An improved Jaya optimization algorithm with Lévy flight

Giovanni Iacca[a], Vlademir Celso dos Santos Junior[b], Vinícius Veloso de Melo[c]

[a] *Department of Information Engineering and Computer Science*
University of Trento
Povo, Italy
[b] *Institute of Science and Technology*
Federal University of São Paulo
São José dos Campos, São Paulo, Brazil
[c] *Wawanesa Insurance*
Department of Analytics
Winnipeg, Manitoba, Canada

Email addresses: giovanni.iacca@unitn.it (Giovanni Iacca), vladejrs@hotmail.com (Vlademir Celso dos Santos Junior), vvdemelo@wawanesa.com (Vinícius Veloso de Melo)

Abstract

Recent advances in metaheuristics have shown the advantages of using the Lévy distribution, which models a kind of random walk (named "Lévy flight") with occasional "big" steps. This characteristic makes Lévy flight especially useful for performing large "jumps" that allow the search to escape from a local optimum and restart in a different region of the search space. In this paper, we investigate this idea by applying Lévy flight to Jaya, a simple yet effective Swarm Intelligence optimization algorithm recently proposed in the literature. We perform experiments on the CEC 2014 benchmark as well as five industrial optimization problems taken from the CEC 2011 benchmark, and compare the performance of the proposed Lévy flight Jaya Algorithm (LJA) against several state-of-the-art algorithms for continuous optimization. Our numerical results show that, although both Jaya and LJA are in general less efficient than the most advanced algorithms on the CEC 2014 benchmark, LJA largely outperforms the original Jaya algorithm in most cases, and is also highly competitive on the tested industrial problems.

Keywords: Continuous Optimization, Swarm Intelligence, Jaya, Lévy Flight

1. Introduction

Optimization methods often deal with hard-to-solve problems with a large number of decision variables, with many local optima, and computationally expensive calculations: in these situations, searching for solutions that minimize or maximize the given objective function as efficiently as possible can become incredibly challenging. For this purpose, researchers have been proposing different kinds of optimization algorithms over the years, whose applicability and performance typically depend on the features of the problem at hand and its mathematical formulation.

A broad class of general-purpose optimization algorithms consists of the so-called metaheuristics, i.e., algorithms that use various computational intelligence strategies to guide the process of exploring the solution space in search of the best value. Some of these metaheuristics are nature-inspired, such as Evolutionary Algorithms, that try to replicate the mechanisms behind biological evolution, or Swarm Intelligence algorithms, that instead mimic the collective behavior of different species of animals or plants, in order to achieve a more efficient search for the global optimum. Well-known examples of Swarm Intelligence are –to name a few– the Particle Swarm Optimization (PSO) (Kennedy, 2010), based on the movement of organisms in a bird flock or fish school, the Firefly Algorithm (FA) (Yang, 2010b), based on the flashing light patterns of fireflies, or the Artificial Bee Colony (ABC) (Karaboga & Basturk, 2007), based on the collective behavior of honey bees. There exist, though, several other algorithms inspired by different natural phenomena.

Importantly, most metaheuristics rely on some (typically fine-tuned) parameters whose effect can dramatically change the algorithmic performance, depending on the specific function to be optimized. On the other hand, from a practitioner's point of view, it would be desirable to have an algorithm that requires as few parameters as possible to generalize its use without the need for further efforts to adjust the parameter setting for each specific problem. Therefore, a recent trend in metaheuristic optimization research is to devise self-adaptive algorithms that can be applied "out-of-the-box" to any optimization problem, with no tuning at all. An example that goes in this direction is the Teaching-Learning-Based Optimization (TLBO) algorithm (Rao et al., 2011), which only needs two parameters, i.e., the population size and the number of generations, which are somehow indispensable for all Swarm Intelligence algorithms. No other parameter setting is needed, hence applying this algorithm is straightforward. In the past few years, TLBO has inspired many other algorithms based on similar principles, among which the so-called Jaya algorithm (Rao, 2016), which also does not need any other parameter besides the population size and the number of generations, has lately shown promising results on various kinds of optimization problems.

