameters *F* and *CR* of JADE is generated according to a Cauchy distribution and a normal distribution separately. They are regenerated in every generation which is a very fast fashion. On the other hand, the parameter *F* and *CR* in SaNSDE are updated every 50 iterations. This relatively slow mechanism means to learn from the parameter values of the successful offspring. So the value of *denum* needs to be carefully designed. A small value is not practical and might cause a bad influence on the original searching ability of the DE candidates. A large value for *denum* could not guarantee changing the subcomponent optimization algorithm in time, and will make the hybrid adaptive optimization strategy lose its meaning and efficiency. Given the features of JADE and SaNSDE, the setting of *denum* should be at least bigger than 50 and the multiple of it.

B. HACC-D

Based on the idea of Delta Grouping and the hybrid

adaptive optimization strategy, the primary steps of HACC-D is as follow:

1) Uniformly randomly initialize the population. Set $i = 1$ to start a new *cycle*.

2) Initialize the Δ vector to zero. It indicates that the Delta Grouping is not used for sorting the variables in the first *cycle*.

3) Divide the decision variables into predefined-size subcomponents. Then optimize the initial population *pop* with JADE and SaNSDE as its subcomponent optimization algorithm for *js* iterations separately. Then get their current best fitness value *jbest* and *sbest*. Note that each of the subcomponent is optimized for only one iteration at a time before evaluation. Also, sort the decision variables based on the magnitude of their corresponding delta value in every iterations.

4) If *jbest* < *sbest*, choose JADE to be the optimizer in the following evolution. Otherwise, select SaNSDE instead.

5) Set the *num* = 0 to start a monitor on the improvement of fitness value.

6) Evolve all the subcomponents with the selected DE candidate for one iteration. If the improvement of fitness is zero between two consecutive cycles, *num*++. Otherwise, reset *num* = 0.

7) If *num* reaches to the predefined threshold number *denum*, replace the subcomponent optimization algorithm with the other one and reset *num* = 0.

8) Stop if the halting criterion is satisfied. Otherwise, go to step 6) for the next *cycle*.

Here one complete evolution of all subcomponents is called a *cycle*. The two control parameters of HACC-D, *js* and *denum*, need to be well defined in order to be effective for as many kinds of problems as possible. Based on their roles in HACC-D, we expect these two parameters to be insensitive to different sorts of problems. As shown in Section IV, HACC-D has the best performance for most of the problems with $js \in [6000, 7000]$ and $denum \in [50, 150]$.

IV. Experimental Results

A. Experiment Setup

All the algorithms are evaluated on CEC'2010 benchmark functions proposed for the Special Session and Competition on Large Scale Global Optimization [23]. This set of benchmark functions is suitable because the non-separability degree is well defined and adjustable. The benchmark functions are scalable as well.

For the purpose of choosing appropriate setting for the control parameters of HACC-D, we conducted two sets of comparative experiment. In the first set of experiment, the *denum* value are fixed to 100 for all the compared HACC-D algorithms while the value of *js* is ranging from 3000 to 10000. In the second set of experiment, the value of *js* are fixed to the best value gained from the first set of comparative experiment for all the HACC-D algorithm while the value of *denum* is ranging from 50 to 200.

Each of the algorithms is conducted 25 independent runs on every function. The population size of them are set to 50 and the maximum number of fitness evaluations are set to 3×10^6 according to [23]. The dimension of the benchmark functions are set to 1000 and the subcomponent size of all the algorithms are set to 100. The global optimum value are zero for all the benchmark functions.

B. Analysis of Results

The experimental results and analysis of the outputs are contained in this section. In order to set a reference line for the comparative experiments analysis, we conducted experiments on DECC-D with SaNSDE as its subcomponent optimization algorithm and DECC-D with JADE as its subcomponent optimization algorithm separately. These two algorithms are called *the two basic algorithms* in the following paper. The better results of the two basic algorithms is highlighted in bold and italic in Table I. The comparative results of the two basic

