International Journal of Mathematical, Engineering and Management Sciences
Vol. 10, No. 4, 1055-1079, 2025
https://doi.org/10.33889/IJMEMS.2025.10.4.051

Ram Arti Publishers

Current-to-Best Crossover for Modified Jaya Algorithm

Sandeep Kumar Mogha
Department of Mathematics,
Chandigarh University, Mohali, Punjab, India.
Corresponding author: moghadma@gmail.com

Sonal Deshwal
Department of Mathematics,
Chandigarh University, Mohali, Punjab, India.
E-mail: sonaldeswal1995@gmail.com

Pravesh Kumar
Department of Mathematics,
Rajkiya Engineering College, Bijnor, Uttar Pradesh, India.
E-mail: praveshtomariitr@gmail.com

(Received on September 14, 2024; Revised on December 12, 2024; Accepted January 28, 2025)

Abstract
The Jaya algorithm is a recently designed metaheuristics and has been verified for its good performance in solving various engineering and scientific optimization applications. In this paper, an enhanced variant named 'MCrJaya' has been developed by incorporating the *DE/current-to-best* approach-based crossover operation at the creation phase and the *DE/*α_{best}/1 mutation operation at the selection phase. The newly designed crossover operation manages the population diversity, while the second approach helps to amplify the convergence rapidity of the algorithm. The proposed MCrJaya is verified on 54 benchmark problems from different test suits and three real-life applications of IEEE CEC 2011. The performance was evaluated using three metrics: mean, standard deviation, and rank. Additionally, statistical analysis was performed using the Wilcoxon rank-sum test to assess the significance of the results, further validating the robustness of the proposed approach. The experimental results confirm the superiority of the MCrJaya over enhanced variants of Jaya, DE, and other metaheuristics algorithms.

Keywords- Crossover, DE algorithm, Jaya algorithm, Mutation, Optimization.

1. Introduction

Optimization can be defined as the procedure of determining the finest solution for a particular problem. Usually, lots of the practical optimization problems that occur in engineering and scientific fields may have discontinuous, non-derivative, nonlinear, and multi-model-based objective functions, and hence it may be a very difficult task for the conventional methods to handle them.

Over the last few years, a number of non-conventional intelligent methods have been offered for solving such optimization problems. These non-conventional methods can be classified mainly into four categories: (i) Evolutionary-based methods such as Genetic Algorithm (Wright, 1991), Differential Evolution (Storn & Price, 1997), Jaya Algorithm (Rao, 2016), Tree Seed Algorithm (TSA) (Kiran, 2015) etc.; (ii) Swarm-based methods such as Particle Swarm Optimization (Kennedy & Eberhart, 1995), Artificial Bee Colony (Karaboga & Basturk, 2007), Grey Wolf Optimization (Mirjalili et al., 2014), Whale Optimization Algorithm (Mirjalili & Lewis, 2016), Tunicate Swarm Algorithm (Kaur et al., 2020), Remora Optimization Algorithm (Jia et al., 2021), Reptile Search Algorithm (Abualigah et al., 2022), Manta Ray Foraging Optimization (Zhao et al., 2020), Red Fox Optimization (RFO) (Połap & Woźniak, 2021), Golden Jackal Optimization (GJO) algorithm (Chopra & Ansari, 2022) etc., (iii) Human-inspired-based methods such as

Brain Storm Optimization (Shi, 2011), Teaching Learning Based Optimization (Rao et al., 2011), Gaining and Sharing Knowledge (Mohamed et al., 2020), Teamwork Optimization (TO) (Dehghani & Trojovský, 2021), Political Optimizer (PO) (Askari et al., 2020) etc., and Physics-based methods such as Gravitational Search Algorithm (Rashedi et al., 2009), Sine Cosine Algorithm (Mirjalili, 2016), Atom Search Optimization (Zhao et al., 2019), Archimedes Optimization Algorithm (AOA) (Hashim et al., 2021) etc.

The Differential Evolution (DE) algorithm was introduced by Storn and Price (1997). It is still popular due to its various benefits, like its dense structure, the requirement of few control parameters, etc., and has been widely used in solving many applications.

Generally, the efficiency of DE effects depends on the mutation strategy and control parameters. Therefore, a lot of research has been carried out in this area. Brest et al. (2006) provides idea for self-adaptive control parameter settings to improve the performance. Qin et al. (2009) proposed self-adaptive based DE (SaDE) which adapt mutation strategy and control parameter value by their past search memories. Zhang and Sanderson (2009) designed JADE in which an adaptive DE with optional external archive method used for adapting control parameters and mutation strategies. Gong et al. (2011) have proposed APadapJADE-SS by including an adaptive strategy selection with JADE. Later Tanabe and Fukunaga (2013) proposed SHADE by using historical memory of successful control parameter for the next generation. The idea of SHADE was also successfully extended in many research like LSHADE (Tanabe & Fukunaga, 2014), iLSHADE (Brest et al., 2016) etc.

Some other admirable and fresh DE variants are including DE with modified random localization (MRLDE) (Kumar & Pant, 2012), A Cluster-based population initialization method for DE (Poikolainen et al., 2015), DE with adaptive approach (MPADE) (Cui et al., 2016), Collective information based modified DE (CIPDE) (Zheng et al., 2017), DE with global optimum search method (DEGOS) (Yu et al., 2019b), Parameter adaptation based DE (PaDE) (Meng et al., 2019), Compound sinusoidal DE (SinDE) (Draa et al., 2019), Multi population based inflationary DE (IDEA) (Di Carlo et al., 2020), Improved Multi-operator based DE (IMODE) (Sallam et al., 2020), Coperative strategy based DE (CS-DE) (Meng et al., 2021), Fitness-based adaptive DE (FADE) (Xia et al., 2021), Neighborhood mutation operators based DE (NBOLDE) (Deng et al., 2021), Ensemble operators based adaptive DE (Yi et al., 2022), Two-stage approach based DE (TDE) (Meng & Yang, 2022), Permutation archive information based DE (PAIDDE) (Li et al., 2022), Exploration and exploitation control based DE (TRADE) (Cai et al., 2023) and so on. A well-organized literature analysis for DE algorithms can also be found in (Bilal et al., 2020).

Jaya algorithm is a population-based intelligent algorithm and designed by Rao (2016). Like other evolutionary algorithms, it also supports the Darwin's law of 'fittest survival'. The working policy in the Jaya is very easy and the movements go straight to the finest position while it goes away from the bad position. Consequently, it improves the solution quality by replacing the old solutions with the better solutions in the subsequent iterations.

The simple structure and requirement of only few essential control parameters make it more appropriate for solving lots of real-life applications such as Grinding of surface process problem (Rao et al., 2016a), Power flow optimization (Warid et al., 2016), Classification of automatic unsupervised data (Kurada & Kanadam, 2016), Engineering design problems (Rao & Waghmare, 2017), Optimization of PV-DSTATCOM (Mishra & Ray, 2016), Micro-channel heat sink's dimensional optimization (Rao et al., 2016b), Optimization of STHE design (Rao & Saroj, 2017), Traffic light scheduling problem (Gao et al., 2017), Optimization of coordination in over current relays (Wadood et al., 2019).

In recent years, numerous modified Jaya variants have also been designed for solving optimization problems e.g. Yu et al. (2017) have proposed an Improve Jaya by introducing a self-adaptive weight adjustment method and used it to identify the photovoltaic models parameters. Gong (2017) has separated all the candidates in two groups by their fitness value and then used the better and worst group mean to improve the Jaya's performance. Rao and Keesari (2018) have improved the Jaya's performance by giving multi team based perturbation idea in MTPG-Jaya algorithm and tested it on different cases of wind farm layout optimization. Solving ELDP by self adaptive Jaya, Yu et al. (2019a) have proposed Jaya-SML by incorporating Levy flight distribution and a self-adaptive strategy for tuning the extra parameter and apply it to solve ELD problem. Solving knapsack problem by Wu & He (2020) have presented double coding and Cauchy mutation based DHJaya and implemented it in solving a set knapsack problems. Zhang et al. (2021) have proposed a tuning free enhanced Jaya (EJaya) where the upper and lower attractors guide the local exploitation and the historical population is used to guide the global exploration. Kumar & Sharma (2022) have added additional selection operation to Jaya algorithm by modified randomised localization based mutation and presented enhanced variants MRL-Jaya for solving unconstrained optimization.

Besides several benefits, Jaya also has some deficiencies mainly facing of slow convergence speed and population stagnation situation. Hence there are requirements of some additional modifications within its structure to overcome these deficiencies.

In this paper, a modified Jaya algorithm named *MCrJaya* is proposed. The *MCrJaya* modifies Jaya in two ways; first there is a crossover operation is designed for Jaya to maintain the population diversity. This crossover operation has taken from DE algorithm from its *current-to-best* mutation method and called it *CrJaya* algorithm. The crossover builds a stable exploration and exploitation ability in Jaya and helps to improve the population diversity.

Next, a $DE/\alpha_{best}/1$ mutation operation has been implemented in the selection process which generates a new vector when Jaya's created vector is rejected to be chosen for the next generation. This approach keeps away from getting stuck in local optima and makes an effective improvement in the algorithm's performance.

Rest of the paper is presented as follow: Section 2 provides the fundamental of DE and Jaya algorithm. The description of the proposed *MCrJaya* is presented in Section 3. The benchmark problems are described in Section 4. The performance assessment of proposed methods is analyzed and discussed in Section 5 and finally the conclusion and future scope is reported in Section 6.

2. Basic DE and Jaya

2.1 Jaya Algorithm

Designed by Rao (2016), Jaya algorithm is a population-based direct search way for solving optimization problems with continuous parameters. It works mainly in three phases: initializing, creating, and updating new locations. These phases are defined below in detail:

(a) *Initializing*: Jaya begins with a randomly produced population by uniform distribution in a bounded domain. If $S = \{s_{i,j}: i = 1,2 \ldots m; j = 1,2, \ldots n)$ is an m-size population of n-dimensional vectors then it can be initially generated by Equation (1) in domain $[s_l, s_u]$ where s_l and s_u represents the lower and upper bound limit respectively for $s_{i,j}$,

$$s_i(0) = s_l + r_{(0,1)} \times (s_u - s_l) \quad (1)$$

and $r_{(0,1)}$ is a uniform random number in bound of 0 and 1.

(b) *Creating*: After initializing, a new location is created for s_i by adding and subtracting a random difference from its present best and worst locations, respectively. Let $s_b(g)$ is best and $s_w(g)$ are and worst vectors, respectively, in existing generation g, then the new location, $s'_i(g)$ for vector $s_i(g)$ is created by Equation (2) as below:

$$s'_i(g+1) = s_i(g) + r^b_{(0,1)} \times (s_b(g) - |s_i(g)|) - r^w_{(0,1)} \times (s_w(g) - |s_i(g)|) \quad (2)$$

where, $r^b_{(0,1)}$ and $r^b_{(0,1)}$ are uniform random numbers in bound of 0 and 1.

(c) *Updating*: The newly created location $s'_i(g+1)$ will be selected for the next generation if it has better function value than the old location, $s_j(g)$, as defined in Equation (3):

$$s_i(g+1) = \begin{cases} s'_i(g+1), \rightleftarrows iff(s'_i(g+1)) < f(s_i(g)) \\ s_i(g) else \end{cases} \quad (3)$$

The above process will continue until the closing criteria are met.

A step by step pseudo-code of Jaya algorithm is given in *Algorithm*-1 below:

Algorithm*-1: *Jaya Algorithm

1) Set parameters value of *population size (m) and dimension (n), Max Generation (G)*
2) Random generate initial population $S = \{s_{i,j}\}$ by Equation-1
3) Estimate function value *f(*s_i*)* for each s_i
4) *while g<G*
5) Obtain best and worst locations
6) *for i=1:m*
7) Obtain new position s'_i by Equation-2
8) Update position s_i by Equation-3
9) *end for*
10) *G=g+1*
11) *end while*

2.2 Differential Evolution Algorithm

The differential evolution algorithm (DE) was designed by Storn & Price (1997). It is an eminent and directly investigated stochastic technique for dealing with complicated optimization problems. After generating a random population, it works through the following phases:

(a) *Mutation*: In this phase, a new position for vectors is created by using the differences between some other randomly selected vectors from the population. Some different mutation approaches are given in Equation (4):

DE/rand/1: $p_i(g+1) = s_{a_1}(g) + F \times (s_{a_2}(g) - s_{a_3}(g))$

DE/best/1: $p_i(g+1) = s_{best}(g) + F \times (s_{a_2}(g) - s_{a_3}(g))$

DE/rand/2: $p_i(g+1) = s_{a_1}(g) + F \times (s_{a_2}(g) - s_{a_3}(g)) + F \times (s_{a_4}(g) - s_{a_5}(g))$

DE/best/2: $p_i(g+1) = s_{best}(g) + F \times (s_{a_2}(g) - s_{a_3}(g)) + F \times (s_{a_4}(g) - s_{a_5}(g))$

$$\text{DE/rand-best:} \quad p_i(g+1) = s_{a_1}(g) + F \times (s_{best}(g) - s_{a_2}(g)) + F \times (s_{a_3}(g) - s_{a_4}(g))$$
$$\text{DE/ curr} - \text{best:} \quad p_i(g+1) = s_i(g) + F \times (s_{best}(g) - s_i(g)) + F \times (s_{a_2}(g) - s_{a_3}(g)) \tag{4}$$

Here a_1, a_2, a_3, a_4 and a_5 are random chosen integer from $[1, m]$. The parameter $F \in (0,2)$ is used to handle the difference of vectors.

