Cooperative Co-evolution using LSHADE with Restarts For The CEC15 Benchmarks

Mohammed El-Abd
Electrical and Computer Engineering Department
American University of Kuwait
Kuwait
Email: melabd@auk.edu.kw

***Abstract*—In this paper, we test the performance of an LSHADE Cooperative Co-evolutionary (CC) algorithm using the CEC15 benchmarks. First, we apply the recently proposed Global Differential Grouping (GDG) to learn the underlying interdependencies of the problem variables. GDG divides both separable and non-separable variables among multiple sets. Second, the method adopts the LSHADE algorithm within the CC framework to simultaneously optimize the identified groups. Results are reported for all required problem sizes.**

I. Introduction

The concept of Cooperative Co-evolution (CC) was first proposed by [1] using GAs to optimize continuous functions. CC adopts the divide-and-conquer approach by dividing the problem into smaller sub-problems that are optimized independently. This was done Instead of optimizing all problem variables at once. All sub-problems were optimized using the same optimizer, referred to as *sub-optimizer*. However, each sub-problem was optimized using a different instance of the sub-optimizer. The global solution to the problem, referred to as the *context vector*, was constructed using the best individual of each instance. Many evolutionary algorithms have been used as the sub-optimizer including for instance PSO [2], DE [3], BFOA [4], and ABC [5].

At first, the problem decomposition was carried out by considering problem variables in sequence. In addition, this problem decomposition was static in the sense that once some problem variables are grouped together, they will remain in the same group until the search terminates. This is known as *static grouping*. *Random grouping* was introduced by [6] to improve the decomposition process. The method works be dividing the problem into k sub-components, which are re-shuffled in every cycle. The authors in [7] further improved the performance of random grouping by suggesting more frequent grouping. Moreover, the concept of *delta grouping* was introduced in [8]. Although improving performance over random grouping, it did not work well when having multiple non-separable groups in the problem.

Up to this point, all grouping strategies were random at best and did not consider the actual interactions among the problem variables. In order to learn problem variables' interdependencies, *variable interaction learning* was proposed in [9]. In their approach, better decomposition was achieved by iteratively discovering the relations between the variables and grouping related ones together. *Differential grouping* is another method introduced in [10] that was further improved into global differential grouping in [11]. By using *differential grouping*, the problem is divided into a group of separable variables and multiple groups of non-separable variables.

In this work, we test the performance of CC with an introduced restarting mechanism on the CEC15 [12] benchmarks. The problem decomposition is done by GDG and the underlying sub-optimizer used is LSHADE [13]. The paper is divided as follows: Section 2 gives a background about the CC framework. Section 3 briefly explains global differential grouping. The adopted algorithm is detailed in Section 4, results are presented in Section 5. Finally, the paper is concluded in Section 6.

II. Cooperative Co-evolution

In the CC framework, the studied problem divided into smaller sub-problems each of which is optimized by a different instance of the same sub-optimizer. This is achieved by dividing the problem variables over multiple sub-components, hence, each sub-optimizer will be tackling a sub-set of the problem variables.

The *context vector* is then constructed by concatenating the best individual of each sub-optimizer. As an illustration, assume we have k sub-components and that sub-optimizer j is currently active, then individuals of sub-optimizer $_j$ are evaluated by inserting each individual in its position in the *context vector* while filling all the other positions using the best individuals of their respective sub-optimizers. In CC, each sub-optimizer is only run for a single iteration. The term *evolutionary cycle* refers to a complete cycle in which all sub-optimizers are invoked for a single iteration each. The basic CC framework is shown in Algorithm 1.

