Cooperative Co-evolution for large scale optimization through more frequent random grouping

Conference Paper · August 2010

DOI: 10.1109/CEC.2010.5586127 · Source: IEEE Xplore

4 authors, including:

Mohammmad Nabi Omidvar
RMIT University
26 PUBLICATIONS **814** CITATIONS

SEE PROFILE

Xiaodong Li
RMIT University
202 PUBLICATIONS **5,444** CITATIONS

SEE PROFILE

Zhenyu Yang
Zhuiyi Technology Inc.
22 PUBLICATIONS **1,995** CITATIONS

SEE PROFILE

Cooperative Co-evolution for Large Scale Optimization Through More frequent Random Grouping

Mohammad Nabi Omidvar, *Member, IEEE* and Xiaodong Li, *Senior Member, IEEE* and
Zhenyu Yang and Xin Yao, *Fellow, IEEE*

***Abstract*— In this paper we propose three techniques to improve the performance of one of the major algorithms for large scale continuous global function optimization. Multilevel Cooperative Co-evolution (MLCC) is based on a Cooperative Co-evolutionary framework and employs a technique called *random grouping* in order to group interacting variables in one subcomponent. It also uses another technique called *adaptive weighting* for co-adaptation of subcomponents. We prove that the probability of grouping interacting variables in one subcomponent using random grouping drops significantly as the number of interacting variables increases. This calls for more frequent random grouping of variables. We show how to increase the frequency of random grouping without increasing the number of fitness evaluations. We also show that adaptive weighting is ineffective and in most cases fails to improve the quality of found solution, and hence wastes considerable amount of CPU time by extra evaluations of objective function. Finally we propose a new technique for self-adaptation of the subcomponent sizes in CC. We demonstrate how a substantial improvement can be gained by applying these three techniques.**

I. Introduction

Many Evolutionary Algorithms(EAs) have been used for optimization problems, but the performance of these algorithms deteriorate as the dimensionality of problem increases, commonly referred to as the curse of dimensionality. The problem of finding the global optimum becomes even harder when some or all of the decision variables have interaction amongst themselves. This class of problems are called non-separable problems. Variable interaction in large scale problems drastically increases the total number of function evaluations in order to find a reasonable solution. Large-scale non-separable problems are considered as very difficult type of optimization and they usually require a large number of fitness evaluations.

The expensive nature of these problems gets magnified in dealing with real world problems specially in engineering. In many real world problems the cost of evaluating the objective function involves interaction with other modules such as simulation software. RoboCup simulation software is a perfect example[1]. In Evolutionary Robotics, Multidisciplinary Design Optimization(MDO), Shape Optimization, and other areas the cost of only one evaluation of the objective function is more than a few minutes if not a few hours. The expensive nature of objective function specially in real world problems along with the difficult nature of large scale non-separable problems demands new ways of cutting down the total number of fitness evaluations.

Many techniques have been proposed for solving large scale problems. Cooperative Co-evolution(CC) proposed by Potter et al.[2] uses a divide-and-conquer approach to divide the decision variables into subpopulation of smaller sizes and each of these subpopulations is optimized with a separate EA in a round robin fashion. CC seems to be a promising framework for large scale problems, but its performance degrades when applied to non-separable problems[2]. In an ideal setting all the interacting variables should be grouped in one subcomponent in order to enhance the performance of optimization, but in real world problems there is almost no prior information about how the variables are interacting. This arises the need for more sophisticated techniques capable of capturing the interacting parameters through the course of evolution and grouping them together in one single subpopulation.

MLCC[3] is a new techniques for large scale non-separable function optimization which extends another algorithm called DECC-G[4] by self-adapting the subcomponent sizes. DECC-G relies on random grouping of decision variables into subcomponents in order to increase the probability of grouping interacting variables in non-separable problems. It also evolves a weight vector for co-adaptation of subcomponents for further improving the solutions. In this paper we investigate the internal mechanism of these algorithms and demonstrate two major bottlenecks that degrades the performance of these algorithms. We also propose some techniques for further improving the performance of MLCC. First we show that more frequent random grouping could result in finding a solution with fewer number of fitness evaluations without sacrificing solution quality. Secondly we show that adaptive weighting does not play a major role in the whole process and in most cases fails to improve the solution and wastes considerable number of fitness evaluations. Overall, we demonstrate the followings in this paper:

- Extend the theoretical background of random grouping to include more than two interacting variables.
- Show that more frequent random grouping is beneficial, specially when there are more than two interacting

M. N. Omidvar and X. Li are with the Evolutionary Computing and Machine Learning Group (ECML), the School of Computer Science and IT, RMIT University, VIC 3001, Melbourne, Australia (emails: momidvar@cs.rmit.edu.au, xiaodong.li@rmit.edu.au).

X. Yao is with the Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), the School of Computer Science, University of Birmingham, Birmingham B15 2TT, UK (e-mail: x.yao@cs.bham.ac.uk).

Z. Yang is with Nature Inspired Computation and Application Laboratory, the Department of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China. (email: zhyuyang@mail.ustc.edu.cn)

978-1-4244-8126-2/10/$26.00

variables.

- Identifying two major bottlenecks in MLCC and show how to increase the frequency of random grouping without increasing the total number of fitness evaluations.
- Demonstrate that adaptive weighting is ineffective, wasting valuable fitness evaluations, and show that how the overall performance will be significantly improved by simply disabling this feature.
- A simpler and more intuitive and yet more efficient alternative to self-adaptation of subcomponent sizes which is used in MLCC. This new algorithm is called DECC-ML
- Showing that DECC-ML shows faster convergence compared to previous techniques. This faster convergence results in saving up to $\frac{2}{3}$ of fitness evaluations which is a significant improvement specially in expensive optimization problems.