(b) *Crossover*: This phase generates a trial vector, say $c_i^{(g+1)} = \{c_{i,j}: j = 1,2,\dots n\}$ by intersect of the target and mutant vector as defined in Equation (5):

$$c_{i,j}^{(g+1)} = \begin{cases} p_{i,j}^{(g+1)}, if(r_{(0,1)}^{cr} < cr || j == k): k \in randi(1,n) \\ s_{i,j}^{(g)}, otherwise \end{cases} \tag{5}$$

Here parameter $cr \in (0,1)$ is known as crossover rate and k decides that at least one component in trail vector $c_j^{(g+1)}$ will be come from $p_i^{(g+1)}$

(c) *Selection*: Finally, a tournament-based-selection between $c_i^{(g+1)}$ and $s_i^{(g)}$ decides whether a newly generated trail vector $c_i^{(g+1)}$ will be chosen or not for the next generation.

$$s_i^{(g+1)} = \begin{cases} c_i^{(g+1)}, iff(c_i^{(g+1)}) < f(s_i^{(g)}) \\ s_i^{(g)}, otherwise \end{cases} \tag{6}$$

3. Proposed MCrJaya

In *MCrJaya*, the Jaya algorithm is modified by the DE algorithm in two ways: first by introducing a crossover operation with the *DE/current-to-best* approach, and second by designing an enhanced selection operation with the *DE/α-best/1* approach.

Next, the proposed modifications are defined in detail:

3.1 DE/*current-to-best* Approach Based Crossover for Jaya (CrJaya)

In the DE algorithm, mutation and crossover phases are considered key operations and provide an excellent searching facility to the algorithm by maintaining proper population diversity. On the other side, the progression by the Jaya algorithm is based on movements of solutions toward the optimal location directly, and hence there is a high possibility of being trapped in a local optima and facing a population stagnation problem in a complex and high-dimensional case.

This population stagnation deficiency can be reduced by including a crossover operation in the algorithm. Hence, a *DE/current-to-best* mutation-based approach is selected for crossover operation. Similar to the Jaya algorithm, the movement in this approach is also based on the difference between the individual and the current best location. The newly designed variant is termed CrJaya and the j^{th} component of the i^{th} trial vector,$s'^{(g+1)}_i = s'_{i,j}: j = 1,2,\dots n\}$ is generated as defined in Equation (7):

$$s_{i,j}(g+1) = \begin{cases} s_{i,j}(g) + r_{(0,1)}^{b} \times (s_{b,j}(g) - |s_{i,j}(g)|) - r_{(0,1)}^{w} \times (s_{w,j}(g) - |s_{i,j}(g)|), if r_{(0,1)}^{cb} < CR_{cb} \rightleftarrows \\ s_{i,j}(g) + F_{cb} \times (s_{b,k}(g) - s_{i,j}(g)) + F_{cb} \times (s_{a_1,k}(g) - s_{a_2,k}(g)): otherwise \end{cases} \tag{7}$$

where, s_{a_1}, s_{a_2} and s_{a_3} are randomly selected vectors from the population, s_b and $s_w$are the best and worst location respectively, $r^b_{(0,1)}$, $r^w_{(0,1)}$,$r^{cb}_{(0,1)}$ are uniform random numbers and CR_{cb} and F_{cb} are control parameters contain a value between 0 & 1.

The movement of any vector s_j by Jaya and CrJaya is illustrated graphically in **Figure 1** in the 2-*D* region. The new locations of the vector created by CrJaya are likely to be on the vertices of the rectangle, as represented by green dots. From the graph, it is clear that the designed crossover improves the exploration capability of Jaya and hence helps to minimize the population stagnation deficiency.

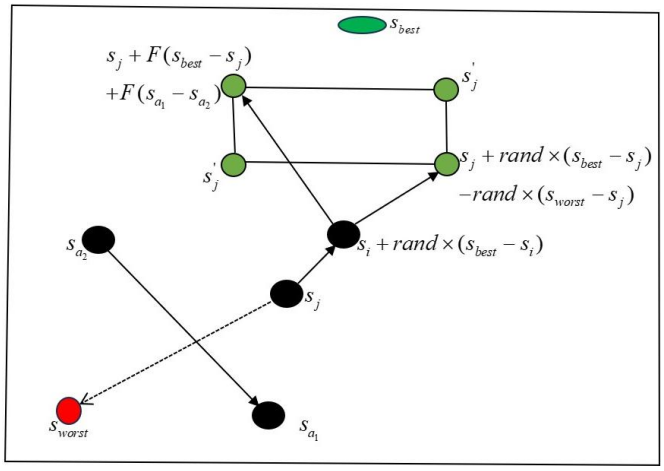

Figure 1. Graphical presentation of movement of vector in *CrJaya*.

Algorithm-2: CrJaya Algorithm

1) Input parameters: *m, n, G, CR_{cb}, F_{cb}*
2) Random generate initial population $S = \{s_{i,j}\}$ by Equation-1
3) Estimate function value *f(s_i)* for each s_i
4) *while g < G*
5) Obtain best and worst locations
6) *for i=1:m*
7) Select two random individuals: s_{a_2}&$s_{a_3} \in S$
8) *for j=1:n*
9) *if* $(rand() < CR_{cb})$
10) $s'_{i,j} = s_{i,j} + rand \times (s_{b,j} - s_{i,j}) - rand \times (s_{w,j} - s_{i,j})$
11) *Else*
12) $s'_{i,j} = s_{i,j} + F_{cb} \times (s_{b,j} - s_{i,j}) + F_{cb} \times (s_{a_2,j} - s_{a_3,j})$
13) *end if*
14) *end for*
15) *if* $f(s'_i) \leq f(s_i)$
16) $s_i = s_i'$
17) Update best location
18) *end if*
19) *end for*
20) *g=g+1*
21) *end while*

3.2 Modified Selection by *DE/α-best/1* Mutation

The second phase in *MCrJaya* is to modify the selection operation by the *DE/α-best/1* mutation approach. The idea of modifying the selection operation is inspired by our other algorithm, MRL-JAYA (Kumar & Sharma, 2022), where the *MRLDE* algorithm was used to modify this operation.

The *DE/α$_{best}$/1* approach is a modified form of the *DE/rand/1* mutation strategy where the basis vector s_{a_1} is selected from the set of *α%* best individual instead of randomly chosen from the entire population.

As it is known, the *DE/rand/1* strategy is considered to be slow while the *DE/best/1* strategy is considered to be a little greedy in nature. Therefore, we can assume that the *DE/α$_{best}$/1* approach acts as a bridge between the *DE/rand/1* and *DE/best/1* mutation strategies and utilizes the advantages of both in a single method.

The working structure of this approach is very easy. If the generated position $s'_i(g+1)$ is not fitter than the previous position $s_i(g)$, than we apply the *DE/α$_{best}$/1* approach to generate a new position $s'_i(g+1)$ again. First, we select a random vector say, s_{a_1} from the top *α-best* vectors and two other different random vectors, say s_{a_2} and s_{a_3} from the population *S*, and generate $s'^{(g+1)}_i = s'_{i,j}: j = 1,2,\ldots n\}$ whose j^{th} component will be generated by Equation (8):

$$s'_{i,j}(g+1) = \begin{cases} s_{a_1,j}(g) + F_{mc} \times (s_{a_2,j}(g) - s_{a_3,j}(g)) : if r^{cb}_{(0,1)} < CR_{mc} \rightleftarrows \\ s_{i,j}(g) : otherwise \end{cases} \quad (8)$$

As discussed in MCrJaya, the modified selection process like this can help avoid getting stuck in local optima by taking advantage of both Jaya and DE algorithms in one place, and consequently, the performance of the algorithm will also be improved.

To understand the working structure, a flowchart of MCrJaya is presented in **Figure 2**

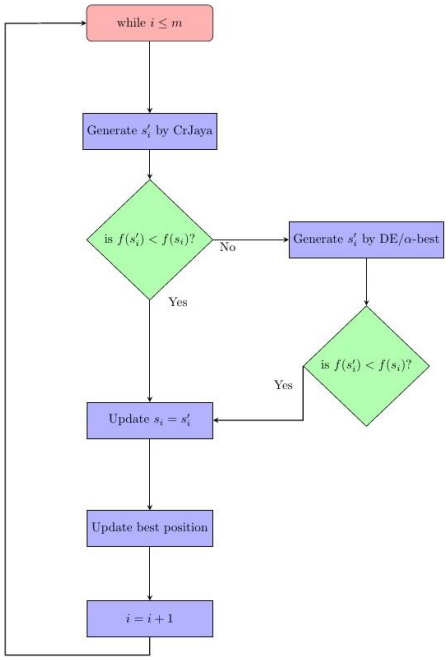

Figure 2. The flowchart of the proposed MCrJaya.

The complete working of *McrJaya* is explained in Algorithm-3 below:

Algorithm-3: MCrJaya Algorithm

1) Input parameters: *m, n, G, CR_{cb}, F_{cb}, CR_{mc}, F_{mc}*
2) Random generate initial population $S = \{s_{i,j}\}$ by equation-1
3) Estimate function value *f(s_i)* for each s_i
4) *while g<G*
5) Obtain best and worst locations
6) *for i=1:m*
7) Select two random individuals: s_{a_1}&$s_{a_2} \in S$
8) *for j=1:n*
9) Generate position s'_i by equation-7 //(*CrJaya)*
10) *end for*
11) *if* $f(s'_i) \leq f(s_i)$
12) $s_i = s_i'$
13) *Else*
14) *for j=1:n*
15) Generate position s'_i by equation-8 // *DE/α_{best}/1)*
16) *end for*
17) *if* $f(s'_i) \leq f(s_i)$
18) $s_i = s_i'$
19) Update best position
20) *end if*
21) *end for*
22) *g=g+1*
23) *end while*

3.3 Algorithm's Complexity

The computational complexity determines the operating effectiveness of an algorithm. In *Jaya*, the computational complexity in the initialization phase is $O(m \times n)$ while it is $O(G \times (m \times (n+n))$ or $O(2 \times G \times m \times n)$ in the creating and selecting phases. So, the computational complexity of Jaya will be $O(m \times n \times (2 \times G + 1))$. Similarly, the complexity of the *CrJaya* will also be $O(m \times n \times (2 \times G + 1))$ as the modification has been done inside the dimension loop of the creating phase. In MCrJaya, the complexity in creating and selecting phases will be $O(G \times (m \times (n+n+n)))$ due to the association of an additional loop for the dimension in the selection operation, and hence complexity of the *MCrJaya* will slightly increase as $O(m \times n \times (3 \times G + 1))$.

4. Benchmark Functions

Due to the limited knowledge of evolutionary algorithms, we used benchmark functions to evaluate the performance of algorithms. In this paper, 54 unconstrained benchmark functions have been taken from various literatures of the following types:

$Minf(s)$: $s = [s_1, s_2, \ldots, s_n] \in R^n; s_i \in (s_l, s_u)$

where, $s_l$and s_u are lower and upper bounds respectively for s $_i$.

4.1 Classical and Shifted Functions

The classical and shifted benchmarks functions are presented in **Table 1**. Here the $TF_1 - TF_9$ are unimodal, $TF_{10} - TF_{19}$ are multimodal and $TF_{20} - TF_{25}$ are shifted functions. The shifted functions are taken from CEC2008 which have taken to check the algorithm's compatibility on complex functions.

4.2 CEC-2017 Functions

29 functions of CEC2017 have also been selected to test the algorithm's capabilities on some more complicated problems. CEC2017 is a famous test suit for having extremely complex, hybrids and compound functions. The details of these functions are specified in Awad et al. (2016).

Table 1. Classical and shifted functions with mathematical models.