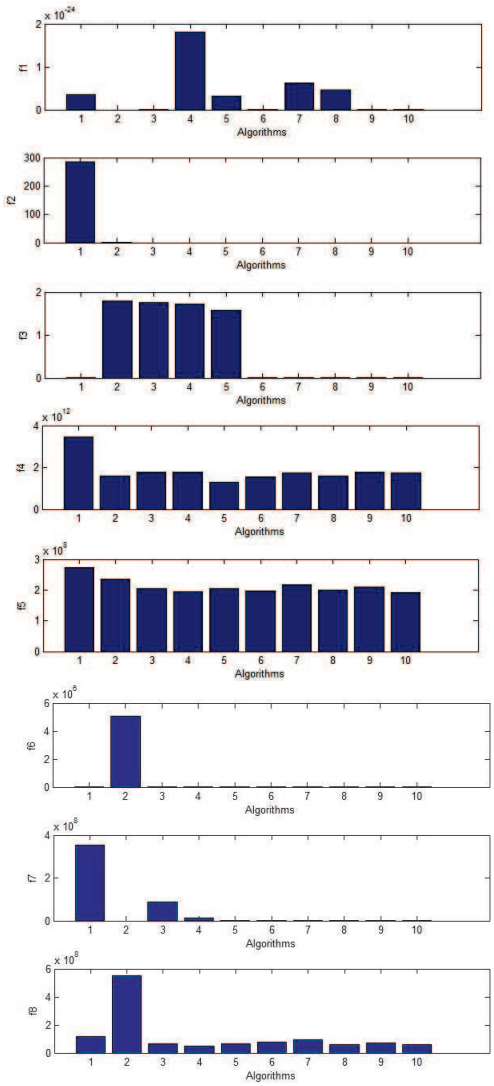

Fig. 1. The average best fitness value ranking of function 1 to 9 for the two basic algorithms and eight HACC-D algorithms with *js* ranging from 3000 to 10000 is shown above. The best fitness value of all the algorithms is the mean of 25 independent runs. The columns in each histogram has the same order as Table I.

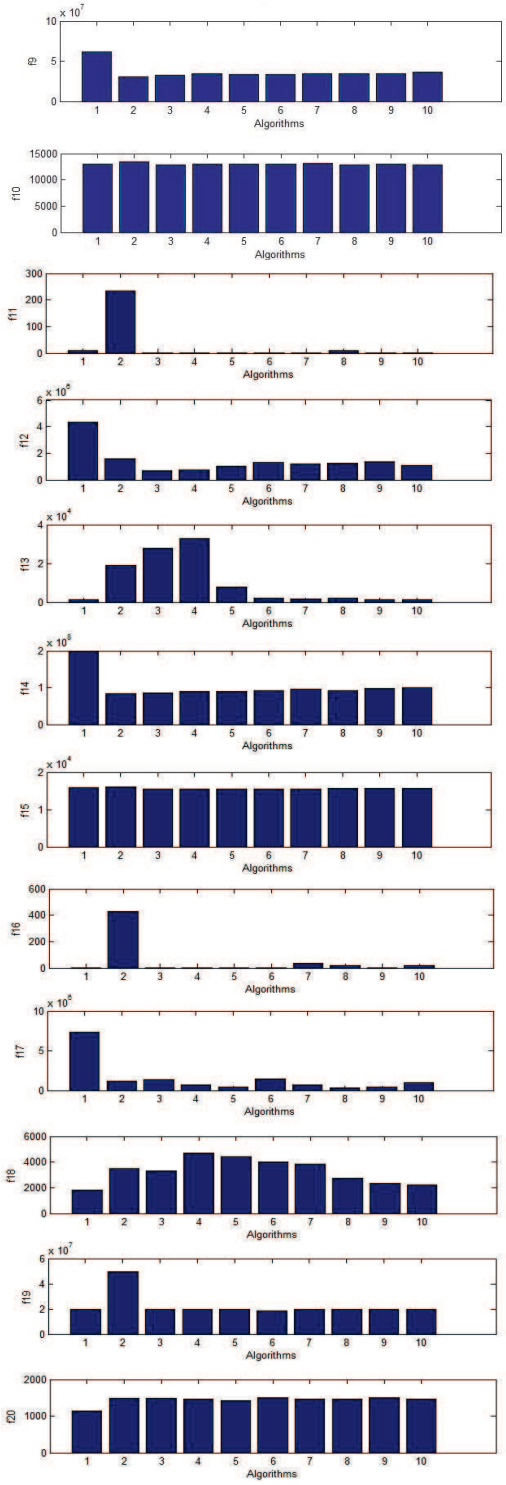

Fig. 2. The average best fitness value ranking of function 10 to 20 for the two basic algorithms and eight HACC-D algorithms with *js* ranging from 3000 to 10000 is shown above. The best fitness value of all the algorithms is the mean of 25 independent runs. The columns in each histogram has the same order as Table I.

algorithms show that their performance gaps are significantly large on functions f_2, f_3, f_6, f_7 and f_{16} which is a reflection of the different search preferences of SaNSDE and JADE.