The organization of the rest of this paper is as follows: Section II describes the preliminaries and background information, Section III describes the new techniques for cutting down the number of fitness evaluations. Section IV demonstrates and analyzes the experimental results, and finally section V summarizes this paper.

II. Preliminaries

A. Cooperative Co-evolution

Divide-and-conquer is an effective technique in solving complex problems. Cooperative Co-evolution [2] uses a similar technique to decompose a complex problem into several simpler sub-problems.

Potter and De Jong made the first attempt to incorporate CC into Genetic Algorithm for function optimization[2]. The success of CC attracted many researchers to incorporate CC into other evolutionary techniques such as Evolutionary Programming [5], Evolutionary Strategies[6], Particle Swarm Optimization[7], and Differential Evolution [4], [8].

The original CC decomposes a n-dimensional decision vector into n subcomponents and optimizes each of the subcomponents using GA in a round robin fashion. This algorithm was called CCGA.

The first attempt for applying CC to large scale optimization was made by Liu et al. using Fast Evolutionary Programming with Cooperative Co-evolution(FEPCC)[5] where they tackled problems with up to 1000 dimensions, but it converged prematurely for one of the non-separable functions, confirming that Potter and De Jong decomposition strategy is ineffective in dealing with variable interaction.

van den Bergh and Engelbrecht[7] were first to apply CC to PSO. They developed two dialects of Cooperative PSO(CPSO) based on the original Potter's framework, but unlike the original CCGA[2] they divided a n-dimensional problem into m s-dimensional subcomponents as depicted in Figure 1.

CPSO follows the following steps [5], [2].

1) Divide the variables of the objective function into m-dimensional subcomponents.

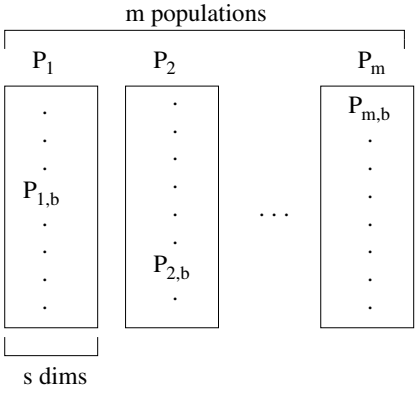


Fig. 1. The n-dimensional objective vector is divided into m s-dimensional subcomponents. $P_{m,b}$ denotes the best individual in m^{th} subcomponent.

2) Optimize each of the *m* subcomponents in a round-robin fashion with a certain EA. Note that the number of evaluations is predetermined.
3) Stop the evolutionary process once the halting criteria is satisfied or the maximum number of evaluations is exceeded.

Every individual in a subcomponent is evaluated by concatenating it with the best-fit individuals in the rest of the subcomponents to form what is called a *context vector*[7]. This context vector contains all the parameters required by the objective function and is fed into it for fitness evaluation. This is actually where the cooperation happens.

One major drawback of CPSO is that its performance degrades rapidly as the dimensionality of the problem increases, mainly due to exponential growth of the size of search space. This problem is magnified when applied to non-separable problems due to parameter interactions.

CC has also been applied to Differential Evolution in [4], [8]. Shi et al. [8] used splitting-in-half strategy where they divide the variables into two equally sized subcomponents each of which is optimized using a separate DE. Their decomposition strategy does not scale up efficiently as the number of dimensions increases. Yang et al.[4] made the first attempt to develop a more systematic way of dealing with variable interactions by random grouping of decision variables into different subcomponents. They have shown that random grouping of variables will increase the probability of grouping interacting variables in one subcomponent. This is further explained in Section II-C.

B. Differential Evolution

Differential Evolution (DE)[2] is a relatively new EA for global optimization. Despite its simplicity it has shown to be very effective on a set of benchmark functions [9].

DE[10] is designed to be less greedy compared to other EA techniques[9] such as PSO [11] which makes it a good choice for large scale optimization. In large scale optimization, due to the vastness of the search space, more exploration is needed in order to find the global optimal point, so

more greedy techniques run the risk of finding a suboptimal solution.

DE's control parameters are problem dependent and hard to determine[12], [13]. Self-Adaptive Differential Evolution with Neighborhood Search(SaNSDE)[14] self-adapts crossover rate CR, scaling factor F, and mutation strategy. It has been shown that SaNSDE performs significantly better than other similar DE algorithms.

C. Random Grouping and Adaptive Weighting

In the framework proposed by Yang et al. [4], following a CC approach, the problem is decomposed into m s-dimensional subcomponents, and each subcomponent is optimized using a certain EA. Their method differs from ordinary CC approaches in two major ways. First they use random grouping of decision variables in order to increase the probability of grouping two interacting variables in one subcomponent, and secondly, the co-adaptation of subcomponents which is done using a technique known as Adaptive Weighting.

The motivation for random grouping is that in most real-world non-separable problems, only a proportion of variables interact with each other, so if an optimization algorithm manages to group highly dependent variables in one subcomponent there will be a better chance to further improve the performance of the algorithm. Since there is no prior information about how the variables are interacting Yang et al.[4] showed that by random grouping of the decision variables one can increase the probability of grouping two interacting variables in one subcomponent for at least some predetermined number of cycles. They demonstrated that random grouping results in a probability of 96.62% in grouping two interacting variables together for at least two cycles in a CC setting with 10 subcomponents and a total of 1000 decision variables. However, it remains unclear what the probability will be if more than 2 variables interacting with each other, which is more likely the case in most optimization problems. In Section II-C we show how the probability of grouping interacting variables drops significantly when there are more than two interacting variables and we also show that given the same number of evaluations how to increase the probability of grouping more than two variables together.