Test function	Limits
$TF_1 = \sum_{i=1}^{n} s_i^2$	$[-100,\ 100]^n$
$TF_2 = \sum_{i=1}^{n} \lvert s_i \rvert + \prod_{i=1}^{n} \lvert s_i \rvert$	$[-10,\ 10]^n$
$TF_3 = \sum_{i=1}^{n} \left(\sum_{k=1}^{i} s_k \right)^2$	$[-100,\ 100]^n$
$TF_4 = max_i\{\lvert s_i \rvert\}$	$[-100,\ 100]^n$
$TF_5 = \sum_{i=1}^{n} i s_i^2$	$[-5,\ 5]^n$
$TF_6 = 10^6 s_1^2 + \sum_{i=2}^{n} s_i^2$	$[-100,\ 100]^n$
$TF_7 = \sum_{i=1}^{n} s_i^2 + \sum_{i=1}^{n} i s_i^4$	$[-100,\ 100]^n$
$TF_8 = \sum_{i=1}^{n} (i s_i)^2$	$[-100,\ 100]^n$
$TF_9 = \sum_{i=1}^{n} s_i^2 + \sum_{i=1}^{n} (0.5 s_i)^2 + \sum_{i=1}^{n} (0.5 s_i)^4$	$[-5,\ 10]^n$
$TF_{10} = \sum_{i=1}^{n/4} ((s_{4i-3} + 10 s_{4i-2})^2 + 5(s_{4i-1} - s_{4i})^2 + (s_{4i-2} - 2 s_{4i-1})^4 + 10(s_{4i-3} - s_{4i})^4)$	$[-10,\ 10]^n$
$TF_{11} = \sum_{i=1}^{n-1} [100(s_{i+1} - s_i^2)^2 + (s_i - 1)^2]$	$[-30,\ 30]^n$
$TF_{12} = \sum_{i=1}^{n} \lvert s_i + 0.5 \rvert^2$	$[-100,\ 100]^n$
$TF_{13} = \sum_{i=1}^{n} i s_i^4 + rand(0\ 1)$	$[-1.28,\ 1.28]^n$
$TF_{14} = \sum_{i=1}^{n} -s_i \sin \sqrt{\lvert s_i \rvert} + 418.982887n$	$[-500,\ 500]^n$
$TF_{15} = 10n + \sum_{i=1}^{n} (s_i^2 - 10 \cos(2\pi s_i))$	$[-5.12,\ 5.12]^n$
$TF_{16} = -20 \exp\left(-0.2 \sqrt{1/n \sum_{i=0}^{n} s_i^2} \right) - \exp\left(\sqrt{1/n} \sum_{i=0}^{n} \cos(2\pi s_i) \right) + 20 + e$	$[-32,\ 32]^n$

Table 1 continued…

$TF_{17} = \frac{1}{4000}\sum_{i=1}^{n\sum\left(\frac{s_i}{\sqrt{i}}\right)} s_i^2 - \prod_{i=1}^{n[]} cos$	$[-600,\ 600]^n$
$TF_{18} = \frac{\pi}{n}\left\{\begin{matrix}10\,sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i-1)^2[1+10\,sin^2(\pi y_{i+1})] \\ +(y_D-1)^2\end{matrix}\right\} + \sum_{i=1}^{D} u(s_i,\ 10,\ 100,\ 4)$ $where\ \ y_i = 1+\frac{1}{4}(s_i+1)\ and u(s_i,a,k,m) = \begin{cases}k(s_i-a)^m\ if\ s_i > a \\ 0\ if\ -a \le s_i \le a \\ k(-s_i-a)^m\ if\ s_i < -a\end{cases}$	$[-50,\ 50]^n$
$TF_{19} = 0.1\left\{\begin{matrix}sin^2(3\pi s_1) + \sum_{i=1}^{n-1}(s_i-1)^2[1+sin^2(3\pi s_{i+1})] \\ +(s_n-1)^2[1+sin^2(2\pi s_n)]\end{matrix}\right\} + \sum_{i=1}^{n} u(s_i,5,\ 100,\ 4)$	$[-50,\ 50]^n$
$TF_{20} = \sum_{i=1}^{n} y_i^2 - 450 : s.t. y = s - o$	$[-100,\ 100]^n$
$TF_{21} = \underset{i}{max}\{\lvert y_s\rvert, 1 \le i \le n\} - 450; s.t.: \rightleftarrows y = x - o,$	$[-100,\ 100]^n$
$TF_{22} = \sum_{i=1}^{n}(100(y_i^2 - y_{i+1})^2 + (y_i-1)^2) + 390 : s.t. y = s - o$	$[-100,\ 100]^n$
$TF_{23} = \sum_{i=1}^{n}(y_i^2 - 10\,cos(2\pi y_i) + 10) - 330 : s.t.: \rightleftarrows y = s - o$	$[-5.12,\ 5.12]^n$
$TF_{24} = \left(\sum_{i=1}^{n}\frac{y_i^2}{4000} - \prod_{i=1}^{n} cos(\frac{y_i}{\sqrt{i}}) + 1\right) - 180 : s.t.: \rightleftarrows y = s - o$	$[-600,\ 600]^n$
$TF_{25} = \left(-20\,exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} y_i^2}) - exp(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi y_i)) + 20 + e\right) - 140 : s.t.: y = s - o;$	$[-32,\ 32]^n$

5. Results and Discussion

5.1 Performance Assessment of Proposed Modifications

This section presents the performance assessment of newly designed modifications CrJaya and MCrJaya. All experiments are executed by put on the following parameters settings:

- *For Jaya*, CrJaya and MCrJaya: $m = 100, CR_{mc} = 0.9, CR_{cb} = 0.5, F_{cb} = F_{mc} = 0.5, \alpha = 20\%$.
- *System Configuration:* CPU: 2.6-GHZ, Processor: intel core™-i3, 64-bit, RAM: 8GB and OS: window-10.
- *Software Used:* Matlab 2019, SPSS.

In **Table 2**, the experimental results are listed for average accuracy $|f(s) - f(s*)|$ and standard deviation of 50 free runs obtained in 1500 iterations.

It can be simply examined the effective improvement in the solution quality by adding the crossover operation on *Jaya*. For example, *Jaya* gives an average accuracy as 1.41E-04 for function $TF_1(n = 30)$ which is enhanced to 4.45E-32 by CrJaya. Similarly, the average accuracy for the same function $TF_1(n = 50)$ is increased from 3.08E-01 to 1.91E-16 by CrJaya. In the same manner, we can examine a major enhancement of CrJaya in solution precision for other functions also.

It can also be observed that the proposed MCrJaya provides more enhanced solution quality. For instance, the average accuracy obtained by MCrJaya for the function TF_1, are 3.15E-84 and 1.29E-47 for n = 30 and

50 respectively. All three algorithms give equal results for function TF_{12} while CrJaya and MCrJaya perform identical for function TF_{18} and TF_{19} in both cases of n = 30 and 50.

Table 2. Numerical results of proposed modifications in terms of average accuracy (Standard deviation).

Fun	*n* = 30			*n* = 50		
	Jaya	**CrJaya**	**MCrJaya**	**Jaya**	**CrJaya**	**MCrJaya**
TF_1	1.41E-04 (4.48E-05)	4.45E-32 (1.12E-32)	**3.15E-84 (4.58E-84)**	3.08E-01 (6.76E-02)	1.91E-16 (9.97E-17)	**1.29E-47 (7.37E-47)**
TF_2	6.78E-03 (8.79E-03)	1.46E-15 (6.92E-15)	**2.02E-43 (2.06E-43)**	9.17E-01 (3.64E-02)	1.94E-07 (3.32E-08)	**3.32E-27 (1.38E-27)**
TF_3	2.48E+03 (4.55E+02)	1.12E+01 (3.83E+00)	**3.89E-14 (2.17E-14)**	1.93E+04 (3.84E+03)	7.52E+03 (4.51E+03)	**7.34E-03 (3.82E-03)**
TF_4	6.60E-01 (1.91E-01)	2.58E-05 (6.214e-06)	**8.52E-21 (2.56E-21)**	8.62E+00 (1.34E+00)	2.92E-01 (3.34E-02)	**1.26E+01 (1.09E-01)**
TF_5	1.26E-06 (3.21E-07)	1.69E-36 (4.86E-36)	**2.30E-87 (1.07E-87)**	2.32E-02 (8.85E-03)	7.74E-18 (3.15E-18)	**4.59E-50 (2.61E-50)**
TF_6	1.93E-04 (2.47E-05)	1.73E-32 (5.91E-33)	**9.63E-82 (8.81E-82)**	7.82E-01 (9.35E-01)	3.22E-16 (1.41E-16)	**1.71E-46 (3.25E-46)**
TF_7	3.18E-03 (1.16E-04)	2.25E-31 (7.02E-32)	**1.19E-80 (8.03E-80)**	4.32E+01 (1.82E+01)	3.17E-14 (2.28E-15)	**3.20E-44 (3.90E-44)**
TF_8	8.05E-03 (1.03E-03)	1.11E-32 (1.06E-32)	**9.51E-83 (2.36E-83)**	1.82E+02 (2.23E+01)	1.52E-14 (1.09E-15)	**8.15E-46 (7.02E-47)**
TF_9	1.12E-04 (7.22E-05)	6.87E-32 (4..62E-33)	**5.01E-81 (6.10E-81)**	2.82E+00 (1.23E+00)	2.00E-15 (1.08E-16)	**5.28E-46 (4.99E-46)**
TF_{10}	2.37E-02 (2.23E-02)	4.71E-04 (1.15E-05)	**7.06E-08 (3.18E-07)**	5.62E+00 (3.68E+00)	2.99E-03 (2.01E-04)	**4.38E-05 (5.28E-05)**
TF_{11}	2.58E+01 (5.81E+01)	1.44E+00 (1.52E+00)	**8.59E-11 (2.56E-10)**	1.78E+02 (5.71E+01)	3.42E+01 (5.51E-01)	**2.42E+01 (1.06E+00)**
TF_{12}	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00E+00 (0.00E+00)**	**0.00 (0.00)**	**0.00 (0.00)**
TF_{13}	2.81E-02 (7.76E-03)	6.38E-03 (1.51E-03)	**1.35E-03 (1.21E-03)**	1.31E-01 (8.87E-02)	1.84E-02 (3.46E-03)	**9.06E-03 (5.04E-03)**
TF_{14}	7.75E+03 (2.62E+02)	7.58E+03 (2.78E+02)	**1.76E+03 (3.16E+02)**	1.49E+04 (2.14E+02)	1.41E+04 (1.02E+02)	**5.65E+03 (4.12E+01)**
TF_{15}	2.01E+02 (2.32E+01)	1.69E+02 (3.19E+00)	**2.11E+01 (5.01E+00)**	4.07E+02 (2.21E+01)	3.88E+02 (1.51E+01)	**4.02E+01 (7.43E+01)**
TF_{16}	4.26E-03 (1.12E-04)	6.96E-15 (0.00)	**4.11E-15 (1.42E-15)**	2.21E-01 (7.75E-01)	1.47E-09 (1.42E-09)	**1.05E-14 (5.97E-13)**
TF_{17}	2.09E-01 (2.42E-01)	9.01E-02 (8.35E-02)	**1.08E-19 (4.21e-19)**	6.21E-01 (1.41E-01)	7.52E-17 (9.08E-17)	**3.25E-19 (5.64E-17)**
TF_{18}	2.09E-04 (4.13E-04)	**1.57E-32** (3.22E-32)	**1.57E-32 (0.00)**	2.36E+00 (1.31E+00)	**2.69E-14** (4.50E-14)	**8.13E-20 (2.94E-30)**
TF_{19}	4.13E-04 (4.12E-05)	**1.34E-32** (1.00E-27)	**1.34E-32 (0.00)**	5.49E-01 (2.40E-01)	**1.33E-14** (5.60E-15)	**1.29E-19** (1.35E-25)
TF_{20}	4.14E-04 (2.32E-05)	5.68E-14 (0.00)	**5.68E-14 (0.00)**	1.82E+00 (6.50E-01)	7.95E-14 (2.78E-14)	5.68E-14 (4.28E-14)
TF_{21}	5.64E-01 (4.32E-01)	2.03E-05 (4.02E-05)	**8.43E-06 (8.81E-06)**	1.24E+01 (2.61E+00)	**2.54E-01 (2.15E-02)**	6.41E+00 (1.81E+00)
TF_{22}	2.65E+01 (3.61E+01)	7.41E-02 (6.03E-01)	**5.01E-12 (5.59E-12)**	1.56E+03 (3.18E+02)	3.67E+01 (1.83E+00)	**1.19E+01 (4.40E+01)**
TF_{23}	2.16E+02 (1.29E+01)	1.93E+02 (2.05E+00)	**2.80E+01 (3.23E+00)**	4.52E+02 (4.22E+01)	4.28E+02 (1.26E+01)	**8.91E+01 (2.23E+01)**
TF_{24}	2.69E-01 (2.23E-01)	2.63E-02 (1.30E-02)	**2.84E-14 (0.00)**	8.33E-01 (3.35E-02)	8.87E-03 (5.07E-03)	**2.84E-14 (3.51E-13)**
TF_{25}	6.43E-03 (1.09E-03)	5.68E-14 (0.00)	**4.54E-14 (1.39E-14)**	6.67E-01 (3.24E-01)	**9.66E-09 (1.39E-09)**	6.51E-01 (1.39E-01)

5.2 Performance Evaluation of Proposed Modifications by Convergence Curves

This section displays the performance of CrJaya and MCrJaya through the convergence curves for selected functions. The convergence curves for TF_1, TF_2, TF_5, TF_8, TF_{10}, TF_{11}, TF_{15}, TF_{16}, TF_{23} and TF_{24} are

presented in **Figures 3, 4, 5, 6, 7, 8, 9, 10, 11, 12** respectively. The graphs verify the quick convergence speed of proposed CrJaya and MCrJaya over Jaya algorithm.

