Comparative Study of Decomposition and Merging Evolutionary Algorithms for Large-Scale Optimization Problems Supplementary Information

Zachary McGovarin, Alanna McNulty, Beatrice Ombuki-Berman

1 Supplementary Notes

1.1 Co-operative Coevolutionary Framework

One implementation of the CC algorithm begins by splitting the problem in half, creating two sub-problems, each optimized using their own EA [2, 11]. Therefore, one EA is responsible for optimizing first half of the decision variables, while the other EA optimizes the second half. Another popular implementation choice is to create a separate EA for each decision variable in the problem [7, 11]. Other variations of CC-based algorithms change the number of sub-problems and EAs created using various methodologies [8, 9]. Regardless, the main idea of statically splitting the problem into sub-problems before execution remains. In our work, we implement the approach where the problem is split in half.

1.2 Decomposition and Merging Condition

The frequency at which the decomposition or merging occurs is crucial for ensuring the search space is both properly explored, and properly exploited. If decomposition or merging does not occur at an appropriate rate, performance will be negatively impacted. The rate of decomposition and merging is taken from [3, 4], and is as follows:

$$n_f = \frac{n_T}{1 + (\frac{log(n_x)}{log(n_r)})} \tag{1}$$

where n_f is the number of fitness evaluations between each decomposition or merging step, n_T is the maximum number of permitted fitness evaluations, n_x is the number of dimensions in the optimization problem, and n_r is the number of EAs to be merged or created, which is kept at a constant value of two as was done in [3, 4].

1.3 Performance Metrics & Statistical Tests

The final fitness values obtained by each of the algorithms were compared using the Kruskal-Wallis Test to check for stochastic dominance amongst the samples [10]. If stochastic dominance was found to be present within the results, the Conover-Iman Test was applied to perform a pairwise analysis in order to determine which algorithms were stochastically dominant [14, 15]. A significance level of 0.05 was used in all statistical tests.

The algorithms were ranked based on the number of functions they won, tied, and lost. An algorithm was determined to "win" a function if it had the best performance (i.e. lowest fitness evaluation) and its performance results were determined to be statistically significant. If an algorithm had the lowest fitness evaluation for a given function, but the independence of the algorithm from one or more other algorithms could not be statistically verified, the algorithm was awarded a tie. The other algorithms lacking independence from the best performing algorithm were also awarded a tie. An algorithm was awarded one point for a win, zero points for a tie, and lost one point for a loss. Using this scoring system, the algorithms were then ranked for each function type, with rank 1 being the best performing algorithm.

As an additional gauge of each algorithm's general performance, we compare the normalized average final fitness evaluations obtained by each algorithm for each function. An algorithm's average fitness evaluation on a given function is min-max normalized using the following equation

$$x'_{ijk} = \frac{x_{ijk} - max(\mathbf{X}_{jk})}{max(\mathbf{X}_{jk}) - min(\mathbf{X}_{jk})} \tag{2}$$

where x is algorithm i's average final fitness value at j dimensions for benchmark function k, $\mathbf{X}$ is a vector containing the average final fitness evaluations of the algorithms, and x' is the resulting normalized average final fitness value. We then sum the normalized average final fitness evaluations of each algorithm at each dimension tested for comparison.

1.4 Benchmark Functions

The benchmark functions are from the CEC'2010 Special Session and Competition on Large-Scale Global Optimization [5] and CEC'2013 Special Session and Competition on Large-Scale Global Optimization [13] The CEC'2010 test suite contains a total of 20 functions which are scalable to any number of decision variables, and consists of three separable, five single-group partially-separable, five $n_x/2m-$ group separable, five $n_x/m-$separable, and two non-separable functions. A function is called separable if there are no dependencies amongst its decision variables. Partially separable indicates that there is a relationship between at least two (but not all) of the decision variables. A nonseparable function has dependencies amongst all of its decision variables.