For the seeking of proper setting for the control parameters of HACC-D, two groups of comparative experiment were conducted on each of the parameter with the other one kept fixed. The results of the first group of comparative experiment are presented in Table I. It is focused on exploring the suitable value for *js*. The results which are better than the better results of *the two basic algorithms* are highlighted in bold. It can see that the results on f_2, f_3 and f_6 are better than the better results of the two basic algorithms when the value of *js* is 6000 or 7000. Among the eight comparative algorithms of

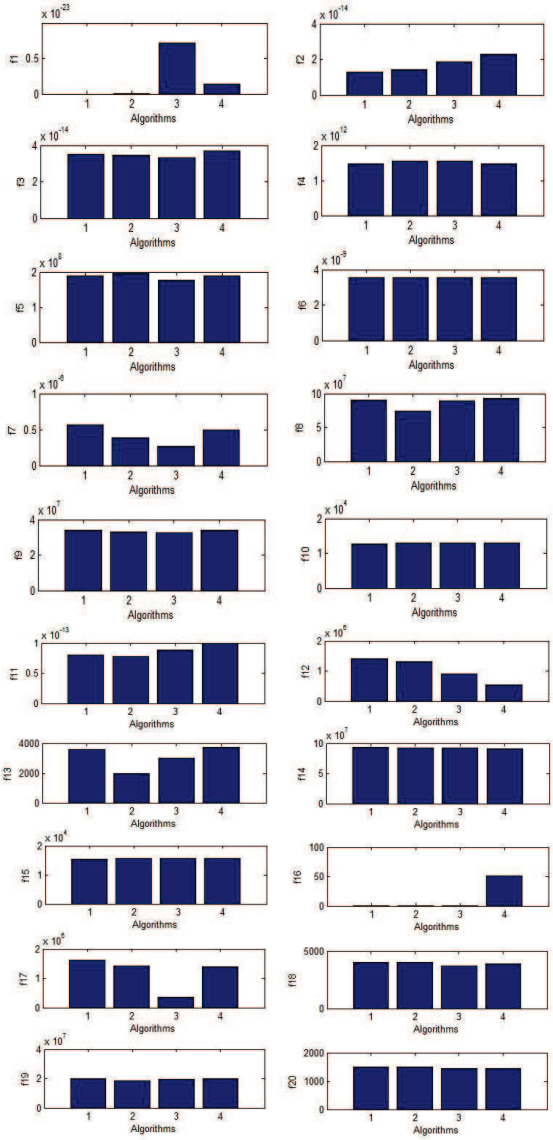

Fig. 3. The average best fitness value ranking of function 1 to 20 for the two basic algorithms and four HACC-D algorithms with *denum* ranging from 50 to 100 is shown above. The best fitness value of all the algorithms is the mean of 25 independent runs. The columns in each histogram has the same order as Table II.

TABLE I

COMPARISON OF DIFFERENT VALUE OF *JS* ON CEC'2010 FUNCTIONS WITH 1000 DIMENSIONS. THE VALUES OF *DENUM* ARE SET TO 100. NUMBERS SHOW THE MEAN OF THE BEST FITNESS OVER 25 RUNS. THE BETTER RESULTS OF THE FIRST TWO COLUMNS ARE IN BOLD AND ITALIC. THE RESULTS OF THE LATTER EIGHT COLUMNS THAT ARE BETTER THAN THE BETTER ONE OF THE FIRST TWO COLUMNS ARE IN BOLD.