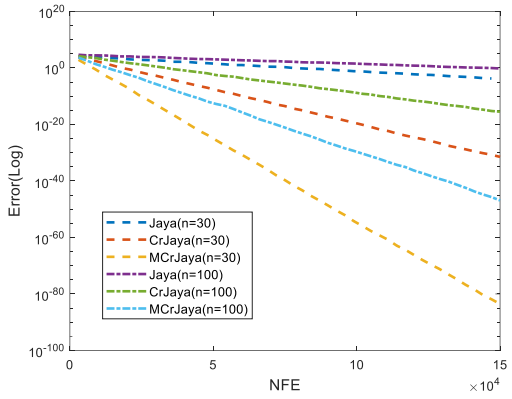

Figure 3. Performance of MCrJaya by convergence curve on TF_1.

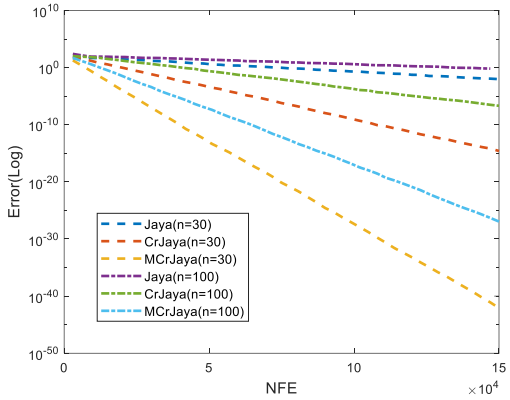

Figure 4. Performance of MCrJaya by convergence curve on TF_2.

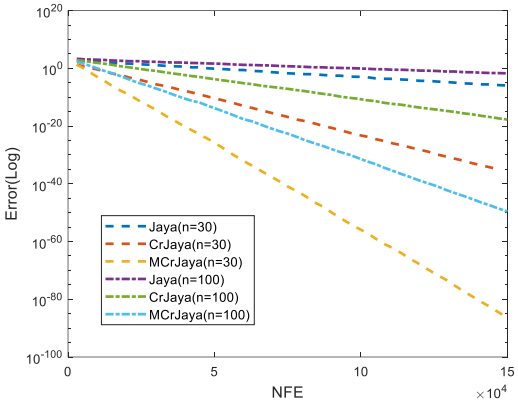

Figure 5. Performance of MCrJaya by convergence curve on TF_5.

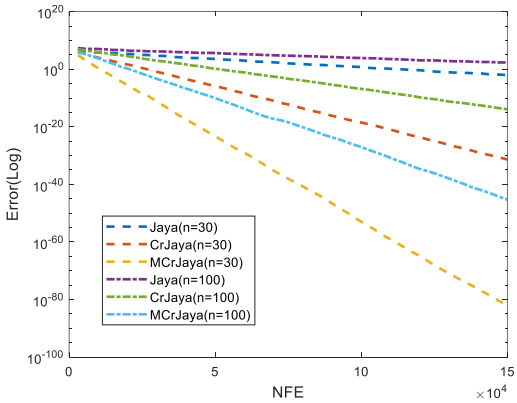

Figure 6. Performance of MCrJaya by convergence curve on TF_8.

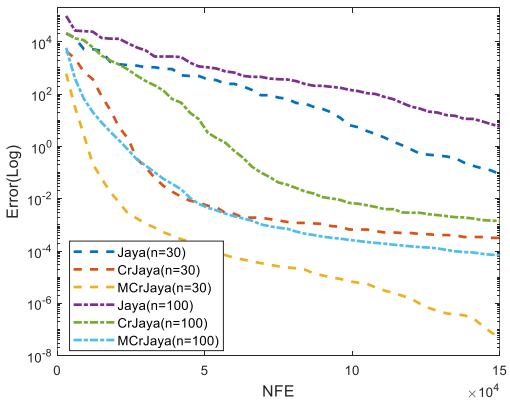

Figure 7. Performance of MCrJaya by convergence curve on TF_{10}.

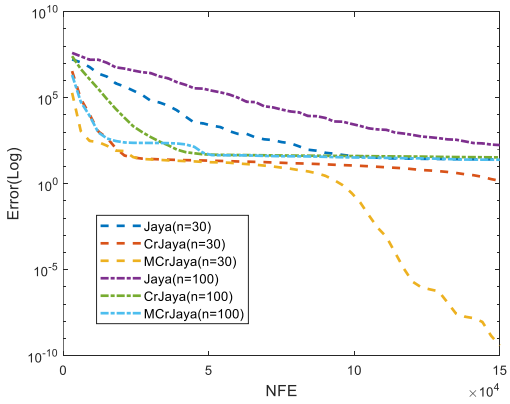

Figure 8. Performance of MCrJaya by convergence curve on TF_{11}.

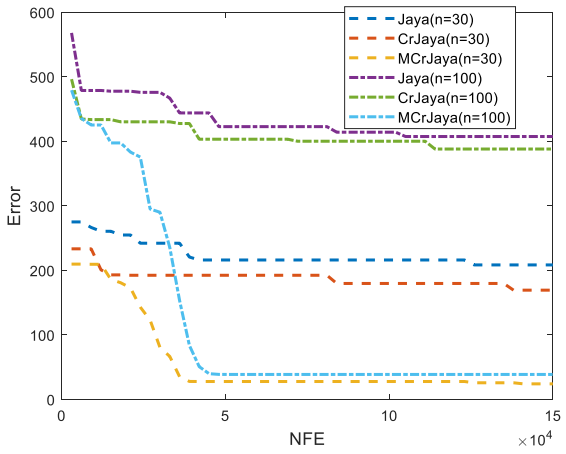

Figure 9. Performance of MCrJaya by convergence curve on TF_{15}.

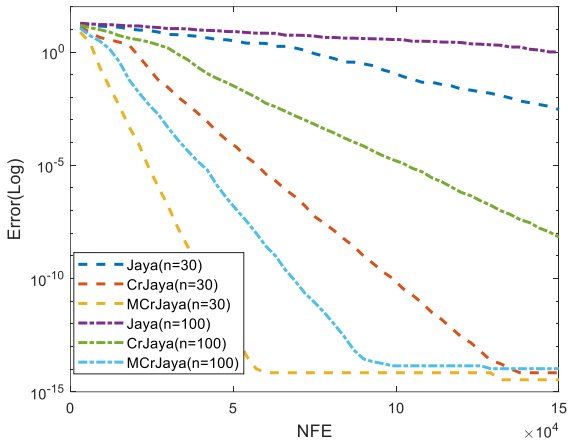

Figure 10. Performance of MCrJaya by convergence curve on TF_{16}.

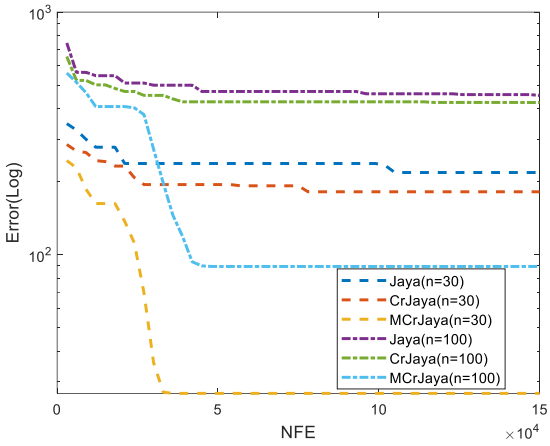

Figure 11. Performance of MCrJaya by convergence curve on TF_{23}.

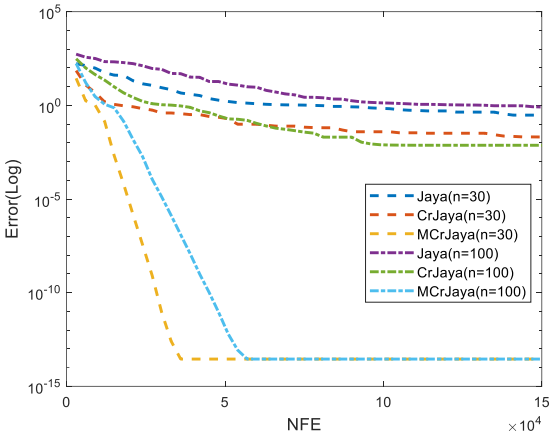

Figure 12. Performance of MCrJaya by Convergence Curve on TF_{24}.

5.3 Comparison with Other Jaya Variants

This section presents the performance assessment of MCrJaya over improved Jaya variants named *EJaya1*, *EJaya* (Zhang et al., 2021) and MRL-JAYA (Kumar & Sharma, 2022). The comparison has performed on 12 selected test functions for n = 40 and n = 60 as given in Zhang et al. (2021). The results for all algorithms are listed in **Table 3** in terms of average error and standard deviation. Here the results for the EJaya1 and EJaya are taken from Zhang et al. (2021). The best outcomes are listed in bold cases.

For n = 40, It can be easily seen that the proposed MCrJaya has obtained the best results for all function and got first rank in all cases. For TF_{10} and TF_{12}, all algorithms are significantly equal whereas MCrJaya and MRL-JAYA have given equal result on TF_{18} and TF_{19}.

A similar performance can be observed for *n* = 60 where proposed MCrJaya has secured the first best place for 11 functions. *EJaya1* has taken the first place for TF_{10}. For TF_{12} all algorithms have performed identical in both cases of *n* = 40 and 60.

A Friedman's rank test based comparison is also given in **Table 3**. The *MCrJaya* has achieved lowest rank for both n = 40 and 60 as 1.21 and 1.46 respectively.

The non-parametric Wilcoxon signed rank sum test has conducted to analyze a pair wise assessment of algorithms. The results are listed in **Table 4**. ΣR^+ denotes the sum of ranks where proposed methods perform better while ΣR^- denotes the reverse case.

The higher positive sum ΣR^+ shows the significant power of the MCrJaya over its competitors. The "+", symbol indicates the superiority of MCrJaya over the other algorithm at 5% level of significance.

Table 3. Comparison of MCrJaya with MRL-Jaya, *EJaya* and *EJaya1*.