Even though such self-adaptive algorithms are of great value from an application point of view, in most cases they still suffer from a more fundamental problem that underlies practically all metaheuristics: stochasticity. Indeed, while metaheuristics perform –in one way or another– a "guided" search towards better values of the objective function, they are, essentially, stochastic processes in which randomness is used either to promote a better exploration of the search space, to escape from a local optimum, or to refine local search. In TLBO, as well as in many other Swarm Intelligence algorithms, stochasticity is usually implemented through sampling random numbers from a uniform distribution $\mathcal{U}(0,1)$. Other algorithms use instead the Gaussian distribution, see e.g. the Gaussian mutations in Evolutionary Algorithms (Opara & Arabas, 2018). Regardless the specific way in which random numbers are used in metaheuristics, one problem with the commonly-used uniform and Gaussian distributions is that they may prevent the algorithm from escaping a local optimum: the more the generations advance, the more the solutions in the population get close to each other, such that it becomes more difficult for the algorithm to generate new solutions that are far away from the existing ones. Although such convergence is usually a desirable exploitation effect, it may be useless if the population has not reached the region where the global optimum lies yet. In other words, these distributions do not promote "bigger steps" in the search space, which can be occasionally needed to get a stuck search unstuck.

To overcome this issue, one obvious solution is to restart the algorithm according to some criteria (Caraffini et al., 2013; de Melo & Iacca, 2014). However, one could also think of "embedding" the restart mechanism directly into the way random numbers are sampled during the algorithm execution: in order to do that, recent works have proposed the use of different distributions with different properties (e.g., kurtosis and skewness) and therefore different effects on the algorithmic functioning. One particularly interesting case is the Lévy distribution (Viswanathan et al., 1999), which is used to model the so-called "Lévy flight", a kind of random walk with occasional "big" steps. In optimization, this characteristic makes Lévy flight especially useful for performing large "jumps", which are precisely what is needed to allow a possibly stuck algorithm to escape from a local optimum and restart the search in a different region of the search space.

Here, we investigate this idea by applying Lévy flight to Jaya. The choice of Jaya for this study is motivated by its extreme algorithmic simplicity, coupled with a good optimization performance, which allows us to focus on the effect of Lévy flight without the need to perform an ablation study to dissect the algorithm in all its components. A preliminary investigation of this kind has been recently conducted in (Bekdaş, 2019), although that study focused on the application of various hybrid variants of Lévy flight-based Jaya to a structural design problem, rather than a thorough algorithmic analysis, that is the purpose of the present work. Furthermore, in (Bekdaş, 2019) the details of the Lévy flight model are not discussed, nor the effect of the parametrization on the algorithmic performance. To cover this gap, here we study the effect of applying Lévy flight to Jaya thoroughly; also, we discuss the details of the Lévy flight model, and the effect of the β parameter used in the model. Finally, to assess the performance of the proposed Lévy flight Jaya Algorithm (LJA), we conduct numerical experimentation on the CEC 2014 benchmark (Liang et al., 2013) and five industrial optimization problems taken from the CEC 2011 benchmark on real-world problems (Das & Suganthan, 2010). We compare the performance of LJA against that of the original Jaya, as well as other state-of-the-art metaheuristics and deterministic global search algorithms for continuous optimization. Our experiments show, on the one hand, that the inclusion of the Lévy flight mechanism greatly improves the performance of the original Jaya algorithm and makes it competitive, at least in some limited cases, against the compared algorithms; on the other hand, we also observed –as expected– that the performance of a very simple algorithm such as LJA is in general inferior to that of the most advanced metaheuristics available in the literature.

The remaining of this paper is organized as follows. In the next section, we provide a brief survey of the related works on Lévy flight applied to Swarm Intelligence algorithms. Then, in Section 3, we introduce the main concepts of Lévy flight and Jaya. In Section 4, we introduce the proposed Lévy flight Jaya Algorithm (LJA) obtained by applying the Lévy flight model to the original Jaya algorithm. Section 5 describes the experimental setup, the comparison methodology, and presents the numerical results. Finally, in Section 6 we give the conclusions of this work.

2. Related works

Various works have recently investigated the use of Lévy flight in Swarm Intelligence algorithms. For instance, the Lévy flight PSO (Hariya et al., 2015) works similarly to the traditional PSO, except for the fact that the inertia coefficient is defined by the Lévy distribution. In particular, at each iteration, each particle has a position and a velocity, and the velocity at the next iteration is updated based on the sum of: 1) the current velocity, multiplied by a random number sampled from the Lévy distribution, 2) the vector pointing to the personal best, multiplied by a random number sampled from a uniform distribution, and 3) the vector pointing to the global best, multiplied by another random number sampled from a uniform distribution. The new position is then defined by the sum of the current position and the new velocity.