III. Global Differential Grouping

GDG [11] is an extension of the previous *Differential Grouping* method proposed by the same authors in [10]. In *Differential Grouping*, in order to identify if two variables i and j are interacting, two solution vectors, p^1 and p^2, are constructed. Initially, p^1 and p^2 are formed using the lower bound for all problem variables. Next, problem variable i in solution p^1 is set to its upper bound and $\triangle_1 = f(p^1) - f(p^2)$ is calculated. This is followed by problem variable j being set

978-1-5090-0623-6/16/$31.00 ©2016 IEEE

Algorithm 1 The CC algorithm

Require: $Max\_Function\_Evaluations$
1: Initialize the population
2: $FEVs \leftarrow$ Evaluate the population
3: $Context\_vector = best\_individual$
4: **while** $FEVs \leq Max\_Function\_Evaluations$ **do**
5: **for** each group j **do**
6: Run sub_optimizer$_j$
7: Update $context\_vector$
8: **end for**
9: **end while**
10: **return** $Context\_vector$

to the center of the search space in both p^1 and p^2 and the same equation $\triangle_2 = f(p^1) - f(p^2)$ is calculated. Finally, the variables i and j are deemed dependent if $|\triangle_1 - \triangle_2| > \epsilon$, where ϵ is a very small number.

Two main drawbacks identified are that the comparisons are not carried between all variables so some interactions might be overlooked. In addition, the grouping was sensitive to the values of ϵ. To overcome these drawbacks, the method applied above for detecting the interaction between two variables is actually applied between every pair of all problem variables and the results of all the $|\triangle_1 - \triangle_2|$ values are maintained in a matrix Λ. Another matric Θ representing the interactions between the values is then generated by having Θ_{ij} equal 1 if $\Lambda_{ij} > \epsilon$ and zero otherwise. In addition, the value of ϵ is automatically set based on the objective space magnitude. Initially, K points are randomly selected in the search space and their objective values are calculated. Then, ϵ is set to $\alpha \times min\{|f(x_1)|, |f(x_2)|, ..., |f(x_K)|\}$, where α equals 10^{-10}.

To further improve the performance for separable functions, *GDG* divides the separable groups into smaller groups of size 20 each following the recommendations in [14]. The authors proposed a rule of thumb for selecting the subcomponent. The size should be small enough in order to be within the capacity of the sub-optimizer, but not smaller.

We have implemented two modifications to GDG. First, as we are not dealing with large-scale optimization problems, separable groups are divided into smaller sets of 10 problem variables instead of 20. Second, if GDG identifies a separable group having a number of problem variables $\leq 10\%$ of the entire problem size, these variables are added to the smallest group identified by GDG. This second modification is implemented to avoid wasting a large number of function evaluations on a very small number of problem variables that are essentially separable.

IV. Cooperative Co-evolution with Restarts

As stated earlier, the algorithm starts by running *GDG* in order to extract the separable group(s) and the non-separable group(s). Afterwards, LSHADE is run within the CC framework to tackle the problem. Two different cases exist depending on the outcome of the first stage.

A. Multiple groups

If multiple groups are identified, the CC framework is run using a different LSHADE instance for each group. As LSHADE sets the initial population size depending on the problem size, larger groups would be allocated more computational budget. Problem variables grouped in a single set i might have different degree and type of dependencies compared to problem variables grouped in another set j. Hence, the convergence behavior for LSHADE instance i could be very much different than the convergence behavior for LSHADE instance j. In our CC framework, the contribution of each LSHADE instance to the improvement of the context vector is recorded. If a certain instance is unable to improve the context vector after this instance consecutively performing a fixed number of function evaluations, f_{stag},the convergence status in the f-space for this instance is examined by evaluating the following equation:

$$fconverged = \begin{cases} 1 & \texttt{if} \quad (f_{max} - f_{min}) \leq \varepsilon, \\ 0 & otherwise. \end{cases} \tag{1}$$

If the instance has converged according to the two conditions above, this LSHADE instance is restarted by clearing the archives and randomly re-initializing its sub-population. However, we make sure that the *context vector* at this point is injected into the new sub-population before restarting. The first stagnation condition mentioned above is similar to the one provided in [15]. However, in [15], the algorithm is required to execute a minimum number of function evaluations before being tested for convergence, this is omitted in our algorithm. In addition, testing for convergence in the f-space is done using a different equation.

B. One group

If the problem is completely non-separable, we adopt the following mechanism:

- One instance of LSHADE is run on the entire problem until $x\%$ of the allocated computational budget is consumed,
- The problem variables are then randomly divided into two groups, of equal size, and the CC framework is run using two LSHADE instances that are randomly initialed making sure that the *context vector* reached thus far is injected into them. This is run until $y\%$ of the allocated computational budget is consumed,
- The problem variables are further divided into K groups, of equal size, and the CC framework is run using D/K LSHADE instances that are randomly initialed making sure that the *context vector* reached thus far is injected into them. This is run until the remaining of the allocated computational budget is consumed.