	***n* = 40**					***n* = 60**				
Fun	**Gen**	**MCrJaya**	**MRL-Jaya**	**EJaya**	**EJaya1**	**Gen**	**MCrJaya**	**MRL-Jaya**	**EJaya**	**EJaya1**
TF_1	2000	**1.17E-84 (8.54E-84)**	4.34E-66$^+$ (5.61E-66)	2.80E-21$^+$ (6.40E-21)	3.50E-19$^+$ (6.00E-19)	3000	**3.45E-81 (1.09E-50)**	3.51E-59$^+$ (1.11E-60)	1.40E-18$^+$ (2.70E-18)	1.20E-16$^+$ (2.90E-16)
	Rank	**1**	2	3	4	Rank	**1**	2	3	4
TF_2	3000	**2.59E-69 (1.98E-69)**	6.57E-42$^+$ (1.63E-61)	6.70E-17$^+$ (3.74E-16)	3.10E-16$^+$ (2.00E-15)	5000	**5.44E-80 (5.23E-80)**	3.98E-29 $^+$ (3.44E-29)	4.10E-21$^+$ (9.80E-21)	1.20E-07$^+$ (6.00E-07)
	Rank	**1**	2	3	4	Rank	**1**	2	3	4
TF_3	2000	**1.45E-09 (1.11E-09)**	2.16E-07 $^+$ (1.02E-07)	*2.00E+02*$^+$ *(5.68E+01)*	2.90E+02$^+$ (6.90E+01)	3000	**5.92E-05 (3.45E-05)**	*4.82E-02* $^+$ *(8.99E-02)*	*5.75E+02*$^+$ *(1.19E+02)*	6.30E+02$^+$ (8.90E+01)
	Rank	**1**	2	3	4	Rank	**1**	2	3	4
TF_5	2000	**1.47E-86 (1.13E-86)**	1.3E-67 $^+$ (9.12E-68)	1.69E-20 $^+$ (4.50E-20)	4.49E-20 $^+$ (1.35E-19)	3000	**9.78E-84 (7.37E-84)**	9.54E-59 $^+$ (1.32E-59)	6.30E-20 $^+$ (9.30E-20)	1.20E-17 $^+$ (4.20E-17)
	Rank	**1**	2	3	4	Rank	**1**	2	3	4
TF_6	1500	**4.16E-61 (4.53E-61)**	4.05E-47 $^+$ (1.29E-48)	2.00E-12 $^+$ (6.19E-12)	1.41E-12 $^+$ (2.50E-12)	3000	**1.28E-78 (8.25E-78)**	1.40E-56 $^+$ (1.28E-56)	6.30E-18 $^+$ (1.28E-17)	1.49E-16 $^+$ (2.29E-16)
	Rank	**1**	2	4	3	Rank	**1**	2	3	4
TF_7	2000	**1.95E-81 (2.09E-80)**	7.81E-64 $^+$ (3.21E-64)	2.60E-15$^+$ (5.10E-15)	7.59E-15 $^+$ (2.80E-14)	3000	**6.42E-77 (9.20E-78)**	9.01E-57 $^+$ (7.12E-57)	4.90E-13 $^+$ (2.10E-12)	2.00E-11$^+$ (3.50E-11)
	Rank	**1**	2	3	4	Rank	**1**	2	3	4
TF_8	3000	**1.34E-126 (5.80E-126)**	3.02E-84$^+$ (3.63E-84)	7.10E-03 $^+$ (1.20E-02)	7.69E-03 $^+$ (1.25E-02)	3000	**2.33E-78 (1.56E-78)**	5.23E-57 $^+$ (1.02E-56)	2.45E-02$^+$ (3.40E-02)	3.59E-02 $^+$ (1.30E-01)
	Rank	**1**	2	3	4	Rank	**1**	2	3	4
TF_{10}	2000	**3.57E-06 (1.58E-06)**	3.89E-06 $^=$ (1.64E-06)	*8.10E-06*$^=$ *(2.29E-05)*	3.90E-06 $^=$ (1.30E-05)	3000	6.42E-05 (7.25E-06)	5.15E-05 $^-$ (2.33E-05)	*2.40E-06* $^-$ *(9.20E-06)*	**1.29E-06**$^-$ **(2.40E-06)**
	Rank	**1**	2	3	4	Rank	4	3	2	1
TF_{12}	2500	**0.00 (0.00)**	**0.00**$^=$ **(0.00)**	**0.00**$^=$ **(5.69E-01)**	**0.00**$^=$ **(4.20E-01)**	4000	**0.00**$^=$ **(0.00)**	**0.00**$^=$ **(0.00)**	**0.00**$^=$ **(0.00)**	**0.00** $^=$ **(0.00)**
	Rank	**1**	2	3	4	Rank	**1**	2	3	4
TF_{16}	4000	**4.82E-15 (1.74E-15)**	6.96E-15$^+$ (0.00)	2.10E-14$^+$ (4.80E-15)	2.30E-14 $^+$ (5.70E-15)	5000	**9.80E-15 (1.42E-15)**	1.41E-14 $^+$ (2.34E-15)	7.50E-02$^+$ (2.60E-01)	*3.60E-14*$^+$ *(5.30E-15)*
	Rank	**1**	2	3	4	Rank	**1**	2	4	*3*
TF_{18}	2500	**1.01E-19 (3.67E-30)**	**1.01E-19** $^=$ **(3.67E-30)**	9.01E-03 $^+$ (1.45E-02)	1.115E-01$^+$ (4.70E-01)	4000	**6.76E-20 (0.00)**	**6.76E-20** $^=$ **(0.00)**	7.60E-01 $^+$ 92.50E+00)	6.80E-01$^+$ (2.10E+00)
	Rank	**1.5**	**1.5**	3	4	Rank	**1.5**	1.5	4	3
TF_{19}	2500	**1.29E-19 (0.00)**	**1.29E-19** $^=$ **(0.00)**	1.30E+00 $^+$ (1.29E+00)	1.69E+00 $^+$ (1.90E+00)	4000	**1.29E-19 (0.00)**	**1.29E-19**$^=$ **(0.00)**	5.10E+00 $^+$ (3.20E+00)	6.99E+00$^+$ (4.50E+00)
	Rank	**1.5**	**1.5**	3	4	Rank	**1.5**	1.5	3	4
		1.21	1.96	3.13	3.71		**1.46**	2.04	3.04	3.46
w/l/t		--	8/0/4	10/0/2	10/0/2		--	8/1/3	10/1/1	10/1/1
p-value		--	0.004$^+$	<0.001$^+$	<0.001$^+$		--	0.039$^+$	0.012$^+$	0.012$^+$

Table 4. Wilcoxon rank sum test results for MCrJaya, MRL-Jaya, *Ejaya* and *Ejaya1*.

	Σ R⁻	**Σ R⁺**	**z-value**	***p*-value**	**Sig at α = 0.05**
MCrJaya *vs* MRL-Jaya (n = 40)	0	45	2.67	0.003	+
MCrJaya *vs* EJaya (n = 40)	0	66	2.94	0.008	+
MCrJaya *vs* EJaya1 (n = 40)	0	66	2.94	0.008	+
MCrJaya *vs* MRL-Jaya (n = 60)	8	37	1.72	0.046	+
MCrJaya *vs* EJaya (n = 60)	6	60	2.41	0.016	+
MCrJaya *vs* EJaya1 (n = 60)	7	59	2.31	0.021	+

5.4 With Comparison *state-to-art* DE Variants

Table 5. Comparison of MCrJaya with DEGOS, SHADE, APadapSS-JADE, JADE and jDE.

Fun	**Gen**		**MCrJaya**	**DEGOS**	**SHADE**	**APadapSS-JADE**	**JADE**	**jDE**
TF_1	1500	Mean	**3.15E-84**	3.53E-27$^+$	2.50E-75$^-$	1.22E-70$^-$	1.79E-60$^-$	2.59E-28$^+$
		SD	**4.58E-84**	5.42E-27	1.39E-74	4.59E-70	8.39E-60	3.99E-28
		Rank	1	6	2	3	4	5
TF_2	2000	Mean	**7.68E-54**	4.72E-19$^+$	1.79E-44$^-$	4.66E-49$^-$	1.79E-25$^+$	1.92E-23$^+$
		SD	**5.06E-54**	4.82E-19	1.29E-43	4.66E-49	9.10E-25	1.11E-23
		Rank	**1**	6	3	2	4	5
TF_3	5000	Mean	**2.27E-74**	3.82E-22$^+$	2.49E-64$^+$	5.49E-61$^+$	6.01E-61$^+$	4.82E-14$^+$
		SD	**3.08E-74**	9.03E-22	8.40E-64	3.67E-61	2.99E-60	2.29E-13
		Rank	**1**	5	2	3	4	6
TF_4	5000	Mean	**2.25E-71**	5.21E-01+	5.20E-22$^+$	2.09E-41$^+$	8.09E-24$^+$	1.16E-15$^+$
		SD	**1.76E-71**	3.10E-01	4.49E-22	2.29E-41	4.11E-23	2.24E-15
		Rank	**1**	6	4	2	3	5
TF_{11}	3000	Mean	**4.70E-29**	3.19E-22$^+$	3.23E-01$^+$	7.88E-02$^+$	8.12E-02$^+$	1.39E+01$^+$
		SD	**5.22E-30**	3.19E-22	9.99E-01	7.67E-02	(7.21E-01	1.29E+01
		Rank	**1**	2	5	3	4	6
TF_{12}	100	Mean	**1.36E-01**	5.79E+01$^+$	1.00E+00$^=$	2.56E+00 $^+$	2.88E+00 $^+$	1.10E+03$^+$
		SD	**4.23E-02**	1.31E+02	1.89E+00	1.20E+00	-1.11E+00	2.11E+02
		Rank	**1**	5	2	3	4	6
TF_{13}	3000	Mean	**4.22E-04**	2.09E-03$^+$	5.89E-04$^+$	5.08E-04$^+$	6.55E-04$^+$	3.59E-03$^+$
		SD	**3.44E-04**	2.21E-04	1.79E-04	2.31E–04	2.12E-04	8.89E-04
		Rank	**1**	5	3	2	4	6
TF_{14}	1000	Mean	2.51E+03	2.10E+02$^-$	1.79E-08$^-$	1.47E-03$^-$	3.09E-05$^-$	**7.79E-11**$^-$
		SD	3.49E+02	1.01E+02	1.09E-07	1.58E-03	2.12E-05	**2.27E-10**
		Rank	6	5	2	4	3	**1**
TF_{15}	1000	Mean	2.08E+01	1.49E+01$^+$	2.88E-01$^+$	1.69E-02$^+$	**1.10E-04**$^-$	1.89E-04$^-$
		SD	7.99E+00	1.39E+01	-5.61E-01	7.29E-02	**6.15E-05**	3.11E-04
		Rank	6	5	4	3	**1**	2
TF_{16}	500	Mean	**1.16E-13**	5.38E-04$^+$	4.09E-10$^+$	2.58E-10$^+$	8.91E-10$^+$	3.88E-04$^+$
		SD	**2.23E-14**	2.70E-04	-1.82E-11	8.78E-10	6.99E-10	2.11E-04
		Rank	**1**	6	3	2	4	5
TF_{17}	500	Mean	**0.00**	3.09E-05$^+$	**0.00**$^=$	1.67E-14$^+$	9.38E-08$^+$	1.99E-05$^+$
		SD	**0.00**	2.52E-05	**0.00**	9.32E-14	6.12E-07	5.90E-05
		Rank	1	6	**1**	3	4	5
TF_{18}	500	Mean	**3.83E-27**	7.8E-04$^+$	2.18E-22$^+$	3.2E-19$^+$	4.38E-17$^+$	1.98E-07$^+$
		SD	**2.42E-27**	-6.00E-04	7.69E-22	6.52E-19	1.90E-16	1.45E-07
		Rank	**1**	6	2	3	4	5
TF_{19}	500	Mean	**1.97E-26**	5.88E-06$^+$	3.69E-20$^+$	3.81E-18$^+$	2.2E-16$^+$	1.67E-06$^+$
		SD	**4.98E-26**	1.22E-06	1.39E-19	3.42E-18	6.61E-16	9.96E-07
		Rank	**1**	6	2	3	4	5
Avg rank			**1.72**	5.35	2.65	2.78	3.65	4.78
w/l/t				11/2/0	10/2/1	11/2/0	11/2/0	11/2/0
p-value				0.022$^+$	0.039$^+$	0.022$^+$	0.022$^+$	0.022$^+$

This section presents a performance assessment of proposed MCrJaya over some state to art DE variants such as jDE (Brest et al., 2006), JADE (Zhang & Sanderson, 2009), APadapSS-JADE (Gong et al., 2011), SHADE (Tanabe & Fukunaga, 2013) and DEGOS (Yu et al., 2019b). The results are listed in **Table 5** in terms of average accuracy and standard deviation on 13 selected test function for n=30. The results for jDE, JADE, APadapSS-JADE and SHADE are taken from their original research paper, while the results for DEGOS and MCrJaya are obtained on same parameters value.

From the **Table 5**, it can be easily examined that MCrJaya has completed the task with more accuracy and got the first position on 11 functions except TF_{14}, and TF_{15}, where jDE and JADE have obtained the first position respectively. For TF_{17}, both MCrJaya and SHADE have achieved the best solution. The *win/loss/tie* performance shows that MCrJaya is achieved better performance over DEGOS, SHADE, APadapSS-JADE, JADE and jDE on 11, 10, 11, 11 and 11 functions respectively.

The Friedman's ranks for MCrJaya, DEGOS, SHADE, APadapSS-JADE, JADE and jDE are 1.72, 5.35, 2.56, 2.68, 3.65 and 4.78. Thus, smaller rank of MCrJaya proves its supremacy on its competitors.

The Wilcoxon sign rank test results are listed in **Table 6**. The higher positive sum ΣR^+ shows the significant power of the MCrJaya over its competitors. However, the *p-value* indicates that all algorithms are also significantly equal with MCrJaya at 5% level of significance.

Table 6. Wilcoxon signed rank sum test results for MCrJaya, DEGOS, SHADE, ApAdap-JADE, JADE and jDE**.**

	ΣR^-	**ΣR^+**	**z-value**	***p*-value**	**Sig at α=0.05**
MCrJaya *vs* DEGOS	24	67	1.503	0.133	=
MCrJaya *vs* SHADE	23	55	1.255	0.209	=
MCrJaya *vs* ApAdapSS-JADE	25	66	1.433	0.152	=
MCrJaya *vs* JADE	25	66	1.433	0.152	=
MCrJaya *vs* jDE	24	67	1.503	0.133	=

5.5 Comparison on CEC2017 Functions

This section presents the performance assessmant of MCrJaya with 07 seven metahueristics SHADE (Tanabe & Fukunaga, 2013), DEGOS (Yu et al., 2019b), TRADE (Cai et al., 2023), HGSA (Wang et al., 2019), TDSD (Li et al., 2020), AGBSO (Peng et al., 2021) and HMRFO (Tang et al., 2023) on CEC2017 function. The results for SHADE, HGSA and TDSD are taken from Li et al. (2020). The results for the TRADE, DEGOS, HMRFO and AGBSO, are acquired by using the code provided by the authors on *https://toyamaailab.github.io/sourcedata.html*. The population size and iterations are taken as 100 and 3000 respectively.

The results are listed in **Table 7** in terms of average accuracy and standard deviation of 50 independent runs. **Table 7** shows that MCrJaya, TRADE, DEGOS, SHADE, HMRFO, AGBSO, HGSA and TDSD has obtained best results on 08, 12, 06, 05, 05, 03 and 04 cases respectively. The *win/loss/tie* performance indicates that MCrJaya exceed the algorithm TRADE, DEGOS, SHADE, HMRFO, AGBSO, HGSA and TDSD on 13, 18, 14, 28, 18, 25 and 22 functions respectively.

The Wilcoxon rank sum test presented in **Table 8**. The higher value of ΣR^+ shows the supremacy of MCrJaya on its competitors. The *p*-values indicates that the MCrJaya is significant better its all competitors except TRADE and SHADE which have given significantly equal performance with MCrJaya.