Function	DECC-D with SaNSDE	DECC-D with JADE	HACC-D (*js* = 3000)	HACC-D (*js* = 4000)	HACC-D (*js* = 5000)	HACC-D (*js* = 6000)	HACC-D (*js* = 7000)	HACC-D (*js* = 8000)	HACC-D (*js* = 9000)	HACC-D (*js* = 10000)
f_1	3.57794E-25	***0.00E+00***	1.76090E-27	1.81406E-24	3.17611E-25	1.98695E-27	6.23376E-25	4.69019E-25	**1.34774E-30**	3.50832E-29
f_2	2.84689E+02	***1.14966E-13***	**9.30811E-15**	**1.19371E-14**	**1.15818E-14**	**1.42819E-14**	**1.95399E-14**	**1.83320E-14**	**2.26663E-14**	**2.68585E-14**
f_3	***1.22782E-13***	1.78704E+00	1.74939E+00	1.71724E+00	1.57388E+00	**3.45324E-14**	**3.48166E-14**	**3.58114E-14**	**3.73745E-14**	**3.63798E-14**
f_4	3.44753E+12	***1.60213E+12***	1.79162E+12	1.79188E+12	**1.30266E+12**	**1.55083E+12**	1.73135E+12	**1.58576E+12**	1.75701E+12	1.73148E+12
f_5	***2.72363E+08***	2.36021E+08	**2.04688E+08**	**1.95657E+08**	**2.05167E+08**	**1.96257E+08**	**2.16391E+08**	**2.00395E+08**	**2.09385E+08**	**1.92507E+08**
f_6	***4.68970E-09***	5.09187E+06	**3.55274E-09**	**3.55275E-09**	**3.55275E-09**	**3.55275E-09**	**3.41064E-09**	**3.41064E-09**	**3.55275E-09**	**3.55275E-09**
f_7	3.52767E+08	***8.45151E-09***	8.64966E+07	1.25250E+07	3.04321E-07	3.86854E-07	8.28283E-07	2.03944E-06	3.03527E-06	8.89508E-06
f_8	***1.19507E+08***	5.52296E+08	**6.30138E+07**	**4.83415E+07**	**6.55209E+07**	**7.43599E+07**	**9.53725E+07**	**6.16187E+07**	**6.99627E+07**	**5.73891E+07**
f_9	6.21254E+07	***3.06609E+07***	3.23031E+07	3.45962E+07	3.35917E+07	3.31677E+07	3.42528E+07	3.40635E+07	3.42916E+07	3.62148E+07
f_{10}	***1.29672E+04***	1.34031E+04	**1.29189E+04**	1.30495E+04	1.29915E+04	1.29697E+04	1.30879E+04	**1.28996E+04**	1.29945E+04	**1.29164E+04**
f_{11}	***9.36254E+00***	2.34567E+02	**7.97229E-14**	**8.18545E-14**	**8.08598E-14**	**7.81597E-14**	**7.98650E-14**	**9.30189E+00**	**9.22284E-14**	**9.06653E-14**
f_{12}	4.36961E+06	***1.57476E+06***	**6.79974E+05**	**7.42874E+05**	**1.06460E+06**	**1.30627E+06**	**1.18706E+06**	**1.23500E+06**	**1.36350E+06**	**1.09518E+06**
f_{13}	***1.30132E+03***	1.91958E+04	2.78523E+04	3.30757E+04	7.74810E+03	1.96263E+03	1.61381E+03	2.15509E+03	**1.23001E+03**	1.38381E+03
f_{14}	1.99022E+08	***8.30360E+07***	8.48703E+07	8.97985E+07	8.95303E+07	9.21128E+07	9.53143E+07	9.23374E+07	9.82225E+07	1.00453E+08
f_{15}	***1.58924E+04***	1.60453E+04	**1.55809E+04**	**1.55239E+04**	**1.55848E+04**	**1.55630E+04**	**1.55078E+04**	**1.56680E+04**	**1.57054E+04**	**1.56496E+04**
f_{16}	***2.22826E-13***	4.28221E+02	1.10148E-01	4.10833E-02	4.10833E-02	**1.95131E-11**	3.41654E+01	1.69685E+01	1.32775E-01	1.70647E+01
f_{17}	7.38221E+06	***1.20642E+06***	1.35663E+06	**6.91856E+05**	**3.93122E+05**	1.41958E+06	**7.45514E+05**	**3.70285E+05**	**4.11024E+05**	**9.80619E+05**
f_{18}	***1.83549E+03***	3.47312E+03	3.29985E+03	4.68288E+03	4.43172E+03	4.02378E+03	3.85539E+03	2.74920E+03	2.30892E+03	2.19868E+03
f_{19}	***1.96141E+07***	4.98982E+07	1.96217E+07	1.98800E+07	**1.95914E+07**	**1.86875E+07**	**1.95151E+07**	**1.94544E+07**	1.96884E+07	**1.94744E+07**
f_{20}	***1.14341E+03***	1.48097E+03	1.47984E+03	1.47136E+03	1.41765E+03	1.50615E+03	1.46568E+03	1.47348E+03	1.50058E+03	1.47194E+03

HACC-D with different value of *js*, the one with *js* valuing 6000 obtained the best result on f_{16}. It is the closest one to the better result of the two basic algorithms. Furthermore, all the HACC-D algorithms besides the one with *js* set to 8000 obtained good results far better than the better result of the two basic algorithms on f_{11}. The algorithms with *js* ranging from 5000 to 10000 achieved great results on f_7 which are very close to the better result of the two basic algorithms.