In contrast, the CEC'2013 test suite consists of 15 benchmark functions statically defined at $1000-$dimensions. These functions offer new challenges, including nonuniform sub-component sizes, imbalanced sub-component contributions, overlapping sub-components, and more, further described in [13]. There are three separable functions, four partially-additively separable functions with a separable sub-component, four partially-additively separable functions with no separable sub-components, three overlapping functions, and one non-separable function.

1.5 Parameter Settings

The parameters used for the three variants of each EA were equivalent in all experiments. The PSO algorithms used a swarm size of 20, social and cognitive coefficient values of 1.49618, and an inertia weight of 0.729844 as in [4, 11]. For ABC, a swarm size of 20 was again used, along with a maximum of one scout bee, the same as in [23]. Determining optimal DE parameters for a specific problem or set of problems requires an exhaustive trial-and-error search, there are no widely agreed upon optimal parameters for general solutions [12, 16, 17]. Several popular parameter values were compared in a few rudimentary benchmark tests and the best performing parameters were used for experimentation. These parameters are a population size of 50, scale factor of 0.5, and a crossover rate of 0.7. Each algorithm will be given a maximum of $3000n_x$ fitness evaluations for each run, taken from [6].

1.6 Differential Evolution

To perform mutation, the "DE/rand/1" method is used, as presented in [1]:

$$V_i = X_{r_1} + F * (X_{r_2} - X_{r_3}) \tag{3}$$

where V_i is i^{th} the trial vector being generated, F is the scale factor, X_i is the i^{th} vector in the population, and r_1, r_2, r_3 are distinct random indices generated within the population size. Additionally, i is distinct from r_1, r_2, r_3. To perform crossover, binomial crossover is used, as seen in [1]. Binomial crossover generates an offspring vector $U_i = u_1, u_2, ..., u_n$ using the following function:

$$u_i = \begin{cases} t_i, & \text{if } random(0,1) < CR. \\ s_i, & \text{otherwise.} \end{cases} \tag{4}$$

In which CR is the crossover rate, t_i is value of the trial vector in the i^{th} dimension, and s_i is the value of the target vector in the i^{th} dimension.

1.7 Particle Swarm Optimization

The algorithm begins by randomly placing the particles throughout the search space. The positions of the particles are then updated through an iterative approach, with their velocities being calculated by the following equation:

$$\mathbf{v_i}(t+1) = w\mathbf{v_i}(t) + c_1\mathbf{r_{1i}}(\mathbf{y_i}(t) - \mathbf{x_i}(t)) + c_2\mathbf{r_{2i}}(\hat{\mathbf{y}}(t) - \mathbf{x_i}(t)) \tag{5}$$

where $\mathbf{v_i}$ is the velocity of particle i at iteration t, c_1 and c_2 are the statically defined cognitive and social coefficients, $\mathbf{r_1}$ and $\mathbf{r_2}$ are vectors with values uniformly and randomly sampled from (0, 1), $\mathbf{x_i}$ is the particle's current position, $\mathbf{y_i}$ is the particle's best position, and $\hat{\mathbf{y}}$ is the swarm's global best position. Once calculated, a given particle's position can be updated using its velocity with the following equation:

$$\mathbf{x_i}(t+1) = \mathbf{x_i}(t) + \mathbf{v_i}(t+1) \tag{6}$$

1.8 Artificial Bee Colony

The algorithm starts by placing the employed bees onto randomly generated “food sources” throughout the search space (i.e. the position of each employed bee represents a food source). For each iteration, the employed bees will explore around their food source, using the following equation to generate a candidate food source:

$$v_{ij} = x_{ij} + r(x_{ij} - x_{kj}) \tag{7}$$

where i, k are distinct randomly selected food source indices, k is a randomly selected dimension, r is a random value sampled from [-1,1], x_{ij} is the i^{th} food source in the j^{th} dimension, and v_{ij} is the candidate food source. If current food source has a worse fitness value than the candidate, it is replaced by the candidate. Each onlooker bee then selects a random food source using roulette wheel selection and again explores around it using the aforementioned employee bee method. Lastly, any employed bee that does not find an improved food source after a specified number of iterations is automatically converted to a scout bee. A scout bee randomly generates a new food source within the search space and then returns to exploring its surroundings as an employed bee.