In adaptive weighting, a numeric weight value is applied to each subcomponent. All of these weight values form a vector called *weight vector*. This weight vector is optimized using a separate optimizer for some predetermined Fitness Evaluations(FEs). The motivation behind adaptive weighting is to co-adapt interdependent subcomponents. It is obvious that optimizing the weight vector is far simpler than the original problem because the dimensionality of the wight vector with only m variables is smaller than the original problem with $m \times s$ variables.

Yang et al. [4] outline the steps of weight vector co-adaptation as follows.

1) set $i = 1$ to start a new *cycle*.
2) Randomly split a n-dimensional objective vector into m s-dimensional vector. This essentially means that any variable has equal chance of being assigned to any of the subcomponents.
3) Optimize the i^{th} subcomponent with a certain EA for a predefined number of FEs.
4) If $i < m$ then $i++$, and go to Step 3.
5) Construct a weight vector and evolve it using a separate EA for the best, worst, and a random member of the current population.
6) Stop if the maximum number of FEs is reached or go to Step 1 for the next *cycle*

This scheme that uses a cooperative co-evolutionary EA with weight co-adaptation was named EACC-G in [4]. Since SaNSDE[14] is used as the subcomponent optimizer in Step 3 the algorithm is called DECC-G.

D. Multilevel Cooperative Co-evolution

One major problem with DECC-G is determining the size of subcomponents. This parameter is problem dependent and is hard to determine. Multilevel Cooperative Co-evolution(MLCC) [3] is an extension of DECC-G that dynamically self-adapts the subcomponent sizes.

MLCC uses a more flexible way of choosing the subcomponent sizes by choosing a group size from a set $S = \{s_1, ..., s_t\}$ of predefined group sizes. The selection probability is calculated based on the performance history of each of decomposers through the course of evolution.

MLCC uses a sophisticated formula for calculating the selection probability of decomposers at the beginning of each cycle. The formula contains some arbitrary constants which are based on some empirical studies. In this paper we propose a new technique for self-adaptation of the subcomponent sizes which simply choose randomly a group size from a predetermined set. DECC-ML shows substantial improvement compared to MLCC.

III. Proposed Techniques

In this paper we propose two techniques for optimizing the performance of DECC-G, and an alternative to MLCC for self-adapting subcomponent sizes which is easier to implement and more intuitive, and yet more efficient. In our algorithm we perform the random decomposition of objective vector at every iteration which results in more frequent random grouping.

A. More Frequent Random Grouping

In their paper Yang et al. proved that random grouping of decision variables and grouping them together at the beginning of each cycle increases the chance of putting two interacting variables in the same subcomponent. In [4] they used 50 cycles and have shown that the probability of grouping two parameters together at least for two iterations would be 96.62% in a CC setting with 10 subcomponents with a total of 1000 variables.

We further generalized their theorem here to any number of interacting variables. Equation(1) calculates the probability of grouping v interacting variables in the same subcomponent for at least k cycles.

Theorem 1. *Given N cycles, the probability of assigning v interacting variables $x_1, x_2, ..., x_v$ into one subcomponent for at least k cycles is:*

$$P(X \geq k) = \sum_{r=k}^{N} \binom{N}{r} \left(\frac{1}{m^{v-1}}\right)^r \left(1 - \frac{1}{m^{v-1}}\right)^{N-r} \quad (1)$$

where N is the number of cycles, v is the total number of interacting variables, m is the number of subcomponents, and the random variable X is the number of times that v interacting variables are grouped in one subcomponent and since we are interested in the probability of grouping v interacting variables for at least k cycles, X takes the values greater than or equal to k. k is also subject to the following condition $k \leq N$.

Proof 1. *A variable can be assigned to a subcomponent in m different ways, and since there are v interacting variables, the probability of assigning all of the interacting variables into one subcomponent would be:*

$$p_{sub} = \underbrace{\frac{1}{m} \times ... \times \frac{1}{m}}_{v \text{ times}} = \frac{1}{m^v}$$

Since there are m different subcomponents, the probability of assigning all v variables to any of the subcomponents would be:

$$p = m \times p_{sub} = \frac{m}{m^v} = \frac{1}{m^{v-1}}$$

There are a total of N independent random decompositions of variables into m subcomponents, so using a binomial distribution the probability of assigning v interacting variables into one subcomponent for exactly r times would be:

$$\begin{aligned} P(X = r) &= \binom{N}{r} p^r (1-p)^{N-r} \\ &= \binom{N}{r} \left(\frac{1}{m^{v-1}}\right)^r \left(1 - \frac{1}{m^{v-1}}\right)^{N-r} \end{aligned}$$

Thus,

$$P(X \geq k) = \sum_{r=k}^{N} \binom{N}{r} \left(\frac{1}{m^{v-1}}\right)^r \left(1 - \frac{1}{m^{v-1}}\right)^{N-r}$$

□

Given $n = 1000$, $m = 10$, $N = 50$ and $v = 4$, we have:

$$P(X \geq 1) = 1 - P(X = 0) = 1 - \left(1 - \frac{1}{10^3}\right)^{50} = 0.0488$$

which means that over 50 cycles, the probability of assigning 10 interacting variables into one subcomponent for at least 1 cycle is only 0.0488. As we can see this probability is very small, and it will be even less if there are more interacting variables. Figure 2 shows how the probability of grouping interacting variables for at least one cycle drops significantly as the number of interacting variables increases. The solid line shows how the probability will change by v for 50 cycles and the dashed line shows how the probability will change when there are $1e4$ cycles. We can see from the graph that the probability of grouping 5 variables for at least once using 50 cycles is close to zero whereas the probability of grouping the same number of interacting variables for at least once will increase to approximately 60% using $1e4$ cycles. Note that the number of fitness evaluations will be the same by applying the techniques described later in this section.