The convergence behaviour of MCrJaya can be analyzed by the convergence graphs of selected functions C_3, C_4, C_{12} and C_{30} which are presented in **Figures 13, 14, 15** and **16** respectively. The graphs also verify that the proposed MCrJaya achieve the accuracy quickly.

Table 7. Performance evaluation of MCrJaya on CEC-2017 functions.

Fun		MCrJaya	TRADE	DEGOS	SHADE	HMRFO	AGBSO	HGSA	TDSD
C_1	Mean	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	2.98E+03	2.28E+03	2.58E+03	1.75E+03
	SD	0.00E+00	0.00E+00	0.00E+00	0.00E+00	2.34E+03	2.50E+03	2.49E+03	9.05E+02
	Rank	**1**	**1**	**1**	**1**	8	6	7	5
C_3	Mean	**0.00E+00**	1.22E+01	2.79E-05	**0.00E+00**	6.14E+01	1.84E+00	4.33E+04	4.04E+04
	SD	0.00E+00	1.01E+01	6.55E-05	0.00E+00	3.57E+01	1.02E+02	5.48E+03	9.55E+03
	Rank	**1**	5	3	**1**	6	4	8	7
C_4	Mean	5.27E+01	6.12E+01	5.91E+01	5.66E+01	3.64E+01	8.97E+01	1.20E+02	**2.02E+01**
	SD	1.85E+01	2.90E+01	1.76E+00	1.07E+01	3.57E+01	1.48E+01	2.70E+00	2.01E+01
	Rank	3	6	5	4	2	7	8	1
C_5	Mean	3.65E+01	3.88E+01	3.90E+01	**1.46E+01**	6.00E+01	1.62E+01	1.53E+02	7.99E+01
	SD	1.64E+01	7.11E+00	1.22E+01	2.36E+00	3.50E+01	5.93E+00	1.28E+01	1.16E+01
	Rank	3	4	5	**1**	6	2	8	7
C_6	Mean	**0.00E+00**	**0.00E+00**	8.28E-07	3.81E-05	4.86E-01	7.08E-05	8.21E+00	3.30E+00
	SD	2.00E-06	0.00E+00	1.40E-06	4.69E-05	1.91E+01	4.81E-05	4.55E+00	6.78E-01
	Rank	**1**	**1**	3	4	6	5	8	7
C_7	Mean	6.18E+01	5.10E+01	7.36E+01	4.41E+01	1.17E+02	5.06E+01	**4.08E+01**	1.32E+02
	SD	1.47E+01	5.42E+00	4.88E+01	2.40E+00	1.07E+00	7.13E+00	3.00E+00	1.38E+01
	Rank	5	4	6	2	7	3	**1**	8
C_8	Mean	2.89E+01	3.18E+01	3.58E+01	1.54E+01	6.53E+01	**1.42E+01**	9.98E+01	8.33E+01
	SD	1.21E+01	6.28E+00	1.44E+01	2.93E+00	1.89E+01	4.92E+00	9.04E+00	8.20E+00
	Rank	3	4	5	2	6	**1**	8	7
C_9	Mean	**0.00E+00**	**0.00E+00**	**0.00E+00**	5.42E+02	4.64E+01	**0.00E+00**	**0.00E+00**	1.71E+03
	SD	0.00E+00	0.00E+00	0.00E+00	1.23E-01	4.07E+01	0.00E+00	0.00E+00	3.48E+02
	Rank	**1**	**1**	**1**	7	6	**1**	**1**	8
C_{10}	Mean	2.93E+03	7.46E+03	4.28E+03	1.70E+03	3.47E+03	**4.56E+02**	3.20E+03	2.19E+03
	SD	5.52E+02	2.32E+02	2.25E+03	1.91E+02	6.76E+02	2.66E+02	2.91E+02	2.26E+02
	Rank	4	8	7	2	6	**1**	5	3
C_{11}	Mean	3.31E+01	**7.81E+00**	1.12E+01	4.32E+01	4.29E+01	4.82E+01	9.73E+01	6.86E+01
	SD	2.63E+01	4.18E+00	2.53E+00	3.25E+01	1.05E+01	2.77E+01	2.98E+01	2.39E+02
	Rank	3	**1**	2	5	4	6	8	7
C_{12}	Mean	**1.25E+03**	1.08E+04	7.84E+03	1.37E+03	3.65E+04	6.72E+05	1.27E+05	2.91E+05
	SD	1.18E+04	8.02E+03	9.39E+03	3.86E+02	1.35E+04	3.41E+05	8.14E+04	1.73E+05
	Rank	**1**	4	3	2	5	8	6	7
C_{13}	Mean	2.55E+01	2.88E+01	**2.36E+01**	3.84E+01	1.48E+04	1.27E+04	1.34E+04	6.95E+02
	SD	1.21E+01	3.30E+00	1.07E+01	1.18E+01	1.02E+04	8.89E+03	5.32E+03	3.29E+02
	Rank	2	3	**1**	4	8	6	7	5
C_{14}	Mean	2.83E+01	2.94E+01	**2.13E+01**	2.93E+01	1.68E+03	2.93E+03	5.32E+03	8.42E+03
	SD	1.14E+01	3.32E+00	1.02E+01	4.94E+00	9.51E+02	2.86E+03	3.04E+03	6.08E+03
	Rank	2	4	**1**	3	5	6	7	8
C_{15}	Mean	1.40E+01	8.15E+00	**7.85E+00**	2.38E+01	2.87E+03	3.36E+03	6.99E+02	3.56E+02
	SD	1.37E+01	2.59E+00	3.61E+00	2.37E+01	3.81E+03	3.60E+03	7.21E+02	2.25E+02
	Rank	3	2	**1**	4	7	8	6	5
C_{16}	Mean	6.26E+02	**3.69E+01**	4.74E+02	2.88E+02	6.30E+02	1.54E+02	1.23E+03	4.85E+02
	SD	3.17E+02	6.96E+01	4.83E+02	1.41E+02	2.92E+02	9.34E+01	2.31E+02	1.14E+02
	Rank	6	**1**	4	3	7	2	8	5
C_{17}	Mean	5.68E+01	**2.49E+01**	9.47E+01	6.14E+01	1.95E+02	7.02E+01	1.07E+03	9.38E+01
	SD	1.02E+02	5.17E+00	7.00E+01	3.79E+01	1.36E+02	3.64E+01	1.98E+02	3.99E+01
	Rank	2	**1**	6	3	7	4	8	5
C_{18}	Mean	1.11E+02	**2.57E+01**	3.21E+01	6.10E+01	8.15E+04	9.37E+04	5.98E+04	8.14E+04
	SD	1.53E+02	9.53E+00	1.50E+01	2.97E+01	3.38E+04	5.16E+04	1.48E+04	3.68E+04
	Rank	4	**1**	2	3	7	8	5	6
C_{19}	Mean	2.85E+01	**5.15E+00**	7.14E+00	1.39E+01	2.52E+03	5.16E+03	3.52E+03	1.53E+02
	SD	3.50E+01	1.15E+00	2.99E+00	7.56E+00	2.66E+03	6.31E+03	1.25E+03	1.02E+02
	Rank	4	**1**	2	3	6	8	7	5

Table 7 continued…

C_{20}	Mean	2.85E+01	**1.86E+01**	6.51E+01	8.63E+01	2.70E+02	1.06E+02	8.56E+02	1.38E+02
	SD	2.00E+02	9.12E+00	9.37E+01	5.82E+01	1.24E+02	6.36E+01	2.23E+02	5.42E+01
	Rank	2	**1**	3	4	7	5	8	6
C_{21}	Mean	2.24E+02	2.26E+02	2.31E+02	**2.17E+02**	2.52E+02	2.29E+02	3.10E+02	2.22E+02
	SD	1.86E+01	6.01E+00	9.33E+00	2.51E+00	1.84E+01	5.65E+00	5.89E+00	8.16E+01
	Rank	3	4	6	**1**	7	5	8	2
C_{22}	Mean	**1.00E+02**	**1.00E+02**	**1.00E+02**	**1.00E+02**	**1.00E+02**	**1.00E+02**	**1.00E+02**	1.11E+02
	SD	0.00E+00	0.00E+00	0.00E+00	0.00E+00	2.40E-13	2.20E-06	3.89E-09	1.90E+00
	Rank	**1**	**1**	**1**	**1**	**1**	**1**	**1**	8
C_{23}	Mean	3.76E+02	3.82E+02	3.75E+02	3.66E+02	4.30E+02	**3.61E+02**	4.57E+02	4.54E+02
	SD	2.31E+01	7.11E+00	1.37E+01	5.25E+00	2.51E+01	6.07E+00	1.34E+02	1.73E+02
	Rank	4	5	3	2	6	**1**	8	7
C_{24}	Mean	4.51E+02	4.65E+02	4.57E+02	4.37E+02	4.81E+02	4.36E+02	5.23E+02	**4.25E+02**
	SD	1.37E+01	9.24E+00	1.82E+01	4.14E+00	1.71E+01	1.04E+01	3.58E+01	1.88E+02
	Rank	4	6	5	3	7	2	8	**1**
C_{25}	Mean	3.87E+02	3.87E+02	4.53E+02	3.87E+02	3.92E+02	3.87E+02	3.90E+02	**3.83E+02**
	SD	1.36E-01	2.32E-02	1.82E+01	7.80E+02	1.37E+01	1.05E+00	7.60E+00	1.14E+01
	Rank	2	2	8	2	7	2	6	**1**
C_{26}	Mean	1.33E+03	9.49E+02	1.43E+03	1.10E+03	1.68E+03	9.87E+02	2.54E+02	**2.21E+02**
	SD	1.10E+02	1.16E+02	1.36E+02	5.44E+00	8.49E+02	7.46E+01	5.08E+01	5.55E+00
	Rank	6	3	7	5	8	4	2	**1**
C_{27}	Mean	4.85E+02	4.93E+02	5.02E+02	5.06E+02	5.45E+02	5.05E+02	5.52E+02	5.21E+02
	SD	1.27E+01	1.06E+01	8.08E+00	6.34E+00	1.58E+01	6.09E+00	2.07E+01	6.14E+00
	Rank	1	2	3	5	7	4	8	6
C_{28}	Mean	**300E+02**	**3.00E+02**	3.54E+02	3.37E+02	3.34E+02	3.83E+02	3.08E+02	4.09E+02
	SD	7.98E+01	4.28E-05	7.16E+01	5.38E+01	5.73E+01	4.01E+01	2.82E+01	1.49E+01
	Rank	**1**	**1**	6	5	4	7	3	8
C_{29}	Mean	4.33E+02	**4.15E+02**	4.63E+02	4.63E+02	7.66E+02	4.81E+02	1.15E+03	5.90E+02
	SD	1.01E+02	7.13E+00	7.69E+01	4.23E+01	1.78E+02	5.65E+01	1.87E+02	4.57E+01
	Rank	2	**1**	4	3	7	5	8	6
C_{30}	Mean	**1.98E+03**	2.14E+03	2.11E+03	2.10E+03	3.91E+03	5.26E+04	7.99E+03	5.04E+03
	SD	1.43E+02	5.30E+01	1.01E+02	1.52E+02	1.15E+03	4.00E+04	2.61E+03	8.30E+02
	Rank	**1**	4	3	2	5	8	7	6
Avg. Rank		**2.97**	3.16	3.93	3.22	6.03	4.72	6.55	5.41
w/l/t		--	13/10/6	18/8/3	14/11/4	28/1/0	18/8/3	25/2/2	22/7/0
p-value		--	$0.678^{=}$	$0.078^{=}$	$0.690^{=}$	0.001^{+}	$0.078^{=}$	0.001^{+}	0.009^{+}

Table 8. Wilcoxon signed rank sum test results for MCrJaya, TRADE, DEGOS, SHADE, HMRFO, AGBSO, HGSA and TDSD**.**

	Σ R^-	**Σ R^+**	**z-value**	***p*-value**	**Sig at α=0.05**
MCrJaya *vs* TRADE	148	128	0.304	0.761	=
MCrJaya *vs* DEGOS	98	252	1.956	0.049	+
MCrJaya *vs* SHADE	161	164	0.040	0.968	=
MCrJaya *vs* HMRFO	15	391	4.281	0.001	+
MCrJaya *vs* AGBSO	88	263	2.222	0.026	+
MCrJaya *vs* HGSA	25	381	4.053	0.001	+
MCrJaya *vs* TDSD	75	360	3.081	0.002	+

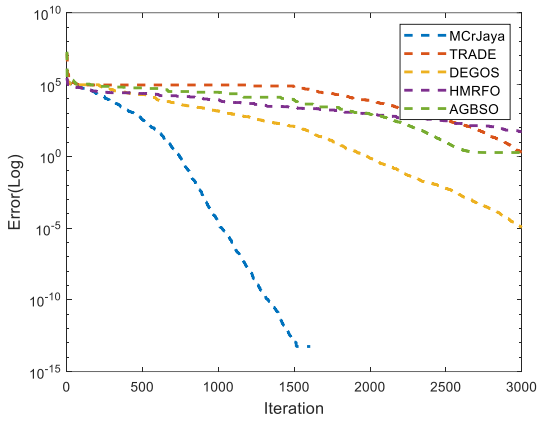

Figure 13. Performance of MCrJaya by convergence curve on C_3 (Unimodel).