Another algorithm based on Lévy flight is the Cuckoo Search algorithm (CS) (Yang & Deb, 2009), whose search logic is based on a cuckoo species behavior. This algorithm first generates n random solutions, which are called "nests". Then, a new solution, which represents a cuckoo egg, is generated through the Lévy distribution, and is compared to the solution of a random nest: if the new solution is better than the previous one, then the previous solution is replaced. Finally, a fraction of the worst solutions is replaced by new random solutions, and a new solution is again generated through the Lévy distribution, such that the process is repeated until reaching a stopping criterion. A multi-objective version of CS with Lévy flight was presented in (Nguyen & Vo, 2017), while two interesting applications of CS with Lévy flight can be found in (Pandey et al., 2019), where the algorithm is used for power optimization in smartphone displays, and (Majumder & Laha, 2016), where a discrete version of CS with Lévy flight is used for solving a 2-machine robotic cell scheduling problem.

The Lévy flight Firefly algorithm (LFA) (Yang, 2010a) mimics instead the behavior of fireflies, which use flashing lights to attract mating partners. In this algorithm, each possible solution is considered a firefly with a light intensity proportional to its fitness, and an attractiveness, directly proportional to its brightness and inversely proportional to its distance from another solution. The less bright fireflies move towards brighter fireflies in steps proportional to the attractiveness, according to a value sampled from the Lévy distribution. A "compact" variant of LFA also exists (Tighzert et al., 2018), which uses a probabilistic distribution model (Neri et al., 2013) instead of an actual swarm of fireflies. Of note, a GPU-based variant of LFA was implemented in (Kalantzis et al., 2016), for solving a constrained optimization problem related to the treatment planning of intensity-modulated radiation therapy.

The Lévy flight Krill Herd algorithm (LKH) (Wang et al., 2013) bases its working principles on the behavior of krill swarms. In this case, the position of each solution is defined by the movement of other solutions, the foraging motion, and the random diffusion, while the step-size is sampled from the Lévy distribution.

The Lévy flight Artificial Bee Colony (LFABC) (Sharma et al., 2016) also uses Lévy flight to determine the step-size used for moving solutions. However, in this case, each solution is considered a food source, which is manipulated by three groups of bees: 1) worker bees, that modify the food sources based on personal experience and the fitness of the new food source; 2) onlooker bees, that receive information from the worker bees and based on that look for better solutions; and 3) scout bees, that replace abandoned food sources with randomly chosen food sources.

The Flower Pollination algorithm (FPA) (Yang, 2012), inspired by how pollination occurs in nature, considers each solution as a pollen particle that moves in the search space according to two different position update rules: local pollination and global pollination. At each step, one of the two rules is chosen randomly: if the chosen movement is local pollination, the particle moves in a limited neighborhood, and the step-size is multiplied by a random number sampled from the uniform distribution $\mathcal{U}(0,1)$; if the global movement is chosen, the particle moves towards the global best, and the step-size is multiplied by a random number sampled from the Lévy distribution.

Finally, in (Wang et al., 2019) Lévy flight was combined with three *ad-hoc* local search methods for solving a specific scheduling problem, that is the dynamic allocation of berths in seaports. The peculiarity of this work is that, in order to handle the combinatorial nature of the problem, the step-sizes generated by the algorithm are rounded to integer numbers.

3. Background

In the following we introduce the original Jaya algorithm (Subsection 3.1) and the Lévy flight model (Subsection 3.2).

3.1. The Jaya algorithm

Jaya is a recent Swarm Intelligence optimization algorithm proposed in (Rao, 2016). Its name comes from the word "victory" in the Sanskrit language, as it was designed in such a way that it needs as few parameters as possible. The result is an algorithm that only requires two parameters, i.e., the population size and the maximum number of generations. As in most Swarm Intelligence algorithms, the functioning of Jaya is based on the principle that each solution has a particular "position" in the search space (corresponding to its variables) and the search process should change these positions so to guide the solutions towards the optimum. In particular, Jaya updates the positions based on the assumption that each solution should move away from the worst solution found so far during the search process, and get closer to the best one. This update rule is described by the following equation:

$$X_{j,k}^{i+1} = X_{j,k}^{i} + r_1(X_{j,best}^{i} - |X_{j,k}^{i}|) - r_2(X_{j,worst}^{i} - |X_{j,k}^{i}|), \tag{1}$$

where $X_{j,k}^{i}$ is the value of the j^{th} variable of the k^{th} particle at the i^{th} generation, $X_{j,best}^{i}$ is the value of the j^{th} variable of the best solution in the i^{th} generation, ${X^i}_{j,worst}$ is the value of the j^{th} variable of the worst solution in the i^{th} generation, r_1 and r_2 are two random numbers sampled from the uniform distribution $\mathcal{U}(0,1)$, and $X_{j,k}^{i+1}$ is the j^{th} variable of X_k^{i+1}, i.e. the new solution (position) to be evaluated. In case of improved fitness, X_k^{i+1} replaces X_k^i. The complete pseudo-code of the Jaya algorithm is provided in Algorithm 1.

Algorithm 1 Pseudo-code of the Jaya algorithm.

```
1: Input: objective function ($f$), problem bounds ($LB$, $UB$), problem dimensionality ($D$),
2:        population size ($P$), maximum number of generations ($G$)
3: Output: best solution and best objective function value
4: // initialization
5: initialize $i = 0$
6: randomly initialize $P$ solutions within the problem bounds ($LB$, $UB$)
7: evaluate the initial $P$ solutions
8: // loop
9: while $i < G$ do
10:    find the best and worst solutions in the current population
11:    for $k \in [1, \ldots, P]$ do
12:        for $j \in [1, \ldots, D]$ do
13:            calculate $X_{j,k}^{i+1}$ according to Eq. (1)
14:            // saturate $X_{j,k}^{i+1}$ within the problem bounds ($LB$, $UB$)
15:            if $X_{j,k}^{i+1} < LB_j$ then
16:                $X_{j,k}^{i+1} = LB_j$
17:            else if $X_{j,k}^{i+1} > UB_j$ then
18:                $X_{j,k}^{i+1} = UB_j$
19:        // replace the current solution in case of improvement
20:        if $f(X_k^{i+1}) < f(X_k^i)$ then
21:            $X_k^i = X_k^{i+1}$
22:    $i = i + 1$
```

As one may see, the algorithm is extremely simple (it can be coded in a very few lines), and it does not require any other parameter except the population size (P) and the number of generations (G). Therefore, it can be applied straightforwardly to various optimization problems without the need for a specific parameter tuning, as it has been shown in the context of engineering optimization (Rao & Saroj, 2017), urban traffic light scheduling (Gao et al., 2017), image classification (Wang et al., 2017; Zhang et al., 2016), and binary problems (Aslan et al., 2019). Furthermore, the simplicity of Jaya makes it also a suitable algorithm to work with from an algorithmic point of view, in order to propose simple changes that can improve its performance without affecting its general simple algorithmic design, see (Iacca et al., 2012; Piotrowski & Napiorkowski, 2018) for a more complete discussion on why and how metaheuristics should be kept simple.

3.2. Lévy flight

Many flying animals show a flying behavior with characteristics typical of Lévy flight (Viswanathan et al., 1999; Heidari & Pahlavani, 2017). Lévy flight is a kind of random walk in which the probability density function (PDF) of the step-sizes is heavy-tailed, which means that a particle that moves according to Lévy flight performs occasional "big" steps interspersed with many frequent "small" steps. A typical motion pattern goes like this: the particle moves locally at first, performing a number of small steps, then it performs a big step and, after that, it moves again locally. From a mathematical point of view, the Lévy flight PDF can be defined as:

$$Distribution_{Lévy} = U/|V|^{1/\beta}, \tag{2}$$

where β denotes the power-law index, V denotes a random number sampled from the Gaussian distribution $\mathcal{N}(0,1)$, and U is a random number sampled from the Gaussian distribution $\mathcal{N}(0,\sigma^2)$, where the std. dev. σ is given by:

$$\sigma = \left(\frac{\Gamma(1+\beta) * sin(\frac{\pi*\beta}{2})}{\Gamma(\frac{1+\beta}{2}) * \beta * 2^{\frac{\beta-1}{2}}} \right)^{\frac{1}{\beta}}, \tag{3}$$

where Γ denotes the gamma function.