Figure 3 shows the effect of increasing N, which is the frequency of random grouping for different number of interacting variables. It is evident that higher rate of random grouping will increase the probability of grouping interacting variables, regardless of the number of them. So more frequent random grouping will not only results in higher probability in grouping interacting variables, but also increases the efficiency of the algorithms in dealing with more than two interacting variables. Our studies on DECC-G and MLCC revealed that the frequency of random grouping could be significantly increased without increasing the total number of fitness evaluations. In order to maximize the frequency of random grouping the subcomponent optimizers should run for only one iteration which is equivalent to $N_{popsize}$ number of fitness evaluations, where $N_{popsize}$ is the population size of the EA.

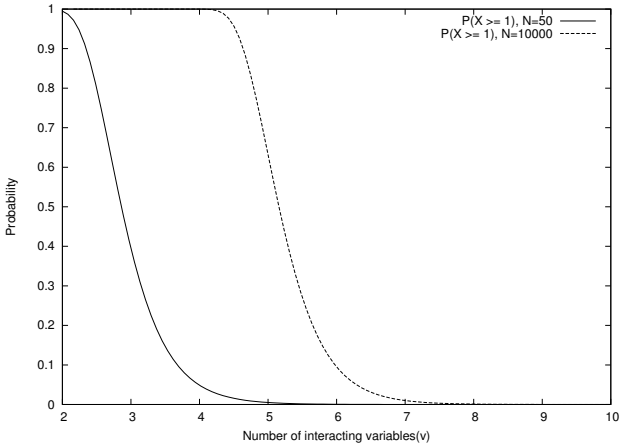


Fig. 2. Increasing v, the number of interacting variables will significantly decrease the probability of grouping them in one subcomponent, given $n = 1000$ and $m = 10$.

In their experiments, Yang et al. used only 50 cycles with $5e6$ FEs while there is a potential for approximately 3800 cycles. This setting is counterintuitive because according to the above discussion, a higher number of random grouping will result in higher probability in grouping any two interacting variables in one subcomponent and also better efficiency in dealing with more than two interacting variables.

B. Removing Adaptive Weighting

Further investigation on DECC-G have shown that Adaptive Weighting is not effective, and the computational resource can be spent on using more frequent random grouping instead.

Although in theory it seems to be a good way of reducing the dimensionality of the original problem, in practice its

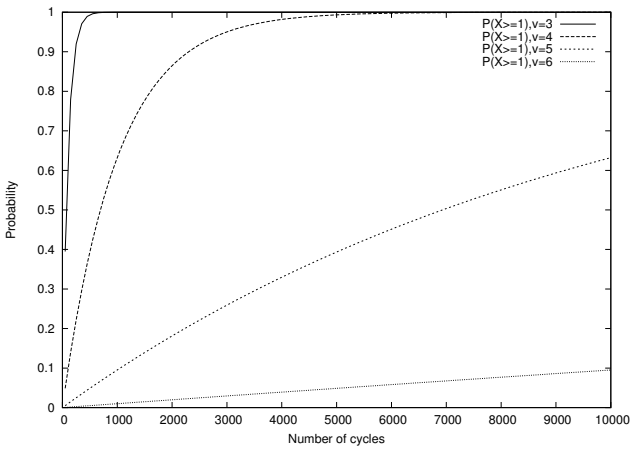


Fig. 3. Increasing N, the number of cycle increases the probability of grouping v number of interacting variables in one subcomponent.

role is insignificant in the whole optimization process. In order to demonstrate this we counted the number of times that adaptive weighting subcomponent manages to find a solution, and based on our experiments we realized that in most cases adaptive weighting fails to find a better solution. This essentially means that a significant number of fitness evaluations are wasted by adaptive weighting. In order to demonstrate this difference we modified DECC-G and disabled the adaptive weighting subcomponent and conducted an experiment with the same settings as DECC-G with a total of 50 cycles. We called this new variant DECC-NW. Tables IV, V, and VI summarize the results of running these two algorithms over the same benchmark functions.

C. Self-adaptation of subcomponent sizes

As we mentioned earlier we also proposed a simpler technique for self-adapting the subcomponent sizes. In our method we preserved the same decomposer set S as it was used in MLCC, but instead of using the sophisticated formula for probability calculation in choosing a decomposer we simply use a uniform random number generator for selecting a decomposer from the set S. This only happens when there is no improvement in the fitness between the previous and the current cycles.

IV. Empirical Results

A. Experiment Setup

We evaluated our proposed techniques with CEC'08 test suite which was proposed on CEC'2008 special session for Large Scale Global Optimization[15].

This test suite consists of seven functions which are summarized in Table I.