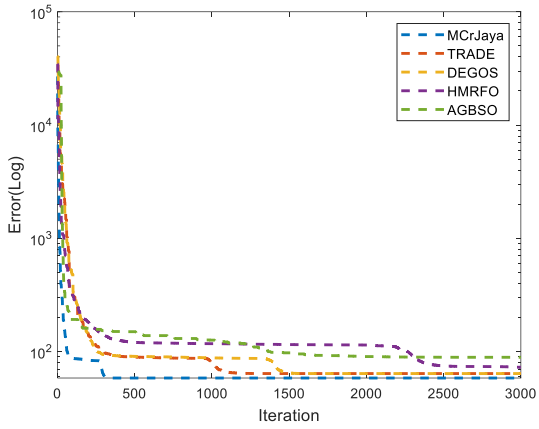

Figure 14. Performance of MCrJaya by convergence curve on C_4 (Multimodel).

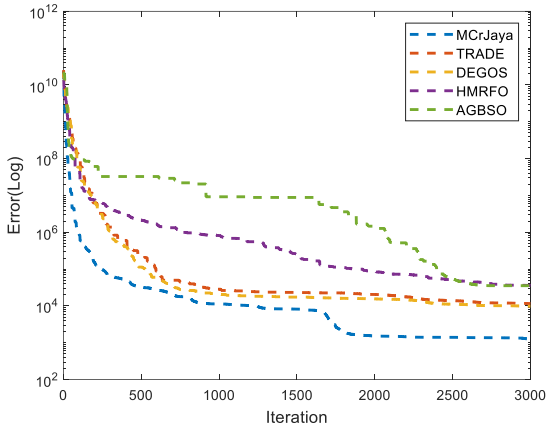

Figure 15. Performance of MCrJaya by convergence curve on C_{12} (Hybrid).

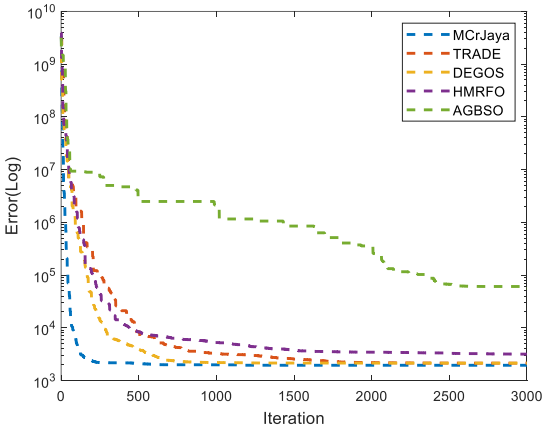

Figure 16. Performance of MCrJaya by convergence curve on C_{30} (Composite).

5.6 Performance Assessment on Real Life Applications

To check the practicability of the proposed MCrJaya, following three real life applications are taken from CEC2011 test suit (Das & Suganthan, 2011):

RF_1: *Frequency-Modulated (FM) Sound Waves Application.*
RF_2: *Spread Spectrum Radar Polly phase Code Design Application.*
RF_3: *Non-Linear Stirred Tank Reactor optimal Control Application.*

The more specifications and mathematical models can be found in Das & Suganthan (2011). Five comparative algorithms including Jaya, MRL-Jaya, DEGOS, SHADE and TDSD are considered for the comparison. The outcomes for the SHADE and TDSD are used from Li et al. (2020). The maximum iterations for RF_1, RF_2 and RF_3 are taken as 600, 2000 and 100 respectively. The results are obtained in term of best, average and SD of 50 independent runs and are listed in **Table 9**. Results shows that MCrJaya has achieved best value for all three applications and hence proves its practicability on real life applications also.

The convergence graphs for MCrJaya, Jaya, MRL-Jaya and DEGOS are presented in **Figures 17, 18** and **19** which also confirms the convergence rapidity of the MCrJaya on its competitors.

Table 9. Solution and comparison for real life applications.

Application	Value	MCrJaya	Jaya	MRL-Jaya	DEGOS	SHADE	TDSD
RF_1	Best	**0.00**	17.94	**0.00**	**0.00**	**0.00**	**0.00**
	Mean	**0.704**	20.53	9.82	6.89	1.82	3.93
	SD	**0.097**	2.25	6.86	6.50	2.60	4.97
	rank	**1**	6	5	4	2	3
RF_2	Best	**0.5001**	1.5014	0.7138	0.8203	1.0346	1.1560
	Mean	**0.5764**	1.6626	0.8935	1.3674	1.2256	1.0236
	SD	**0.0712**	0.1248	0.0789	0.5885	0.0975	0.0775
	rank	**1**	6	2	5	4	3
RF_3	Best	**13.77**	14.46	13.77	14.32	13.85	**13.77**
	Mean	**13.81**	15.96	14.35	18.74	14.28	13.93
	SD	**0.192**	2.120	0.394	3.821	0.20	0.17
	rank	**1**	5	4	6	3	2

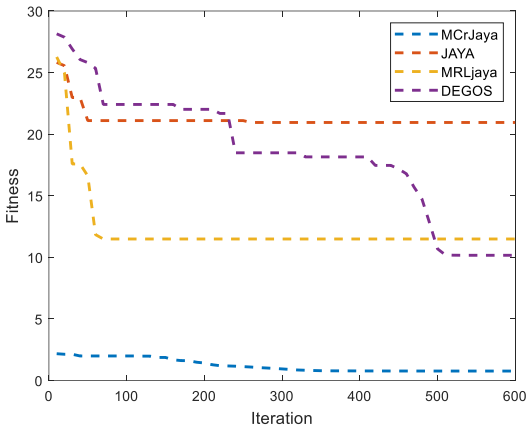

Figure 17. Performance of MCrJaya by convergence curve on real life application RF_1.

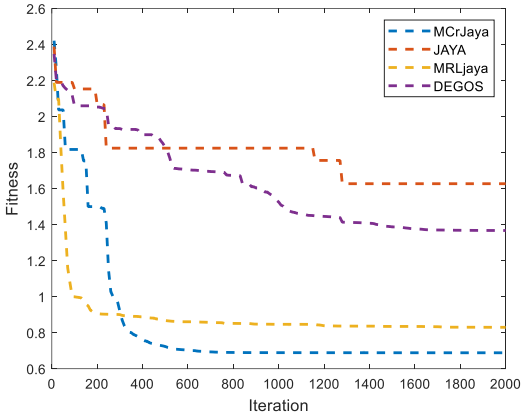

Figure 18. Performance of MCrJaya by convergence curves on real life application RF_2.

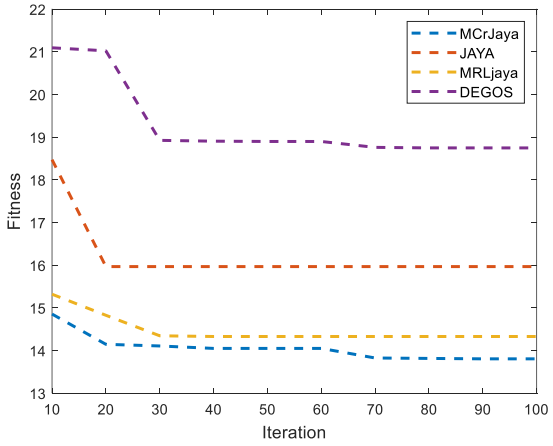

Figure 19. Performance of MCrJaya by convergence curve on real life application RF_3.

6. Conclusion and Future Scope

The Jaya algorithm is a recently designed metaheuristics and has been verified for its good performance in solving various engineering and scientific optimization applications. In this paper, a novel modified Jaya algorithm variant named *MCrJaya* is presented. The proposed variant improves the Jaya performance by including the *DE/current-to-best* and *DE/*α_{best}*/1* approach based crossover and selection operation respectively. The newly designed crossover operation manages the population diversity, while the second approach helps to amplify the convergence rapidity of the algorithm. The performance of proposed variant has verified and compared with several improved varaints of Jaya and DE as well as other metahueristics on 54 benchmark functions and 03 real life applications. The experimental outcomes prove that the exploration ability of Jaya has improved by including the newly designed crossover operation whereas including of *DE/*α_{best}*/1* mutation approach based selection make it more robust and effective in term of convergence rapidity and solution quality.

In the future, first we are planning to extend this work to attempt constrained and multi objective optimization problems. Second, it will also be interesting to implement our proposed modifications on other meta-heuristics algorithms.

Conflict of Interest

The authors confirm that there is no conflict of interest to declare for this publication.

AI Disclosure

The author(s) declare that no assistance is taken from generative AI to write this article.

Acknowledgments

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors. The most important thing is that I want to thank my baby boy, Advik Kharb. Your birth has given me more courage and strength. I love you forever. The authors would like to thank the editor and anonymous reviewers for their comments that help improve the quality of this work.

References

Abualigah, L., Elaziz, M.A., Sumari, P., Geem, Z.W., & Gandomi, A.H. (2022). Reptile search algorithm (RSA): a nature-inspired meta-heuristic optimizer. *Expert Systems with Applications, 191*, 116158.

Askari, Q., Younas, I., & Saeed, M. (2020). Political optimizer: a novel socio-inspired meta-heuristic for global optimization. *Knowledge-Based Systems, 195*, 105709. https://doi.org/10.1016/j.knosys.2020.105709.

Awad, N.H., Ali, M.Z., Liang, J.J., Qu, B.Y., & Suganthan, P.N. (2016). Problem definitions and evaluation criteria for the CEC 2017 special session and competition on single objective bound constrained real-parameter numerical optimization. In *Technical Report* (pp. 1-34). Nanyang Technological University, Singapore.

Bilal, Pant, M., Zaheer, H., Garcia-Hernandez, L., & Abraham, A. (2020). Differential evolution: a review of more than two decades of research. *Engineering Applications of Artificial Intelligence, 90*, 103479. https://doi.org/10.1016/j.engappai.2020.103479.

Brest, J., Greiner, S., Bošković, B., Mernik, M., & Zumer, V. (2006). Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation, 10*(6), 646-657. https://doi.org/10.1109/tevc.2006.872133.

Brest, J., Maučec, M.S., & Bošković, B. (2016). IL-SHADE: Improved L-SHADE algorithm for single objective real-parameter optimization. In *2016 IEEE Congress on Evolutionary Computation* (pp. 1188-1195). Vancouver, BC, Canada. https://doi.org/10.1109/cec.2016.7743922.

Cai, Z., Yang, X., Zhou, M.C., Zhan, Z.H., & Gao, S. (2023). Toward explicit control between exploration and exploitation in evolutionary algorithms: a case study of differential evolution. *Information Sciences*, *649*, 119656. https://doi.org/10.1016/j.ins.2023.119656.

Chopra, N., & Ansari, M. (2022). Golden jackal optimization: a novel nature-inspired optimizer for engineering applications. *Expert Systems with Applications*, *198*, 116924. https://doi.org/10.1016/j.eswa.2022.116924.

Cui, L., Li, G., Lin, Q., Chen, J., & Lu, N. (2016). Adaptive differential evolution algorithm with novel mutation strategies in multiple sub-populations. *Computers and Operations Research*, *67*, 155-173. https://doi.org/10.1016/j.cor.2015.09.006.

Das, S., & Suganthan, P.N. (2011). Problem definitions and evaluation criteria for CEC 2011 competition on testing evolutionary algorithms on real world optimization problems. Technical Report, pp.341-359).

Dehghani, M., & Trojovský, P. (2021). Teamwork optimization algorithm: a new optimization approach for function minimization/maximization. *Sensors*, *21*(13), 4567. https://doi.org/10.3390/s21134567.

Deng, W., Shang, S., Cai, X., Zhao, H., Song, Y., & Xu, J. (2021). An improved differential evolution algorithm and its application in optimization problem. *Soft Computing*, *25*(7), 5277-5298. https://doi.org/10.1007/s00500-020-05527-x.

Di Carlo, M., Vasile, M., & Minisci, E. (2020). Adaptive multi-population inflationary differential evolution. *Soft Computing*, *24*(5), 3861-3891. https://doi.org/10.1007/s00500-019-04154-5.

Draa, A., Chettah, K., & Talbi, H. (2019). A compound sinusoidal differential evolution algorithm for continuous optimization. *Swarm and Evolutionary Computation*, *50*, 100450. https://doi.org/10.1016/j.swevo.2018.10.001.

Gao, K., Zhang, Y., Sadollah, A., Lentzakis, A., & Su, R. (2017). Jaya, harmony search and water cycle algorithms for solving large-scale real-life urban traffic light scheduling problem. *Swarm and Evolutionary Computation*, *37*, 58-72. https://doi.org/10.1016/j.swevo.2017.05.002.

Gong, C. (2017). An enhanced jaya algorithm with a two group adaption. *International Journal of Computational Intelligence Systems*, *10*(1), 1102-1115. https://doi.org/10.2991/ijcis.2017.10.1.73.