We still use the convergence tests previously identified but the action taken is dependent on the current number of groups. If we have a single group, the LSHADE instance is restarted as previously explained. However, if we have multiple groups, these groups are re-shuffled. The re-shuffling

decision was based on what was previously shown in [7] that having more frequent grouping increases the probability of having interacting variables together in a single group.

V. Experimental Results

A. Parameters Settings

To test the performance of our proposed framework, LSHADE is used as the underlying sub-optimizer as it was a joint winner of the CEC14 competition. Table I presents the parameter setting for both LSHADE and the CC algorithm. Setting ε to 10^{-8} was based on the fact that this value represents a zero for the CEC15 benchmarks. Hence, when the difference between f_{max} and f_{min} is less than this value, we consider that all individuals in the sub-population have the same fitness. In addition, setting the f_{stag} to the value identified in the table gives a different limit for each LSHADE instance based on the number of problem variables in the group.

Experiments are carried on the CEC15 benchmarks [12]. Results reported are based on 51 independent runs. The maximum number of fitness evaluations is set to $10000 \times Dim$ for $Dim = 10, 30, 50,$ and 100.

The code for LSHADE was downloaded from [16] and incorporated into our implementation of the CC framework. The code for GDG was downloaded from [17]. We further modified the code to add the restarting mechanism explained earlier.

TABLE I
Parameter Settings

Algorithm	Parameter	Value
LSHADE	$r^{N^{init}}$	18
	p	0.11
	r^{arc}	1.4
	H	5
	N_{min}	4
CC	f_{stag}	$1000 \times Group\_Dim$
	ε	10^{-8}
	x	50
	y	30
	K	5

B. Results and Discussion

1) Global Differential Grouping Stage: Applying the GDG algorithm provided the results in Tables II, III, IV, and V. The tables show the number of non-separable groups (containing subcomponents of interacting problem variables), the number of separable variables (grouped in a single set), and the consumed number of function evaluations by GDG. Functions missing from the tables ($f9 \sim f15$) were all successfully identified as non-separable functions having a single group.

As an additional measure of the GDG performance, problem variables in hybrid functions $f6 \sim f8$ are always accurately grouped into $3 \sim 5$ sets with the only exception of $f8$ at $Dim = 50$.

TABLE II
Results of global differential grouping - D = 10

Benchmark Function	# of non-separable groups	# of separable variables	FEVs consumed
f1	3	0	76
f2	2	3	76
f3	1	0	76
f4	3	0	76
f5	3	0	76
f6	3	0	76
f7	4	0	76
f8	4	1	76

TABLE III
Results of global differential grouping - D = 30

Benchmark Function	# of non-separable groups	# of separable variables	FEVs consumed
f1	6	0	506
f2	4	6	506
f3	1	0	506
f4	6	0	506
f5	6	0	506
f6	3	0	506
f7	4	0	506
f8	4	3	506

TABLE IV
Results of global differential grouping - D = 50

Benchmark Function	# of non-separable groups	# of separable variables	FEVs consumed
f1	8	0	1336
f2	4	22	1336
f3	1	0	1336
f4	8	0	1336
f5	8	0	1336
f6	3	0	1336
f7	4	0	1336
f8	5	3	1336

TABLE V
Results of global differential grouping - D = 100

Benchmark Function	# of non-separable groups	# of separable variables	FEVs consumed
f1	10	0	5161
f2	5	50	5161
f3	1	0	5161
f4	10	0	5161
f5	10	0	5161
f6	3	0	5161
f7	4	0	5161
f8	5	0	5161

2) CCLSHADE Results: The results of CCLSHADE are presented in Tables VI, VII, VIII, and IX. The tables show the best, worst, median, mean, and standard deviation of the results based on 51 runs.