We ran each algorithm for 25 independent runs for 100, 500, and 1000 dimensions and the mean and standard deviation of the best fitness values over 25 runs was recorded. The population size was set to 50, and the maximum number of fitness evaluations(FEs) was calculated by the following formula, $FEs = 5000 \times D$, where D is the number of dimensions. For the subpopulation sizes we used the same set as it was used in[3], $S = \{5, 10, 25, 50, 100\}$. The experimental setup is summarized in Table II. For better clarity all the algorithms mentioned in this paper are summarized in Table III

TABLE I
SUMMARY OF THE 7 CEC'08 TEST FUNCTIONS

Func	Description	Modality
f_1	Shifted Sphere Function	Unimodal
f_2	Shifted Schwefel's Problem 2.21	Unimodal
f_3	Shifted Rosenbrock's Function	Multimodal
f_4	Shifted Rastrigin's Function	Multimodal
f_5	Shifted Griewank's Function	Multimodal
f_6	Shifted Ackley's Function	Multimodal
f_7	FastFractal "DoubleDip" Function	Multimodal

TABLE II
EXPERIMENTAL SETUP PARAMETERS

Parameter	**Value**
Dimensions	$D = \{100, 500, 1000\}$
FEs	$5000 \times D$ $\{5e+5, 2.5e+6, 5e+6\}$
Population size	$N_{popsize} = 50$
Subpopulation sizes	$S = \{5, 10, 25, 50, 100\}$
Number of runs	25

TABLE III
SUMMARY OF ALGORITHMS DESCRIBED IN THIS PAPER. THOSE WITHOUT CITATION ARE PROPOSED IN THIS PAPER.

Algorithm	Description
DECC-G[4]	Cooperative Co-evolutionary DE with random grouping and adaptive weighting.
DECC-NW	Similar to DECC-G with the adaptive weighting subcomponent disabled.
DECC	Cooperative Co-evolutionary DE without adaptive weighting and running the subcomponent optimizers for only one iteration.
DECC-ML	Similar to DECC, but uses a uniform selection for self-adapting the subcomponent sizes.
MLCC[3]	Similar to DECC-G, but it self-adapts the subcomponent sizes using historical performance of different decomposers.

B. Analysis of Results

The result of 25 independent runs for DECC-ML is listed in Tables VII, and VIII for 1000 dimensions. These tables show the progress of the algorithm and is based on provided template for CEC'08 competition for Large Scale Global Optimization[15]. The summary Tables IV, V, and VI compares all five algorithms for 100, 500, 1000 dimensions respectively.

According to Tables VII, and VIII, DECC-ML shows faster convergence than MLCC on 6 out of 7 functions. A deeper analysis of the 25 runs revealed that existence of two outliers in results of DECC-ML increased the mean significantly on f_5. Running Wilcoxon rank-sum test using 99% confidence interval has shown that DECC-ML is also significantly better than MLCC on f_5. The p-values of Wilcoxon rank-sum test is recorded in Table VI.

The global optimum point for f_7 is unknown, but from the Tables VII, and VIII it is clear that the algorithm has

a steady progress in the improvement of the fitness value. This shows the ability of DECC-ML to converge even on a difficult function such as f_7 which is highly multimodal and has a rugged surface. It is noteworthy that DECC-ML found the absolute global point in all 25 runs for f_4 and in at least 19 out of 25 runs for f_1.

By comparing DECC and DECC-G we can see that DECC found a better solution using same number of fitness evaluations in 6 out of 7 benchmark functions which shows that more frequent random grouping of variables yields better performance. This trend is almost the same for 100, 500, and 1000 dimensions.

The summary Tables IV, V, and VI also confirm our speculation about high failure rate of adaptive weighting subcomponent of DECC-G.

From the results in Table VI, V, IV, we can see that a significant number of fitness evaluations could be saved for using more frequent random grouping in order to increases the performance of the algorithm, we can conclude that using the saved fitness evaluations for increasing the frequency of random grouping will improve the performance of the algorithm even further. We incorporated both of these changes into DECC-G,and created another version called DECC. Experimental results shows that DECC outperforms DECC-G over 6 out of 7 benchmark functions.

DECC-NW and DECC-G are identical except that in DECC-NW the adaptive weighting subcomponent is disabled. We tested both algorithms for 50 cycles and equal number of fitness evaluations. By looking at the Tables IV, V, and VI we can observe that DECC-NW converged faster than DECC-G in 6 out of the 7 functions.

Tables IV, V, VI show that DECC-ML outperforms MLCC in 6 out of 7 test functions which is quite substantial. The same trend continues over all dimensions which shows the better scalability of DECC-ML.

Another observation is DECC-ML's faster convergence behavior than MLCC for most of the test functions. This behavior is specially clear from Figures 4(a), 4(e), 4(f). For functions f_1, f_5, f_6 a solution is found with almost the same quality with only half of the FEs. This is a very valuable property specially in real world problems where evaluation of the fitness function is very costly. DECC-ML also shows a quicker convergence over the rest of test functions but the trend is less significant compared to f_1, f_5, f_6.

TABLE IV

RESULTS OF DIFFERENT ALGORITHMS OVER 100 DIMENSIONS(AVERAGE OVER 25 RUNS). BEST RESULTS ARE HIGHLIGHTED IN BOLD.

	DECC	DECC-G	DECC-ML	DECC-NW	MLCC
f_1	**2.7263e-29**	1.1903e-08	5.7254e-28	8.9824e-28	6.8212e-14
f_2	5.4471e+01	6.0946e+01	**2.7974e-04**	5.9949e+01	2.5262e+01
f_3	1.4244e+02	4.6272e+02	1.8871e+02	**1.2959e+02**	1.4984e+02
f_4	5.3370e+01	1.1783e+02	**0**	4.5768e+00	4.3883e-13
f_5	2.7589e-03	3.7328e-03	3.6415e-03	7.3233e-03	**3.4106e-14**
f_6	2.3646e-01	1.0365e+00	**3.3822e-14**	1.2224e+00	1.1141e-13
f_7	-9.9413e+02	-1.2666e+03	**-1.5476e+03**	-1.3967e+03	-1.5439e+03

TABLE V

RESULTS OF DIFFERENT ALGORITHMS OVER 500 DIMENSIONS(AVERAGE OVER 25 RUNS). BEST RESULTS ARE HIGHLIGHTED IN BOLD.