1.9 Decomposition & Merging Approach

The main idea of the algorithm is to improve upon the standard CC framework by making the divide-and-conquer approach more dynamic. In the standard CC-based implementations, the problem is decomposed into two sub-problems at the beginning of the algorithm’s execution and then left unchanged. However, utilizing the decomposition and merging techniques seen in the DCPSO and MCPSO algorithms, we will instead dynamically decompose and merge the problem-space throughout the execution of the algorithm.

For the decomposition approaches, a single EA with n_x decision variables is initialized. Throughout the execution of the EA algorithm, the EA is decomposed at a fixed interval. During a decomposition, every existing EA instance is split into two separate EAs. This process will continue until there are n_x one-dimensional EAs. Note that in the case where a decomposition cannot evenly occur (i.e., when a DE has an odd number of dimensions) the resulting EAs will not have the same number of dimensions.

For example, consider an optimization problem with 100 dimensions. In DCDE, a single DE is initialized with 100 dimensions. Once the decomposition condition is met, the order of the decision variables is shuffled and then the single DE is split into two 50-dimensional DEs, each with their own independent populations and decision variables. The random shuffling of decision variables at each decomposition step is crucial to ensure that a variety of decision variable groupings is explored. At the next decomposition step, the two 50-dimensional DEs are decomposed into four 25-dimensional DEs, followed by eight DEs of 12 and 13 dimensions, and so on and so forth until there are n_x one-dimensional DEs.

The merging approach presented reverses the procedure of the first and includes the MCPSO, MCDE, and MCABC algorithms. While the decomposition approach begins with a single n_x-dimensional DE and decomposes it until there are n_x one-dimensional EAs, the merging approach begins with n_x one-dimensional EAs and merges them until there is only a single n_x-dimensional EA. In the case that there is an odd number of EAs, there will be a single EA that does not get merged and thus remains unchanged until the next merge event occurs.

2 Supplementary Tables

In Supplementary Tables 1 and 2, the algorithms are ranked based upon the minimum and average final fitness values that they obtained over the course of 30 runs. The results are provided in the “Win/Tie/Loss - Rank” format. The rankings are differentiated by function type (i.e. by row). The best-ranked algorithm is indicated in bold for easy identification. Additionally, the ranks are colour-coded, with the top-ranked algorithm(s) being highlighted in green and the worst algorithm(s) highlighted in red.