Last but not least, all the eight algorithms with different value of *js* almost have the same results or a little better than the better results of the two basic algorithms on the rest of the benchmark functions. Since HACC-D spends a certain time budget in the early learning stage to select the suitable DE candidate for the problem, it has less time budget for the later evolution process. This may cause its poor performance on some of the functions comparing with the two basic algorithms. The average ranking for the two basic algorithms and those eight HACC-D algorithms with different *js* value is shown in Fig. 1 and Fig. 2. The order of these algorithms is the same as they are in Table I.

Given the results from the first set of comparative experiment on *js*, the most suitable value of *js* is around 6000. So we conducted another group of comparative experiment on the setting of *denum* with all their *js* fixed to 6000. The results are revealed in Table II. The best results on all the functions are highlighted in bold. For the functions where the two basic algorithms have remarkable difference, all of the four HACC-D algorithms with different *denum* values performed well on them except on f_{16}. Only the ones with *denum* set to 50 and 100 have good performance on f_{16}. Comparing these two algorithms with *denum* set to 50 and 100, it indicates that the one with *denum* set to 100 have better results on 12 out of 20 functions than the one with *denum* set to 50. Therefore, we can conclude that the most appropriate value for *denum* is around 100. The average ranking for these four HACC-D algorithms with different *denum* value is shown in Fig. 3.

From the analysis results of the two groups of comparative experiments on *js* and *denum*, we can give a conclusion that

TABLE II

COMPARISON OF DIFFERENT VALUE OF *DENUM* ON CEC'2010 FUNCTIONS WITH 1000 DIMENSIONS. THE VALUES OF *JS* ARE SET TO 6000. NUMBERS SHOW THE MEAN OF THE BEST FITNESS OVER 25 RUNS. BEST RESULTS ARE IN BOLD.

Function	HACC-D (*denum* = 50)	HACC-D (*denum* = 100)	HACC-D (*denum* = 150)	HACC-D (*denum* = 200)
f_1	**0.00E+00**	1.98695E-27	7.22620E-24	1.41435E-24
f_2	**1.27187E-14**	1.42819E-14	1.87583E-14	2.29505E-14
f_3	3.51008E-14	3.45324E-14	**3.31113E-14**	3.70903E-14
f_4	**1.47153E+12**	1.55083E+12	1.55508E+12	1.48589E+12
f_5	1.88494E+08	1.96257E+08	**1.77787E+08**	1.90083E+08
f_6	**3.55275E-09**	**3.55275E-09**	**3.55275E-09**	**3.55275E-09**
f_7	5.69671E-07	3.86854E-07	**2.65166E-07**	4.90587E-07
f_8	9.10512E+07	**7.43599E+07**	8.92652E+07	9.26996E+07
f_9	3.37695E+07	3.31677E+07	**3.24815E+07**	3.37418E+07
f_{10}	**1.28347E+04**	1.29697E+04	1.29891E+04	1.28848E+04
f_{11}	8.07177E-14	**7.81597E-14**	8.81073E-14	9.93339E-14
f_{12}	1.40832E+06	1.30627E+06	8.97216E+05	**5.40116E+05**
f_{13}	3.60962E+03	**1.96263E+03**	3.02180E+03	3.73116E+03
f_{14}	9.38062E+07	9.21128E+07	9.27589E+07	**9.08917E+07**
f_{15}	**1.54656E+04**	1.55630E+04	1.56981E+04	1.56818E+04
f_{16}	**1.23066E-13**	1.95131E-11	6.24513E-02	5.11992E+01
f_{17}	1.60879E+06	1.41958E+06	**3.55930E+05**	1.38588E+06
f_{18}	4.03339E+03	4.02378E+03	**3.69344E+03**	3.89739E+03
f_{19}	1.99009E+07	**1.86875E+07**	1.96428E+07	2.01051E+07
f_{20}	1.49404E+03	1.50615E+03	**1.44411E+03**	1.45724E+03

the most appropriate parameter setting for HACC-D is that *js* is around 6000 and *denum* is around 100.

In Table III, the best, worst and median mean and standard deviation are recorded. The HACC-D algorithm in Table III has the setting of *js* = 6000 and *denum* = 100. The information is recorded at different stages of evolution to present the convergence behavior of the new proposed algorithm. For the reason that the value of maximum generation (Max_Gen = FEs/NP) is smaller than *js* (*js* = 6000) when the FEs is 1.2+e5, Table III only contains the data of HACC-D when FEs is 6e+5 and 3e+6.