	DECC	DECC-G	DECC-ML	DECC-NW	MLCC
f_1	**8.0779e-30**	1.0326e-25	1.6688e-27	2.4479e-27	4.2974e-13
f_2	4.0904e+01	7.6080e+01	**1.3396e+00**	7.2776e+01	6.6663e+01
f_3	6.6822e+02	1.4295e+03	**5.9341e+02**	1.2499e+03	9.2466e+02
f_4	1.3114e+02	5.6116e+00	**0**	5.2932e+00	1.7933e-11
f_5	2.9584e-04	4.1332e-03	1.4788e-03	2.7584e-02	**2.1259e-13**
f_6	**6.6507e-14**	1.8778e+00	1.2818e-13	1.0634e+00	5.3433e-13
f_7	-5.5707e+03	-6.0972e+03	**-7.4582e+03**	-6.9403e+03	-7.4350e+03

TABLE VI

RESULTS OF DIFFERENT ALGORITHMS OVER 1000 DIMENSIONS(AVERAGE OVER 25 RUNS). BEST RESULTS ARE HIGHLIGHTED IN BOLD.

	DECC	DECC-G	DECC-ML
f_1	**1.2117e-29**	3.8745e-27	5.1750e-28
f_2	4.2729e+01	7.4234e+01	**3.4272e+00**
f_3	1.2673e+03	2.6306e+03	**1.0990e+03**
f_4	2.4498e+02	1.0666e+01	**0**
f_5	2.9584e-04	7.8435e-03	9.8489e-04
f_6	**1.3117e-13**	2.3475e+00	2.5295e-13
f_7	-1.4339e+04	-1.3015e+04	**-1.4757e+04**

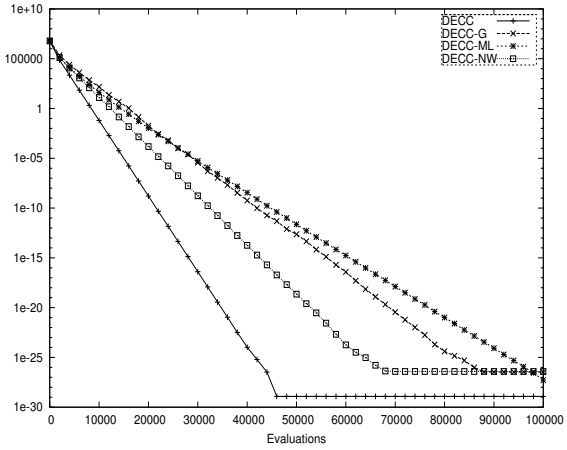
	DECC-NW	MLCC	DECC-ML,MLCC MWW rank-sum test
f_1	3.8145e-27	8.4583e-13	2.851e-06
f_2	7.2561e+01	1.0871e+02	6.535e-06
f_3	2.5411e+03	1.7986e+03	6.535e-06
f_4	7.7209e+00	1.3744e-10	6.482e-06
f_5	1.4976e-02	**4.1837e-13**	1.161e-03
f_6	1.7613e+00	1.0607e-12	6.506e-06
f_7	-1.41679e+04	-1.4703e+04	6.530e-06

V. CONCLUSION

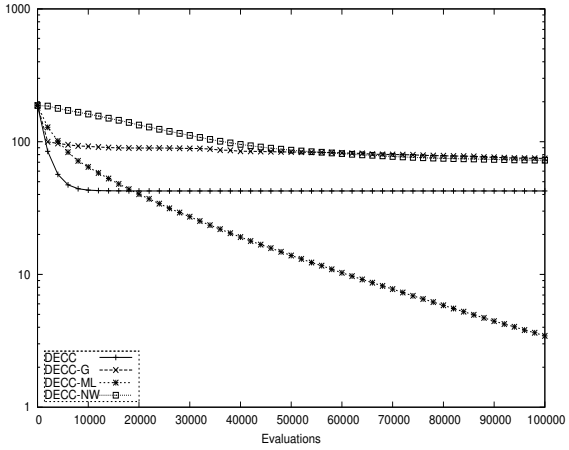
In this paper we proposed three techniques for further refinement of MLCC aiming to substantially reduce its computational cost. We have shown that more frequent random grouping will result in faster convergence without sacrificing solution quality due to increased probability in grouping interacting variables in a subpopulation. More frequent random grouping will also increase the efficiency of the algorithms in dealing with problems with more interacting variables. We have also shown that Adaptive Weighting is not as efficient as it was initially thought in reducing the dimensionality of the problem and in most cases fails to improve the fitness value. This will waste a considerable amount of fitness evaluations which might be better used for more effective random grouping and co-evolution of subcomponents.

We have also proposed an alternative technique for self-adapting the subcomponent sizes in CC framework and have shown that this simple technique is very effective.