Table 1: Algorithms Best Results - Ranked

Function Type	Algorithm								
	CCDE (W/T/L - R)	DCDE (W/T/L - R)	MCDE (W/T/L - R)	CPSO (W/T/L - R)	DCPSO (W/T/L - R)	MCPSO (W/T/L - R)	CABC (W/T/L - R)	DCABC (W/T/L - R)	MCABC (W/T/L - R)
100 Dimensions									
Separable	0/0/3— 3	0/0/3— 3	0/0/3— 3	0/0/3— 3	0/1/2— 2	0/1/2— 2	**1/2/0— 1**	0/0/3— 3	0/0/3— 3
Single-group	0/0/5— 3	1/0/4— 2	0/0/5— 3	**3/0/2— 1**	0/0/5— 3	0/0/5— 3	1/0/4— 2	0/0/5— 3	0/0/5— 3
n_x/2m-group	0/1/4— 2	**2/1/2— 1**	0/1/4— 2	**2/1/2— 1**	0/1/4— 2	0/0/5— 3	0/1/4— 2	0/1/4— 2	0/0/5— 3
n_x/m-group	0/1/4— 4	**2/0/3— 1**	0/0/5— 5	1/1/3— 2	0/0/5— 5	0/2/3— 3	0/0/5— 5	0/0/5— 5	0/0/5— 5
Non-separable	0/0/2— 2	0/0/2— 2	0/0/2— 2	0/0/2— 2	0/0/2— 2	**2/0/0— 1**	0/0/2— 2	0/0/2— 2	0/0/2— 2
500 Dimensions									
Separable	0/0/3— 3	0/0/3— 3	0/0/3— 3	0/0/3— 3	0/0/3— 3	1/0/2— 2	**2/0/1— 1**	0/0/3— 3	0/0/3— 3
Single-group	0/1/4— 4	**2/1/2— 1**	0/1/4— 4	0/2/3— 3	0/0/5— 5	1/2/2— 2	0/0/5— 5	0/0/5— 5	0/0/5— 5
n_x/2m-group	0/0/5— 3	2/0/3— 2	0/0/5— 3	0/0/5— 3	0/0/5— 3	**3/0/2— 1**	0/0/5— 3	0/0/5— 3	0/0/5— 3
n_x/m-group	0/1/4— 3	1/1/3— 2	0/0/5— 4	0/0/5— 4	0/0/5— 4	**3/0/2— 1**	0/0/5— 4	0/0/5— 4	0/0/5— 4
Non-separable	0/0/2— 2	0/0/2— 2	0/0/2— 2	0/0/2— 2	0/0/2— 2	**2/0/0— 1**	0/0/2— 2	0/0/2— 2	0/0/2— 2
1000 Dimensions (2010)									
Separable	0/0/3— 3	0/0/3— 3	0/0/3— 3	0/0/3— 3	0/0/3— 3	1/0/2— 2	**2/0/1— 1**	0/0/3— 3	0/0/3— 3
Single-group	0/0/5— 3	2/0/3— 2	0/0/5— 3	0/0/5— 3	0/0/5— 3	**3/0/2— 1**	0/0/5— 3	0/0/5— 3	0/0/5— 3
n_x/2m-group	0/0/5— 3	2/0/3— 2	0/0/5— 3	0/0/5— 3	0/0/5— 3	**3/0/2— 1**	0/0/5— 3	0/0/5— 3	0/0/5— 3
n_x/m-group	0/0/5— 4	1/0/4— 2	0/1/4— 3	0/0/5— 4	0/0/5— 4	**3/0/2— 1**	0/0/5— 4	0/0/5— 4	0/1/4— 3
Non-separable	0/0/2— 2	0/0/2— 2	0/0/2— 2	0/0/2— 2	0/0/2— 2	**2/0/0— 1**	0/0/2— 2	0/0/2— 2	0/0/2— 2
1000 Dimensions (2013)									
Separable	0/0/3— 2	0/0/3— 2	0/0/3— 2	0/0/3— 2	0/0/3— 2	**1/0/2— 1**	**1/0/2— 1**	0/0/3— 2	**1/0/2— 1**
Additive	0/1/7— 4	**2/3/3— 1**	1/1/6— 3	0/5/3— 2	0/0/8— 5	0/0/8— 5	0/0/8— 5	0/0/8— 5	0/0/8— 5
Overlapping	0/0/3— 3	0/1/2— 2	0/0/3— 3	**2/0/1— 1**	0/0/3— 3	0/0/3— 3	0/1/2— 2	0/0/3— 3	0/0/3— 3
Non-separable	0/0/1— 2	**1/0/0— 1**	0/0/1— 2	0/0/1— 2	0/0/1— 2	0/0/1— 2	0/0/1— 2	0/0/1— 2	0/0/1— 2
2000 Dimensions									
Separable	0/0/3— 3	0/0/3— 3	0/0/3— 3	0/0/3— 3	0/0/3— 3	1/0/2— 2	**2/0/1— 1**	0/0/3— 3	0/0/3— 3
Single-group	1/0/4— 2	1/0/4— 2	0/0/5— 3	0/0/5— 3	0/0/5— 3	**3/0/2— 1**	0/0/5— 3	0/0/5— 3	0/0/5— 3
n_x/2m-group	1/0/4— 2	**2/0/3— 1**	0/0/5— 3	0/0/5— 3	0/0/5— 3	**2/0/3— 1**	0/0/5— 3	0/0/5— 3	0/0/5— 3
n_x/m-group	0/1/4— 3	1/0/4— 2	0/1/4— 3	0/0/5— 4	0/0/5— 4	**3/0/2— 1**	0/0/5— 4	0/0/5— 4	0/0/5— 4
Non-separable	0/0/2— 2	0/0/2— 2	0/0/2— 2	0/0/2— 2	0/0/2— 2	**1/0/1— 1**	**1/0/1— 1**	0/0/2— 2	0/0/2— 2