Fig. 4 shows the convergence plots of *the two basic algorithms* (DECC-D with SaNSDE, DECC-D with JADE) and HACC-D algorithm with *js* = 6000 and *denum* = 100 on $f_2, f_3, f_6, f_7, f_{11}$ and f_{16}. These functions are the ones on which there is a large gap between the two basic algorithms or the one on which HACC-D have a significantly better performance than the two basic algorithms. In Fig. 4, HACC-D converges faster than the two basic algorithms on f_2, f_3 and f_{11} when the evaluation number is over 4×10^4. However, the convergence speed of the two basic algorithms is almost zero. In particular, HACC-D converges faster than the two basic algorithms on f_{16} when the evaluation number is over 4.5×10^4. On all these six functions, HACC-D almost keeps the same speed as the faster one of the two basic algorithm and still has a speed at the end stage of evolution. It proves the success of the hybrid adaptive optimization strategy in synthesizing the preponderance of the DE variants and its extra searching ability to some extent. Consequently, the hybrid adaptive optimization strategy succeeds to composite the superiority of JADE and SaNSDE and is efficient for large scale optimization problems.

V. CONCLUSION

In this paper, we proposed a novel technique named hybrid adaptive optimization strategy for large scale optimization problems. It integrates the preponderance of two most successful differential evolution algorithms, JADE and SaNSDE, as the subcomponent optimization algorithm of CC. For the class of problems on which the two algorithms with only one single subcomponent optimization algorithm (DECC-D with SaNSDE, DECC-D with JADE) have large performance gap, the hybrid adaptive optimization strategy showed significant success and managed to obtain the better results of the two basic algorithms. For the reason that the novel strategy allocates a certain time budget in the early learning stage, it has less time budget for the following evolution process. This make HACC-D has poorer performance than the two basic algorithms on some of the functions. Experimental results confirmed that HACC-D algorithm is capable of coalescing the advantages of JADE and SaNSDE as the subcomponent optimization algorithm of CC for large scale optimization problems.

Hybrid adaptive optimization method seems to be a promising approach for large scale optimization problems. But it is still in its infancy and the setting of the two control parameters, *js* and *denum*, is rough. Further research is desired in order to improve the hybrid adaptive mechanism and make it more suitable and robuster to more kinds of problems. In the meanwhile, there are still other kinds of EAs. It is promising to explore a new algorithm selection method in the subcomponent optimization process of CC to conquer large scale optimization problems

ACKNOWLEDGMENT

The authors would like to thank Mr. Omidvar for providing us with the source code of DECC-D and the anonymous reviewers.

TABLE III
EXPERIMENT RESULT OF CEC"2010 FUNCTIONS FOR 25 INDEPENDENT RUNS WITH 1000 DIMENSIONS.