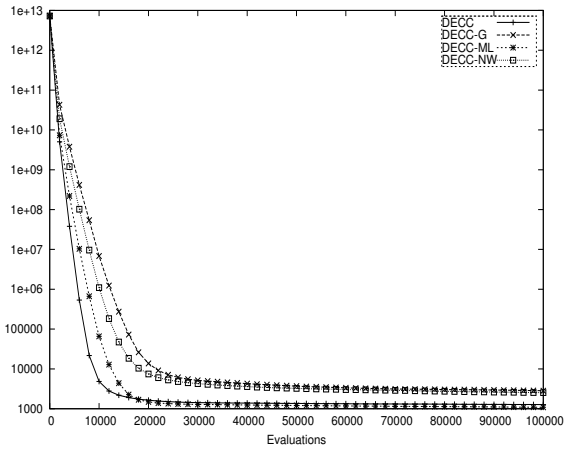
Finally we proposed an algorithm called DECC-ML which uses the new uniform selection of subcomponent sizes along with more frequent random grouping. Experimental results show that this new algorithm outperforms MLCC in most cases with a considerable difference. Fast convergence of DECC-ML allows for saving a significant amount of fitness evaluations for most of the functions. This saves considerable amount of CPU time specially in real-world problems with costly objective functions. The findings of this research is not limited to the CEC'2008 benchmark functions. We also observed similar behavior on CEC'2010 benchmark



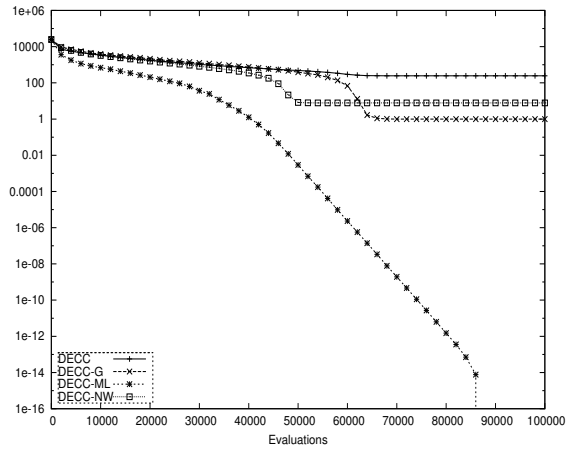
(a) f_1



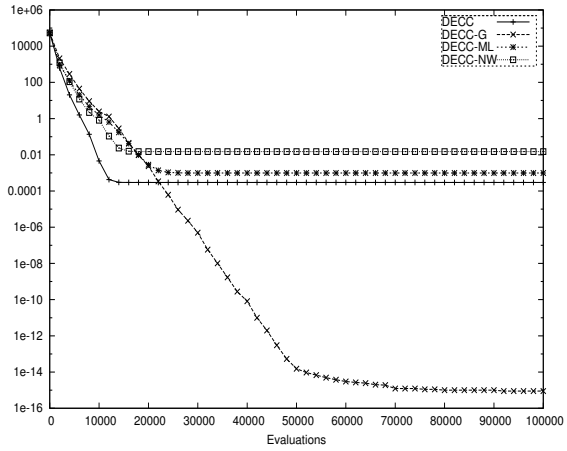
(b) f_2



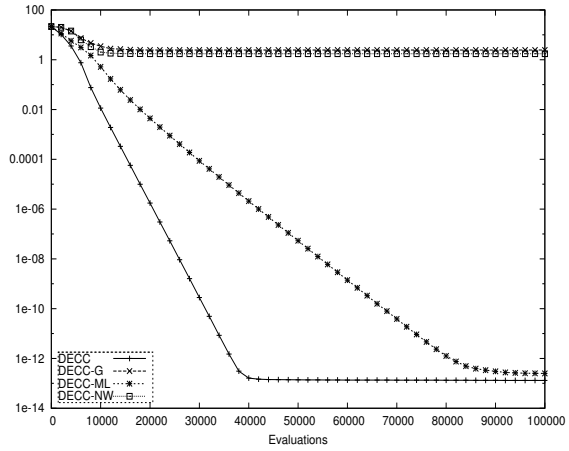
(c) f_3



(d) f_4



(e) f_5



(f) f_6

Fig. 4. Convergence plots for $f_1 - f_6$ with 1000 dimensions

TABLE VII

EXPERIMENTAL RESULTS OF 25 INDEPENDENT RUNS FROM DIFFERENT STAGES OF EVOLUTION($\frac{FEs}{100}$, $\frac{FEs}{10}$,FEs).

1000D		f_1	f_2	f_3	f_4
5e+4	1^{st}	1.7161e+05	1.1887e+02	6.6224e+09	4.0066e+03
	7^{th}	4.1466e+05	1.3322e+02	3.4286e+10	5.4659e+03
	13^{th}	4.7111e+05	1.3836e+02	4.0710e+10	5.6821e+03
	19^{th}	4.8698e+05	1.6499e+02	9.6494e+10	7.1419e+03
	25^{th}	5.1115e+05	1.8057e+02	1.0925e+11	8.3134e+03
	M	4.3797e+05	1.4607e+02	5.9240e+10	6.1375e+03
	Std	7.7341e+04	1.8315e+01	3.5704e+10	1.3938e+03
5e+5	1^{st}	9.1964e-01	3.0613e+01	1.0463e+04	3.0303e+02
	7^{th}	2.2217e+01	3.5070e+01	3.2238e+04	3.5180e+02
	13^{th}	3.0578e+01	5.5145e+01	4.4875e+04	8.6533e+02
	19^{th}	3.2747e+01	9.6366e+01	1.2383e+05	8.9242e+02
	25^{th}	1.3197e+02	1.2751e+02	1.4550e+05	9.1424e+02
	M	4.1338e+01	6.4509e+01	6.9043e+04	6.8650e+02
	Std	4.0586e+01	3.2558e+01	4.7066e+04	2.6892e+02
5e+6	1^{st}	0.0000e+00	6.9941e-02	9.2453e+02	0.0000e+00
	7^{th}	0.0000e+00	9.0952e-01	9.8941e+02	0.0000e+00
	13^{th}	0.0000e+00	2.0454e+00	1.0675e+03	0.0000e+00
	19^{th}	0.0000e+00	2.8289e+00	1.1714e+03	0.0000e+00
	25^{th}	1.2937e-26	1.5571e+01	1.4235e+03	0.0000e+00
	M	5.1750e-28	3.4272e+00	1.0990e+03	0.0000e+00
	Std	2.5875e-27	4.4636e+00	1.3217e+02	0.0000e+00