Table 2: Algorithms Average Results - Ranked

Function Type	Algorithm								
	CCDE (W/T/L - R)	DCDE (W/T/L - R)	MCDE (W/T/L - R)	CPSO (W/T/L - R)	DCPSO (W/T/L - R)	MCPSO (W/T/L - R)	CABC (W/T/L - R)	DCABC (W/T/L - R)	MCABC (W/T/L - R)
100 Dimensions									
Separable	0/0/3— 3	0/0/3— 3	0/0/3— 3	0/0/3— 3	0/0/3— 3	1/0/2— 2	**2/0/1— 1**	0/0/3— 3	0/0/3— 3
Single-group	0/0/5— 3	2/0/3— 2	0/0/5— 3	**3/0/2— 1**	0/0/5— 3	0/0/5— 3	0/0/5— 3	0/0/5— 3	0/0/5— 3
n_x/2m-group	0/0/5— 3	**2/0/3— 1**	0/0/5— 3	**2/0/3— 1**	0/0/5— 3	1/0/4— 2	0/0/5— 3	0/0/5— 3	0/0/5— 3
n_x/m-group	0/1/4— 4	**2/0/3— 1**	0/0/5— 5	1/1/3— 2	0/0/5— 5	0/2/3— 3	0/0/5— 5	0/0/5— 5	0/0/5— 5
Non-separable	0/0/2— 2	0/0/2— 2	0/0/2— 2	0/0/2— 2	0/0/2— 2	**2/0/0— 1**	0/0/2— 2	0/0/2— 2	0/0/2— 2
500 Dimensions									
Separable	0/0/3— 3	0/0/3— 3	0/0/3— 3	0/0/3— 3	0/0/3— 3	1/0/2— 2	**2/0/1— 1**	0/0/3— 3	0/0/3— 3
Single-group	0/1/4— 3	**2/1/2— 1**	0/1/4— 3	0/1/4— 3	0/0/5— 4	1/2/2— 2	0/0/5— 4	0/0/5— 4	0/0/5— 4
n_x/2m-group	0/0/5— 3	2/0/3— 2	0/0/5— 3	0/0/5— 3	0/0/5— 3	**3/0/2— 1**	0/0/5— 3	0/0/5— 3	0/0/5— 3
n_x/m-group	0/1/4— 3	1/1/3— 2	0/0/5— 4	0/0/5— 4	0/0/5— 4	**3/0/2— 1**	0/0/5— 4	0/0/5— 4	0/0/5— 4
Non-separable	0/0/2— 2	0/0/2— 2	0/0/2— 2	0/0/2— 2	0/0/2— 2	**2/0/0— 1**	0/0/2— 2	0/0/2— 2	0/0/2— 2
1000 Dimensions (2010)									
Separable	0/0/3— 3	0/0/3— 3	0/0/3— 3	0/0/3— 3	0/0/3— 3	1/0/2— 2	**2/0/1— 1**	0/0/3— 3	0/0/3— 3
Single-group	0/0/5— 3	2/0/3— 2	0/0/5— 3	0/0/5— 3	0/0/5— 3	**3/0/2— 1**	0/0/5— 3	0/0/5— 3	0/0/5— 3
n_x/2m-group	0/0/5— 3	2/0/3— 2	0/0/5— 3	0/0/5— 3	0/0/5— 3	**3/0/2— 1**	0/0/5— 3	0/0/5— 3	0/0/5— 3
n_x/m-group	0/0/5— 3	2/0/3— 2	0/0/5— 3	0/0/5— 3	0/0/5— 3	**3/0/2— 1**	0/0/5— 3	0/0/5— 3	0/0/5— 3
Non-separable	0/0/2— 2	0/0/2— 2	0/0/2— 2	0/0/2— 2	0/0/2— 2	**2/0/0— 1**	0/0/2— 2	0/0/2— 2	0/0/2— 2
1000 Dimensions (2013)									
Separable	0/0/3— 3	0/0/3— 3	0/0/3— 3	0/0/3— 3	0/0/3— 3	**2/0/1— 1**	1/0/2— 2	0/0/3— 3	0/0/3— 3
Additive	0/1/7— 4	**2/3/3— 1**	1/1/6— 3	0/5/3— 2	0/0/8— 5	0/0/8— 5	0/0/8— 5	0/0/8— 5	0/0/8— 5
Overlapping	0/0/3— 3	0/1/2— 2	0/0/3— 3	**2/0/1— 1**	0/0/3— 3	0/0/3— 3	0/1/2— 2	0/0/3— 3	0/0/3— 3
Non-separable	0/0/1— 2	**1/0/0— 1**	0/0/1— 2	0/0/1— 2	0/0/1— 2	0/0/1— 2	0/0/1— 2	0/0/1— 2	0/0/1— 2
2000 Dimensions									
Separable	0/0/3— 3	0/0/3— 3	0/0/3— 3	0/0/3— 3	0/0/3— 3	1/0/2— 2	**2/0/1— 1**	0/0/3— 3	0/0/3— 3
Single-group	0/0/5— 3	2/0/3— 2	0/0/5— 3	0/0/5— 3	0/0/5— 3	**3/0/2— 1**	0/0/5— 3	0/0/5— 3	0/0/5— 3
n_x/2m-group	1/0/4— 2	**2/0/3— 1**	0/0/5— 3	0/0/5— 3	0/0/5— 3	**2/0/3— 1**	0/0/5— 3	0/0/5— 3	0/0/5— 3
n_x/m-group	0/0/5— 3	2/0/3— 2	0/0/5— 3	0/0/5— 3	0/0/5— 3	**3/0/2— 1**	0/0/5— 3	0/0/5— 3	0/0/5— 3
Non-separable	0/0/2— 2	0/0/2— 2	0/0/2— 2	0/0/2— 2	0/0/2— 2	**2/0/0— 1**	0/0/2— 2	0/0/2— 2	0/0/2— 2