1000D		f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
6.0E+05	Best	2.88E+02	2.99E+03	7.14E-01	7.45E+12	7.66E+07	6.59E+03	9.98E+07	4.61E+07	2.45E+08	1.27E+04
	Median	2.32E+03	3.11E+03	7.59E-01	1.46E+13	1.87E+08	8.84E+03	3.59E+08	1.24E+08	3.44E+08	1.34E+04
	Worst	1.60E+04	3.23E+03	8.15E-01	2.56E+13	2.68E+08	1.17E+04	9.19E+08	2.65E+08	3.99E+08	1.36E+04
	Mean	2.98E+03	3.11E+03	7.67E-01	1.50E+13	1.77E+08	8.99E+03	3.96E+08	1.40E+08	3.39E+08	1.33E+04
	StDev	3.12E+03	6.11E+01	2.86E-02	4.66E+12	5.36E+07	1.39E+03	1.92E+08	5.71E+07	3.56E+07	3.00E+02
3.0E+06	Best	0.00E+00	5.33E-15	2.84E-14	6.01E+11	9.05E+07	3.55E-09	8.26E-09	2.63E+02	2.47E+07	1.25E+04
	Median	0.00E+00	1.42E-14	3.20E-14	1.60E+12	2.17E+08	3.55E-09	2.22E-07	3.11E+07	3.30E+07	1.30E+04
	Worst	4.97E-26	2.49E-14	5.68E-14	2.70E+12	2.89E+08	3.55E-09	1.66E-06	3.72E+08	4.08E+07	1.36E+04
	Mean	1.99E-27	1.43E-14	3.45E-14	1.55E+12	1.96E+08	3.55E-09	3.87E-07	7.44E+07	3.32E+07	1.30E+04
	StDev	9.93E-27	5.77E-15	7.78E-15	4.80E+11	6.04E+07	5.59E-15	4.41E-07	8.86E+07	3.88E+06	2.39E+02
1000D		f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}
6.0E+05	Best	1.48E+01	6.08E+05	3.40E+03	8.10E+08	1.47E+04	8.83E+01	2.18E+06	2.56E+04	1.84E+07	4.30E+03
	Median	1.83E+01	8.39E+05	1.90E+04	9.34E+08	1.58E+04	1.03E+02	2.86E+06	4.81E+04	2.32E+07	4.72E+03
	Worst	5.20E+01	5.20E+06	5.63E+04	1.11E+09	1.65E+04	4.27E+02	9.18E+06	6.95E+04	2.84E+07	7.77E+03
	Mean	2.13E+01	1.33E+06	2.25E+04	9.30E+08	1.57E+04	1.29E+02	3.48E+06	4.72E+04	2.35E+07	4.90E+03
	StDev	8.06E+00	1.32E+06	1.33E+04	8.10E+07	4.41E+02	7.51E+01	1.80E+06	1.18E+04	2.19E+06	6.75E+02
3.0E+06	Best	6.39E-14	3.82E+03	7.58E+02	7.91E+07	1.43E+04	1.07E-13	4.23E+04	1.40E+03	1.60E+07	1.28E+03
	Median	7.82E-14	5.22E+03	1.10E+03	9.11E+07	1.57E+04	1.24E-13	5.92E+04	3.70E+03	1.83E+07	1.51E+03
	Worst	9.59E-14	4.61E+06	6.77E+03	1.06E+08	1.64E+04	4.85E-10	9.05E+06	1.16E+04	2.15E+07	1.80E+03
	Mean	7.82E-14	1.31E+06	1.96E+03	9.21E+07	1.56E+04	1.95E-11	1.42E+06	4.02E+03	1.87E+07	1.51E+03
	StDev	9.00E-15	1.98E+06	1.67E+03	8.55E+06	5.11E+02	9.69E-11	3.19E+06	2.17E+03	1.44E+06	1.29E+02

REFERENCES

[1] M. A. Potter and K. A. D. Jong, "A cooperative coevolutionary approach to function optimization," in P*roc. of the Third Conference on Parallel Problem Solving from Nature*, vol. 2, 1994, pp. 249-257.

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

[3] T. Ray and X. Yao, "A cooperative coevolutionary algorithm with correlation based adaptive variable partitioning," in Proc. *of IEEE Congress on Evolutionary Computation*, May 2009, pp. 983 – 989.

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

[5] W. Cheng, T. Weise, Z. Yang, K. Tang, "Large-Scale Global Optimization Using Cooperative Coevolution with Variable Interaction Learning," in Parallel Problem Solving from Nature – PPSN XI. vol. 6239, R. Schaefer, C. Cotta, J. Kolodziej, and G. Rudolph, Eds., ed: Springer Berlin / Heidelberg, 2010, pp. 300-309.

[6] E. Sayed, D. Essam, and R. Sarker. "Dependency identification technique for large scale optimization problems." Evolutionary Computation (CEC), 2012 IEEE Congress on. IEEE, 2012.

[7] D. Molina, M. Lozano, and F. Herrera. "MA-SW-Chains: Memetic algorithm based on local search chains for large scale continuous global optimization." *Evolutionary Computation (CEC)*, *2010 IEEE Congress on*. IEEE, 2010.

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

[9] J. Liu and J. Lampinen, "A fuzzy adaptive differential evolution algorithm," *Soft Comput.: Fusion Found., Methodologies Applicat.*, vol.9, no. 6, pp. 448-462, 2005.

[10] Z. Yang, K. Tang, and X. Yao, "Self-adaptive differential evolution with neighborhood search," in *Proc. of IEEE World Congress on Computational Intelligence*, June 2008, pp. 1110-1116.

[11] J. Zhang and A. C. Sanderson, "JADE: Adaptive Differential Evolution with Optional External Archive," *IEEE Transactions on Evolutionary Computation 13(5)*, pp. 945-958, 2008.

[12] M. N. Omidvar, X. Li, Z. Yang, and X. Yao, "Cooperative co-evolution for large scale optimization through more frequent random grouping," in *Proc. of the 2010 IEEE Congress on Evolutionary Computation,* 2010, pp.1-8.

[13] Z. Yang, J. Zhang, K. Tang, X. Yao , "An Adaptive Coevolutionary Differential Evolution Algorithm for Large-scale optimization," in *Proc. of the 2009 IEEE Congress on Evolutionary Computation,* 2009, pp. 102-109.