Gong, W., Fialho, Á., Cai, Z., & Li, H. (2011). Adaptive strategy selection in differential evolution for numerical optimization: an empirical study. *Information Sciences*, *181*(24), 5364-5386. https://doi.org/10.1016/j.ins.2011.07.049.

Hashim, F.A., Hussain, K., Houssein, E.H., Mabrouk, M.S., & Al-Atabany, W. (2021). Archimedes optimization algorithm: a new metaheuristic algorithm for solving optimization problems. *Applied Intelligence*, *51*(3), 1531-1551. https://doi.org/10.1007/s10489-020-01893-z.

Jia, H., Peng, X., & Lang, C. (2021). Remora optimization algorithm. *Expert Systems with Applications*, *185*, 115665. https://doi.org/10.1016/j.eswa.2021.115665.

Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization*, *39*(3), 459-471. https://doi.org/10.1007/s10898-007-9149-x.

Kaur, S., Awasthi, L.K., Sangal, A.L., & Dhiman, G. (2020). Tunicate swarm algorithm: a new bio-inspired based metaheuristic paradigm for global optimization. *Engineering Applications of Artificial Intelligence*, *90*, 103541. https://doi.org/10.1016/j.engappai.2020.103541.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks* (Vol. 4, pp. 1942-1948). IEEE. Perth, WA, Australia. https://doi.org/10.1109/icnn.1995.488968.

Kiran, M.S. (2015). TSA: tree-seed algorithm for continuous optimization. *Expert Systems with Applications*, *42*(19), 6686-6698. https://doi.org/10.1016/j.eswa.2015.04.055.

Kumar, P., & Pant, M. (2012). Enhanced mutation strategy for differential evolution. In *2012 IEEE Congress on Evolutionary Computation* (pp. 1-60). IEEE. Brisbane, QLD, Australia. https://doi.org/10.1109/cec.2012.6252914.

Kumar, P., & Sharma, A. (2022). MRL-JAYA: a fusion of MRLDE and JAYA algorithm. *Palestine Journal of Mathematics, 11*(1), 65-74.

Kurada, R.R., & Kanadam, K.P. (2016). Automatic unsupervised data classification using jaya evolutionary algorithm. *Advanced Computational Intelligence: An International Journal, 3*(2), 35-42. https://doi.org/10.5121/acii.2016.3204.

Li, X., Cai, Z., Wang, Y., Todo, Y., Cheng, J., & Gao, S. (2020). TDSD: A new evolutionary algorithm based on triple distinct search dynamics. *IEEE Access, 8*, 76752-76764. https://doi.org/10.1109/access.2020.2989029.

Li, X., Wang, K., Yang, H., Tao, S., Feng, S., & Gao, S. (2022). PAIDDE: a permutation-archive information directed differential evolution algorithm. *IEEE Access, 10*, 50384-50402. https://doi.org/10.1109/access.2022.3173622.

Meng, Z., Pan, J.S., & Tseng, K.K. (2019). PaDE: an enhanced differential evolution algorithm with novel control parameter adaptation schemes for numerical optimization. *Knowledge-Based Systems, 168*, 80-99. https://doi.org/10.1016/j.knosys.2019.01.006.

Meng, Z., & Yang, C. (2022). Two-stage differential evolution with novel parameter control. *Information Sciences, 596*, 321-342. https://doi.org/10.1016/j.ins.2022.03.043.

Meng, Z., Zhong, Y., & Yang, C. (2021). CS-DE: cooperative strategy based differential evolution with population diversity enhancement. *Information Sciences, 577*, 663-696. https://doi.org/10.1016/j.ins.2021.07.080.

Mirjalili, S. (2016). SCA: a sine cosine algorithm for solving optimization problems. *Knowledge-Based Systems, 96*, 120-133. https://doi.org/10.1016/j.knosys.2015.12.022.

Mirjalili, S., & Lewis, A. (2016). The whale optimization algorithm. *Advances in Engineering Software, 95*, 51-67. https://doi.org/10.1016/j.advengsoft.2016.01.008.

Mirjalili, S., Mirjalili, S.M., & Lewis, A. (2014). Grey wolf optimizer. *Advances in Engineering Software, 69*, 46-61. https://doi.org/10.1016/j.advengsoft.2013.12.007.

Mishra, S., & Ray, P.K. (2016). Power quality improvement using photovoltaic fed DSTATCOM based on JAYA optimization. *IEEE Transactions on Sustainable Energy, 7*(4), 1672-1680. https://doi.org/10.1109/tste.2016.2570256.

Mohamed, A.W., Hadi, A.A., & Mohamed, A.K. (2020). Gaining-sharing knowledge-based algorithm for solving optimization problems: a novel nature-inspired algorithm. *International Journal of Machine Learning and Cybernetics, 11*(7), 1501-1529. https://doi.org/10.1007/s13042-019-01053-x.

Peng, Y., Shen, Z., & Wang, S. (2021). Adaptive grouping brain storm optimization for multimodal optimization problems. *International Journal of Swarm Intelligence Research, 12*(4), 81-100. https://doi.org/10.4018/ijsir.2021100105.

Poikolainen, I., Neri, F., & Caraffini, F. (2015). Cluster-based population initialization for differential evolution frameworks. *Information Sciences, 297*, 216-235. https://doi.org/10.1016/j.ins.2014.11.026.

Połap, D., & Woźniak, M. (2021). Red fox optimization algorithm. *Expert Systems with Applications, 166*, 114107. https://doi.org/10.1016/j.eswa.2020.114107.

Qin, A.K., Huang, V.L., & Suganthan, P.N. (2009). Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation, 13*(2), 398-417. https://doi.org/10.1109/tevc.2008.927706.

Rao, R.V., & Keesari, H.S. (2018). Multi-team perturbation guiding Jaya algorithm for optimization of wind farm layout. *Applied Soft Computinh, 71*, 800-815. https://doi.org/10.1016/j.asoc.2018.07.036.

Rao, R.V., Rai, D.P., & Balic, J. (2016a). Surface grinding process optimization using Jaya algorithm. In: Behera, H., Mohapatra, D. (eds) *Computational Intelligence in Data Mining-Volume 2*. Springer, New Delhi, pp. 487-495. https://doi.org/10.1007/978-81-322-2731-1_46.

Rao, R.V., & Saroj, A. (2017). Economic optimization of shell-and-tube heat exchanger using Jaya algorithm with maintenance consideration. *Applied Thermal Engineering*, *116*, 473-487. https://doi.org/10.1016/j.applthermaleng.2017.01.071.

Rao, R.V., & Waghmare, G.G. (2017). A new optimization algorithm for solving complex constrained design optimization problems. *Engineering Optimization*, *49*(1), 60-83. https://doi.org/10.1080/0305215x.2016.1164855.

Rao, R.V., More, K.C., Taler, J., & Ocłoń, P. (2016b). Dimensional optimization of a micro-channel heat sink using Jaya algorithm. *Applied Thermal Engineering*, *103*, 572-582. https://doi.org/10.1016/j.applthermaleng.2016.04.135.

Rao, R.V., Savsani, V.J., & Vakharia, D.P. (2011). Teaching-learning-based optimization: a novel method for constrained mechanical design optimization problems. *Computer Aided Design*, *43*(3), 303-315. https://doi.org/10.1016/j.cad.2010.12.015.

Rashedi, E., Nezamabadi-pour, H., & Saryazdi, S. (2009). GSA: a gravitational search algorithm. *Information Sciences, 179*(13), 2232-2248. https://doi.org/10.1016/j.ins.2009.03.004.

Sallam, K.M., Elsayed, S.M., Chakrabortty, R.K., & Ryan, M.J. (2020). Improved multi-operator differential evolution algorithm for solving unconstrained problems. In *2020 IEEE Congress on Evolutionary Computation* (pp. 1-8). IEEE. Glasgow, UK. https://doi.org/10.1109/cec48606.2020.9185577.

Shi, Y. (2011). Brain storm optimization algorithm. In: Tan, Y., Shi, Y., Chai, Y., Wang, G. (eds) *Advances in Swarm Intelligence*. Springer, Berlin, Heidelberg, pp. 303-309. ISBN: 978-3-642-21515-5. https://doi.org/10.1007/978-3-642-21515-5_36.

Storn, R., & Price, K. (1997). Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, *11*(4), 341-359. https://doi.org/10.1023/a:1008202821328.

Tanabe, R., & Fukunaga, A. (2013). Success-history based parameter adaptation for differential evolution. In *2013 IEEE Congress on Evolutionary Computation* (pp. 71-78). IEEE. Cancun, Mexico. https://doi.org/10.1109/cec.2013.6557555.

Tanabe, R., & Fukunaga, A.S. (2014). Improving the search performance of SHADE using linear population size reduction. In *2014 IEEE Congress on Evolutionary Computation* (pp. 1658-1665). IEEE. Beijing, China. https://doi.org/10.1109/cec.2014.6900380.

Tang, Z., Wang, K., Tao, S., Todo, Y., Wang, R.L., & Gao, S. (2023). Hierarchical manta ray foraging optimization with weighted fitness-distance balance selection. *International Journal of Computational Intelligence Systems, 16*(114), 1-30. https://doi.org/10.1007/s44196-023-00289-4.

Rao, R.V. (2016). Jaya: a simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *International Journal of Industrial Engineering Computations*, *7*, 19-34. https://doi.org/10.5267/j.ijiec.2015.8.004.

Wadood, A., Farkoush, S.G., Khurshaid, T., Yu, J.T., Kim, C.H., & Rhee, S.B. (2019). Application of the JAYA algorithm in solving the problem of the optimal coordination of overcurrent relays in single- and multi-loop distribution systems. *Complexity, 2019*(1), 1-13. https://doi.org/10.1155/2019/5876318.

Wang, Y., Yu, Y., Gao, S., Pan, H., & Yang, G. (2019). A hierarchical gravitational search algorithm with an effective gravitational constant. *Swarm and Evolutionary Computation*, *46*, 118-139. https://doi.org/10.1016/j.swevo.2019.02.004.

Warid, W., Hizam, H., Mariun, N., & Abdul-Wahab, N.I. (2016). Optimal power flow using the Jaya algorithm. *Energies, 9*(9), 678. https://doi.org/10.3390/en9090678.

Wright, A.H. (1991). Genetic algorithms for real parameter optimization. *Foundations of Generic Algorithms, 1*, 205-218. https://doi.org/10.1016/b978-0-08-050684-5.50016-1.

Wu, C., & He, Y. (2020). Solving the set-union knapsack problem by a novel hybrid Jaya algorithm. *Soft Computing, 24*(3), 1883-1902. https://doi.org/10.1007/s00500-019-04021-3.

Xia, X., Gui, L., Zhang, Y., Xu, X., Yu, F., Wu, H., Wei, B., He, G., Li, Y., & Li, K. (2021). A fitness-based adaptive differential evolution algorithm. *Information Sciences, 549*, 116-141. https://doi.org/10.1016/j.ins.2020.11.015.

Yi, W., Chen, Y., Pei, Z., & Lu, J. (2022). Adaptive differential evolution with ensembling operators for continuous optimization problems. *Swarm and Evolutionary Computation, 69*, 100994. https://doi.org/10.1016/j.swevo.2021.100994.

Yu, J.T., Kim, C.H., Wadood, A., Khurshaid, T., & Rhee, S.B. (2019a). Jaya algorithm with self-adaptive multi-population and lévy flights for solving economic load dispatch problems. *IEEE Access, 7*, 21372-21384. https://doi.org/10.1109/access.2019.2899043.

Yu, K., Liang, J.J., Qu, B.Y., Chen, X., & Wang, H. (2017). Parameters identification of photovoltaic models using an improved JAYA optimization algorithm. *Energy Conversion and Management, 150*, 742-753. https://doi.org/10.1016/j.enconman.2017.08.063.

Yu, Y., Gao, S., Wang, Y., & Todo, Y. (2019b). Global optimum-based search differential evolution. *IEEE/CAA Journal of Automatica Sinica, 6*(2), 379-394. https://doi.org/10.1109/jas.2019.1911378.

Zhang, J., & Sanderson, A.C. (2009). JADE: adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation, 13*(5), 945-958. https://doi.org/10.1109/tevc.2009.2014613.

Zhang, Y., Chi, A., & Mirjalili, S. (2021). Enhanced Jaya algorithm: a simple but efficient optimization method for constrained engineering design problems. *Knowledge-Based Systems, 233*, 107555. https://doi.org/10.1016/j.knosys.2021.107555.

Zhao, W., Wang, L., & Zhang, Z. (2019). A novel atom search optimization for dispersion coefficient estimation in groundwater. *Future Generation Computer Systems, 91*, 601-610. https://doi.org/10.1016/j.future.2018.05.037.

Zhao, W., Zhang, Z., & Wang, L. (2020). Manta ray foraging optimization: an effective bio-inspired optimizer for engineering applications. *Engineering Applications of Artificial Intelligence, 87*, 103300. https://doi.org/10.1016/j.engappai.2019.103300.

Zheng, L.M., Zhang, S.X., Tang, K.S., & Zheng, S.Y. (2017). Differential evolution powered by collective information. *Information Sciences, 399*, 13-29. https://doi.org/10.1016/j.ins.2017.02.055.