We compared the results of our CCLSHADE individually to the best performers of the CEC15 competition, which are SPS-L-SHADE-EIG [18], DEsPA [19], MVMO [20], and LSHADE-ND [21]. Comparisons were based on the Wilcoxon test at a confidence interval of 5%. The comparisons were run for dimensions 10, 30, 50 and 100 using the KEEL software [22] following the suggestions in [23]. Results presented in

TABLE VI
RESULTS FOR 10D

Benchmark Function	Best	Worst	Median	Mean	Std
f1	0.00e+000	1.42e-014	0.00e+000	6.97e-015	7.17e-015
f2	0.00e+000	2.84e-014	0.00e+000	1.23e-014	1.42e-014
f3	5.68e-014	2.00e+001	2.00e+001	1.84e+001	5.43e+000
f4	0.00e+000	9.95e-001	0.00e+000	1.95e-002	1.39e-001
f5	6.25e-002	6.83e+000	2.50e-001	6.94e-001	1.34e+000
f6	0.00e+000	4.16e-001	6.54e-006	8.43e-002	1.16e-001
f7	0.00e+000	2.91e-002	1.94e-002	1.75e-002	9.45e-003
f8	3.84e-005	1.27e-002	2.15e-003	3.27e-003	3.17e-003
f9	1.00e+002	1.00e+002	1.00e+002	1.00e+002	3.38e-002
f10	2.17e+002	2.17e+002	2.17e+002	2.17e+002	9.94e-003
f11	6.18e-001	3.00e+002	3.13e+000	1.42e+002	1.50e+002
f12	1.00e+002	1.01e+002	1.01e+002	1.01e+002	2.54e-001
f13	3.04e-002	3.05e-002	3.05e-002	3.05e-002	5.32e-005
f14	2.93e+003	6.68e+003	2.93e+003	4.40e+003	1.85e+003
f15	1.00e+002	1.00e+002	1.00e+002	1.00e+002	0.00e+000

TABLE VII
RESULTS FOR 30D

Benchmark Function	Best	Worst	Median	Mean	Std
f1	1.42e-014	4.26e-014	2.84e-014	2.28e-014	7.56e-015
f2	2.84e-014	8.53e-014	5.68e-014	5.07e-014	1.31e-014
f3	2.00e+001	2.05e+001	2.02e+001	2.02e+001	7.95e-002
f4	1.99e+000	9.58e+000	4.98e+000	4.92e+000	2.06e+000
f5	2.89e+001	7.42e+002	2.47e+002	3.00e+002	1.37e+002
f6	2.00e+000	3.22e+002	3.18e+001	8.17e+001	9.03e+001
f7	6.37e-001	2.52e+000	1.51e+000	1.50e+000	4.03e-001
f8	6.17e-001	1.24e+002	1.21e+001	1.58e+001	1.80e+001
f9	1.02e+002	1.03e+002	1.03e+002	1.03e+002	1.32e-001
f10	2.06e+002	9.22e+002	6.12e+002	6.06e+002	1.20e+002
f11	3.01e+002	4.43e+002	4.00e+002	4.01e+002	1.84e+001
f12	1.03e+002	1.04e+002	1.04e+002	1.04e+002	2.60e-001
f13	2.56e-002	2.72e-002	2.59e-002	2.60e-002	3.43e-004
f14	3.11e+004	3.35e+004	3.31e+004	3.28e+004	7.27e+002
f15	1.00e+002	1.00e+002	1.00e+002	1.00e+002	0.00e+000

TABLE VIII
RESULTS FOR 50D

Benchmark Function	Best	Worst	Median	Mean	Std
f1	7.11e-014	1.38e-010	5.10e-012	1.33e-011	2.41e-011
f2	2.84e-014	1.14e-013	5.68e-014	7.13e-014	1.83e-014
f3	2.00e+001	2.07e+001	2.04e+001	2.04e+001	9.78e-002
f4	2.78e+000	1.10e+001	6.54e+000	6.43e+000	1.82e+000
f5	2.18e+001	9.53e+002	4.94e+002	5.31e+002	1.99e+002
f6	1.48e+001	1.15e+003	4.85e+002	5.01e+002	2.07e+002
f7	3.67e+001	3.90e+001	3.82e+001	3.81e+001	5.47e-001
f8	4.91e+000	6.48e+002	2.63e+002	2.39e+002	1.38e+002
f9	1.04e+002	1.05e+002	1.04e+002	1.04e+002	1.80e-001
f10	8.15e+002	1.81e+003	1.08e+003	1.14e+003	2.38e+002
f11	4.00e+002	4.63e+002	4.09e+002	4.17e+002	1.85e+001
f12	1.04e+002	2.00e+002	1.05e+002	1.29e+002	4.18e+001
f13	7.27e-002	9.04e-002	8.01e-002	7.98e-002	3.70e-003
f14	5.90e+004	7.40e+004	6.89e+004	6.52e+004	5.78e+003
f15	1.00e+002	1.00e+002	1.00e+002	1.00e+002	0.00e+000