[14] Y. Liu, X. Yao, Q. Zhao, and T. Higuchi, "Scaling up fast evolutionary programming with cooperative coevolution," in *Proc of the 2001 Congress on Evolutionary Computation*, 2001, pp. 1101-1108.

[15] D. Sofge, K. D. Jong, and A. Schultz, "A blended population approach to cooperative coevolution fordecomposition of complex problems," in *Proc. of IEEE World Congress on Computational Intelligence*, 2002, pp. 413-418.

[16] F. van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation 8 (3)*, pp. 225-239, 2004.

[17] Y. Shi, H. Teng, , and Z. Li, "Cooperative co-evolutionary differential evolution for function optimization," in *Proc. of the First International Conference on Natural Computation*, 2005, pp. 1080-1088.

[18] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp, "A Racing Algorithm for Configuring Metaheuristics." GECCO. Vol. 2. 2002, pp. 11-18 .

[19] B. Yuan, and M. Gallagher. "Statistical racing techniques for improved empirical evaluation of evolutionary algorithms." Parallel Problem Solving from Nature-PPSN VIII. Springer Berlin Heidelberg, 2004.

[20] Z. Yang, X. Yao, J. He, "Making a difference to differential evolution," in *Advances in metaheuristics for hard optimization*, Springer Berlin Heidelberg, 2008, pp. 397-414.

[21] E. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello, "A comparative study of differential evolution variants for global optimization," in *Proc. Genetic Evol. Comput. Conf.*, Seattle, WA, Jul. 2006, pp. 485-492.

[22] R. Mendes, I. Rocha, E. C. Ferreira, and M. Rocha, "A comparison of algorithms for the optimization of fermentation processes," in *Proc. IEEE Congr. Evol. Comput.*, Vancouver, BC, Jul. 2006, pp. 2018-2025.

[23] K. Tang, X. Li, P. N. Suganthan, Z. Yang, and T. Weise, "Benchmark functions for the cec'2010 special session and competition on large scale global optimization," NICAL, USTC, China, Tech. Rep., 2009, http://nical.ustc.edu.cn/cec10ss.php

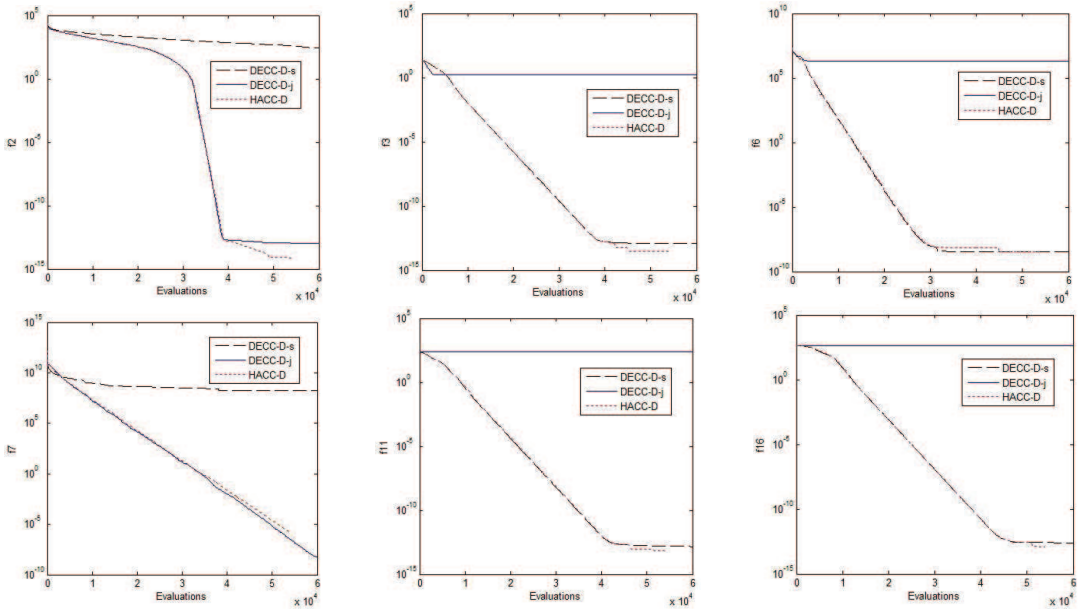

Fig. 4. Convergence plots of f_2, f_3, f_6, f_7, f_{11} and f_{16} for DECC-D with SaNSDE, DECC-D with JADE and HACC-D with *js* = 6000 and *denum* = 100.