References

[1] Storn, R., & Price, K. (1997). Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization, 11(4)*, pp. 341-359.

[2] Shi, Y. J., Teng, H. F., & Li, Z. Q. (2005). Cooperative co-evolutionary differential evolution for function optimization. In *Advances in Natural Computation: First International Conference (ICNC 2005)*, Proceedings Part II, pp. 1080-1088. Springer Berlin Heidelberg.

[3] Douglas, J., Engelbrecht, A., & Ombuki-Berman, B. (2018). Merging and decomposition variants of cooperative particle swarm optimization: New algorithms for large scale optimization problems. In *Proceedings of the 2nd International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence*, pp. 70–77. ACM.

[4] McNulty, A., Ombuki-Berman, B., & Engelbrecht, A. (2022). Decomposition and Merging Co-operative Particle Swarm Optimization with Random Grouping. In *Swarm Intelligence: 13th International Conference (ANTS 2022)*, pp. 117-129. Cham: Springer International Publishing.

[5] Tang, K., Li, X., Suganthan, P. N., Yang, Z., & Weise, T. (2009). Benchmark functions for the CEC'2010 special session and competition on large scale global optimization (Technical Report). *Nature Inspired Computation and Applications Laboratory*, USTC, China.

[6] Clark, M., Ombuki-Berman, B., Aksamit, N., & Engelbrecht, A. (2022). Cooperative Particle Swarm Optimization Decomposition Methods for Large-scale Optimization. *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, Singapore, pp. 1582-1591, doi: 10.1109/SSCI51031.2022.10022095.