Table X show the improved performance of CCLSHADE over three of these algorithms. This improvement is achieved despite not tuning our parameters for each benchmark function separately as allowed by the competition.

3) Algorithm Complexity: Experiments are run under MATLAB (R2013b) on a Quad-Core machine running @2.40 GHz with 4 GB of RAM. The computational complexity of the algorithm is provided in Table XI. Since this complexity is calculated based on function $f1$, which is a function containing multiple non-separable groups, it only measures the complexity of one of the two cases of our CC framework that is having different number of groups across different problem sizes.

TABLE IX
RESULTS FOR 100D

Benchmark Function	Best	Worst	Median	Mean	Std
f1	5.26e-013	1.07e-009	2.41e-011	7.44e-011	1.87e-010
f2	8.53e-014	1.99e-013	1.42e-013	1.41e-013	2.41e-014
f3	2.02e+001	2.09e+001	2.07e+001	2.07e+001	9.48e-002
f4	1.10e+001	2.39e+001	1.59e+001	1.62e+001	3.33e+000
f5	8.83e+002	2.96e+003	2.12e+003	2.08e+003	4.83e+002
f6	1.35e+003	4.26e+003	2.16e+003	2.22e+003	4.47e+002
f7	8.95e+001	9.73e+001	9.13e+001	9.19e+001	2.05e+000
f8	1.01e+003	5.33e+003	1.68e+003	2.04e+003	1.08e+003
f9	1.06e+002	1.07e+002	1.07e+002	1.07e+002	1.77e-001
f10	3.20e+003	4.93e+003	3.86e+003	3.96e+003	4.77e+002
f11	4.31e+002	7.37e+002	5.91e+002	5.87e+002	6.79e+001
f12	1.12e+002	2.00e+002	1.12e+002	1.14e+002	1.23e+001
f13	5.97e-002	6.63e-002	6.45e-002	6.43e-002	1.47e-003
f14	1.09e+005	1.09e+005	1.09e+005	1.09e+005	2.07e+001
f15	1.00e+002	1.00e+002	1.00e+002	1.00e+002	0.00e+000

TABLE X
COMPARISON AGAINST THE CEC15 COMPETITION BEST PERFORMERS

Vs.	R^+	R^-	p-value
LSHADE-ND	1107.5	662.5	0.092029
MVMO	1089.0	681.0	0.120931
DEsPA	1165.5	664.5	0.064637
SPS-L-SHADE-EIG	571.0	1199.0	1

TABLE XI
COMPUTATIONAL COMPLEXITY

Problem Size	$T0$	$T1$	$\hat{T}2$	$\frac{\hat{T}2-T1}{T0}$
10	0.22	2.42	3.17	3.41
30		3.06	3.85	3.59
50		4.17	18.81	66.55
100		8.23	21.84	61.86

VI. CONCLUSIONS

In this paper, we tested the performance CC using LSHADE and a restarting mechanism on the CEC15 benchmarks. The algorithm starts by using GDG to decompose the problem and then apply CC to the identified groups. In future work, it is intended to study the use of different algorithms as the underlying sub-optimizers and to investigate different restarting mechanisms.

REFERENCES

[1] M. A. Potter and K. A. de Jong, "A cooperative coevolutionary approach to function optimization," in *Proc. 3rd Parallel problem Solving from Nature*, 1994, pp. 249–257.

[2] F. van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 225–239, 2004.