TABLE VIII

EXPERIMENTAL RESULTS OF 25 INDEPENDENT RUNS FROM DIFFERENT STAGES OF EVOLUTION($\frac{FEs}{100}$, $\frac{FEs}{10}$,FEs).

1000D		f_5	f_6	f_7
5.0e+04	1^{st}	9.9964e+02	1.2131e+01	-1.1674e+04
	7^{th}	2.9989e+03	1.4462e+01	-1.1328e+04
	13^{th}	3.1484e+03	1.6667e+01	-1.0720e+04
	19^{th}	4.1706e+03	1.6810e+01	-9.7556e+03
	25^{th}	4.5463e+03	1.7210e+01	-9.6350e+03
	M	3.3631e+03	1.5605e+01	-1.0589e+04
	Std	9.4624e+02	1.7201e+00	7.7443e+02
5.0e+05	1^{st}	2.1304e-01	1.5612e-01	-1.3881e+04
	7^{th}	1.1769e+00	3.9539e-01	-1.3812e+04
	13^{th}	1.6448e+00	4.1639e-01	-1.3445e+04
	19^{th}	2.1568e+00	7.7794e-01	-1.2909e+04
	25^{th}	2.2286e+00	9.7411e-01	-1.2785e+04
	M	1.5547e+00	5.2513e-01	-1.3362e+04
	Std	6.0200e-01	2.8627e-01	4.1907e+02
5.0e+06	1^{st}	1.3323e-15	2.0606e-13	-1.4780e+04
	7^{th}	1.4433e-15	2.4158e-13	-1.4771e+04
	13^{th}	1.4433e-15	2.5224e-13	-1.4758e+04
	19^{th}	1.5543e-15	2.7711e-13	-1.4743e+04
	25^{th}	1.7226e-02	2.8777e-13	-1.4731e+04
	M	9.8489e-04	2.5295e-13	-1.4757e+04
	Std	3.6923e-03	2.3677e-14	1.4880e+01

functions[16], but lack of space does not allow to include all of the experimental results.

ACKNOWLEDGMENT

This work was partially supported by an EPSRC grant (No. EP/G002339/1) on "Cooperatively Coevolving Particle Swarms for Large Scale Optimisation".

REFERENCES

[1] S. Luke, "Genetic programming produced competitive soccer softbot teams for robocup97," in *University of Wisconsin*. Morgan Kaufmann, 1998, pp. 214–222.

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

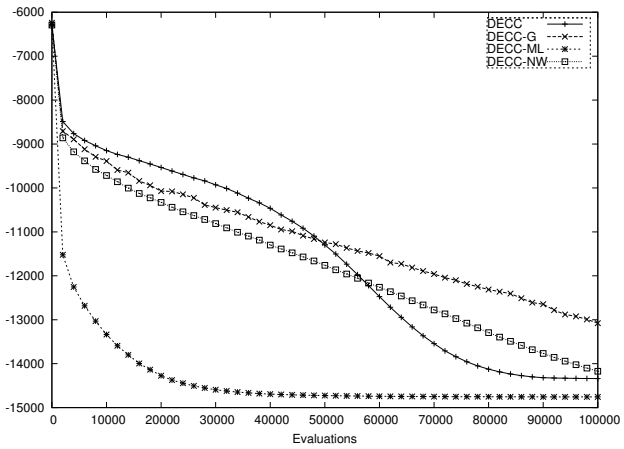


Fig. 5. Convergence plots for f_7 with 1000 dimensions

[3] Z. Yang, K. Tang, and X. Yao, "Multilevel cooperative coevolution for large scale optimization," in *Proc. of IEEE World Congress on Computational Intelligence*, June 2008, pp. 1663–1670.

[4] ——, "Large scale evolutionary optimization using cooperative coevolution," *Information Sciences*, vol. 178, pp. 2986–2999, August 2008.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] J. Vesterstrom and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical," in *Proc. of the 2004 Congress on Evolutionary Computation*, vol. 2, 2004, pp. 1980–1987.

[10] R. Storn and K. Price, "Differential evolution . a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization 11 (4)*, pp. 341–359, 1995.

[11] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[12] R. Gamperle, S. Muller, and P. Koumoutsakos, "A parameter study for differential evolution," in *Proc. WSEAS International Conference on Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, 2002, pp. 293–298.

[13] 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.

[14] 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.

[15] K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, , and Z. Yang, "Benchmark functions for the cec'2008 special session and competition on large scale global optimization," Nature Inspired Computation and Applications Laboratory, USTC, China, Tech. Rep., 2007, http://nical.ustc.edu.cn/cec08ss.php.

[16] 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," Nature Inspired Computation and Applications Laboratory, USTC, China, Tech. Rep., 2009, http://nical.ustc.edu.cn/cec10ss.php.