[7] Potter, M. A., & Jong, K. A. D. (1994). A cooperative coevolutionary approach to function optimization. In *International conference on parallel problem solving from nature*, pp. 249-257. Berlin, Heidelberg: Springer Berlin Heidelberg.

[8] Yang, Z., Tang, K., & Yao, X. (2007). Differential evolution for high-dimensional function optimization. In *2007 IEEE Congress on Evolutionary Computation*, pp. 3523-3530. IEEE.

[9] Wang, C., Xu, M., Zhang, Q., Jiang, R., Feng, J., Wei, Y., & Liu, Y. (2022). Cooperative co-evolutionary differential evolution algorithm applied for parameters identification of lithium-ion batteries. *Expert Systems with Applications, 200*, pp. 1-18. https://doi.org/10.1016/j.eswa.2022.117192

[10] Kruskal, W. H., & Wallis, W. A. (1952). Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association, 47*(260), pp. 583–621.

[11] Van den Bergh, F., & Engelbrecht, A. P. (2004). A Cooperative approach to particle swarm optimization. *IEEE Transactions on Evolutionary Computation, 8*(3), pp. 225-239. doi:10.1109/TEVC.2004.826069.

[12] Mallipeddi, R., Suganthan, P. N., Pan, Q. K., & Tasgetiren, M. F. (2011). Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied Soft Computing, 11*(2), pp. 1679-1696.

[13] Li, X., Tang, K., Omidvar, M. N., Yang, Z., Qin, K., & China, H. (2013). Benchmark functions for the CEC 2013 special session and competition on large-scale global optimization. *Gene, 7*(33), pp. 1-8.

[14] Conover, W. J., & Iman, R. L. (1979). *Multiple-comparisons procedures. Informal report* (No. LA-7677-MS). Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[15] Dinno, A. (2017). *Conover-Iman test of multiple comparisons using rank sums* [Online]. Vienna: R Foundation for Statistical Computing.

[16] Montgomery, J., & Chen, S. (2010). An analysis of the operation of differential evolution at high and low crossover rates. In *IEEE Congress on Evolutionary Computation*, pp. 1-8. IEEE.

[17] Segura, C., Coello Coello, C. A., Segredo, E., & León, C. (2015). On the adaptation of the mutation scale factor in differential evolution. *Optimization Letters, 9*, pp. 189-198.

[18] Slowik, A., & Kwasnicka, H. (2020). Evolutionary algorithms and their applications to engineering problems. *Neural Computing & Applications, 32*, pp. 12363–12379.

[19] Ponsich, A., A. L. Jaimes, & C. A. C. Coello. (2013). A Survey on Multiobjective Evolutionary Algorithms for the Solution of the Portfolio Optimization Problem and Other Finance and Economics Applications. *IEEE transactions on evolutionary computation, 17(3)* pp. 321–344.

[20] Buzdalov, M., Kolyubin, S., Egorov, A., & Borisov, I. (2020) Optimizing Robotic Cheetah Leg Parameters Using Evolutionary Algorithms. *Bioinspired Optimization Methods and Their Applications, 12438* pp. 214–227.

[21] Maučec, M.S., & Brest, J. (2019). A review of the recent use of Differential Evolution for Large-Scale Global Optimization: An analysis of selected algorithms on the CEC 2013 LSGO benchmark suite *Swarm and Evolutionary Computation, 50*, pp. 1-18.

[22] Kennedy, J., & Eberhart, R. (1995, November). Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks* (Vol. 4, pp. 1942-1948). ieee.

[23] Karaboga, D. (2005). *An idea based on honey bee swarm for numerical optimization* (Vol. 200, pp. 1-10). Technical report-tr06, Erciyes university, engineering faculty, computer engineering department.

[24] Karaboga, D., & Basturk, B. (2008). On the performance of artificial bee colony (ABC) algorithm. *Applied soft computing, 8*(1), 687-697.