[3] Y. Shi, H. Teng, and Z. Li, "Cooperative co-evolutionary differential evolution for function optimization," in *Proc. of International Conference on Natural Computation*, 2005, pp. 1080–1088.
[4] H. Chen, Y. Zhu, and K. Hu, "Cooperative bacterial foraging optimization," *Discrete Dynamics in Nature and Society*, vol. 2009, pp. 1–17, 2009.
[5] M. El-Abd, "A cooperative approach to the artificial bee colony algorithm," in *Proc. of IEEE Congress on Evolutionary Computation*, 2010, pp. 1–5.
[6] Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Information Sciences*, no. 178, pp. 2985–2999, 2008.
[7] 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 IEEE Congress on Evolutionary Computation*, 2010, pp. 1–8.
[8] M. N. Omidvar, X. Li, and X. Yao, "Cooperative co-evolution with delta grouping for large scale non-separable function optimization," in *Proc. of IEEE Congress on Evolutionary Computation*, 2010, pp. 1762–1769.
[9] W. Chen, T. Weise, Z. Yang, and K. Tang, "Large-scale global optimization using cooperative coevolution with variable interaction learning," in *Proc. of International Conference on Parallel Problem Solving from Nature*, ser. Lecture Notes in Computer Science, vol. 6239. Berlin:Springer, 2011, pp. 300–309.
[10] M. N. Omidvar, X. Li, Y. Mei, and X. Yao, "Cooperative co-evolution with differential grouping for large scale optimization," *IEEE Transactions on Evolutionary Computation*, pp. 378–393, 2014.
[11] Y. Mei, M. N. Omidvar, X. Li, and X. Yao, "A copmpetitive divide-and-conquer algorithm for unconstrained large scale black-box optimization," *ACM Transactions on Mathematical Software*, 2015.
[12] B. Y. Qu, J. J. Liang, Z. Y. Wang, Q. Chen, and P. N. Suganthan, "Novel benchmark functions for continuous multimodal optimization with comparative results," *Swarm and Evolutionary Computation*, vol. 26, pp. 23–34, 2016.
[13] R. Tanabe and A. Fukunaga, "Improving the search performance of shade using linear population size reduction," in *Proc. of IEEE Congress on Evolutionary Computation*, 2014, pp. 1658–1665.
[14] M. N. Omidvar, Y. Mei, and X. Li, "Effective decomposition of large-scale separable continuous functions for cooperative co-evolutionary algorithms," in *Proc. of IEEE Congress on Evolutionary Computation*, 2014, pp. 1305–1312.
[15] R. Tanabe and A. Fukunaga, "Tuning differential evolution for cheap, medium, and expensive computational budgets," in *Proc. of IEEE Congress on Evolutionary Computation*, 2015, pp. 2018–2025.
[16] R. Tanabe, "lshade code," https://sites.google.com/site/tanaberyoji/software, 2014.
[17] Y. Mei, "gdg code," http://www.mathworks.com/matlabcentral/fileexchange/45783-the-cc-gdg-cmaes-algorithm, 2015.
[18] S.-M. Guo, J.-H. Tsai, C.-C. Yang, and P.-H. Hsu, "A self-optimization approach for l-shade incorporated with eigenvector-based crossover and successful-parent-selecting framework on cec 2015 benchmark set," in *Proc. of IEEE Congress on Evolutionary Computation*, 2015, pp. 1003–1010.
[19] N. Awad, M. Z. Ali, and R. G. Reynolds, "A differential evolution algorithm with successbased parameter adaptation for cec2015 learning based optimization," in *Proc. of IEEE Congress on Evolutionary Computation*, 2015, pp. 1098–1105.
[20] J. L. Rueda and I. Erlich, "Testing mvmo on learning-based real-parameter single objective benchmark optimization problems," in *Proc. of IEEE Congress on Evolutionary Computation*, 2015, pp. 1025–1032.
[21] K. M. Sallam, R. A. Sarkar, D. L. Essam, and S. M. Elsayed, "Neurodynamic differential evolution algorithm and solving cec2015 competition problems," in *Proc. of IEEE Congress on Evolutionary Computation*, 2015, pp. 1033–1040.
[22] "Keel software," http://sci2s.ugr.es/keel/download.php, 2012.
[23] J. Derrac, S. Garcia, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, 2011.