Figure 1 shows a 2-dimensional random walk compared to a 2-dimensional Lévy flight, where it is possible to see the difference between the two patterns: random walk provides approximately the same size for every step, while Lévy flight performs a number of small steps and occasionally a big step.

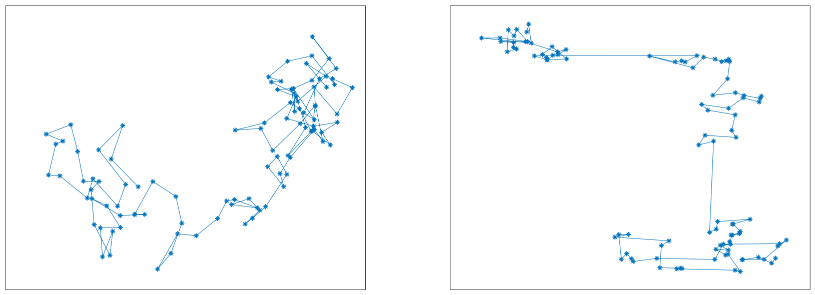

Figure 1: Random walk (left) vs. Lévy flight (right).

4. Lévy flight Jaya algorithm (LJA)

As shown in previous literature (Rao, 2016), in some cases the original Jaya algorithm may not be able to find the global optimum, due to the presence of local optima that can get the search trapped. To overcome this problem, we propose to modify the original Jaya algorithm such that the size of its steps are multiplied by random numbers –see r_1 and r_2 in Eq. (1)– which are no longer sampled from the uniform distribution $\mathcal{U}(0,1)$, but rather from the Lévy distribution. In the proposed approach, named “Lévy flight Jaya algorithm” (LJA), the particles perform a local search and occasionally “jump” to another region of the search space for a new local search. This way, the risk that the algorithm gets stuck in a local optimum is drastically reduced, while it is still possible to perform sufficient local refinements. In other words, the algorithm presents a natural balance between exploration and exploitation. The pseudo-code of LJA is the same as that of the original Jaya algorithm, shown in Algorithm 1, with the only exception being that instead of using Eq. (1) to update the particle position (line 13 in the pseudo-code), the following equation is used:

$$X_{j,k}^{i+1} = X_{j,k}^{i} + |levy_1|(X_{j,best}^{i} - |X_{j,k}^{i}|) - |levy_2|(X_{j,worst}^{i} - |X_{j,k}^{i}|), \quad (4)$$

where $levy_1$ and $levy_2$ are two random numbers sampled from the Lévy distribution. It should be noted that the only additional parameter with respect to the original Jaya algorithm is the power-law index β, needed in Eq. (2) to sample random numbers from the Lévy distribution. Despite being a simple change in the algorithm, this new distribution induces drastic changes in the optimization process which, as we show below, in turn lead to an increase in the overall performance in terms of solution quality.

5. Numerical experiments

In this section, we first present the experimental setup (Subsection 5.1), followed by an explanation of the overall experimental methodology (Subsection 5.2). We then report a detailed analysis of the effect of the β parameter (Subsection 5.3), followed by a comparison of the performance of the proposed LJA against Jaya (Subsection 5.4), a comparison against the metaheuristics that participated in the CEC 2014 competition (Subsection 5.5), and a comparison against twp deterministic global search algorithms (Subsection 5.6). We conclude the experimentation with an analysis (Subsection 5.7) of the results of the proposed LJA on five real-world industrial optimization problems taken from the CEC 2011 benchmark.

5.1. Experimental setup

We considered in our experimentation two different sets of problems, namely the 30 benchmark functions of the CEC 2014 benchmark (Liang et al., 2013) and five industrial optimization problems taken from the CEC 2011 benchmark (Das & Suganthan, 2010). The details of the tested problems are presented below.

5.1.1. CEC 2014 benchmark

To assess the performance of the proposed Lévy flight Jaya Algorithm, in the first parts of the experimentation we considered the 30 benchmark functions used at the CEC 2014 competition. The main details of the benchmark functions are summarized in Table 1, while the complete mathematical description of each function can be found in (Liang et al., 2013). Of note, all functions are non-separable except for f_8 and f_{10}.

5.1.2. Real-world industrial optimization problems

The last part of our experimentation is devoted to evaluate LJA on real-world optimization problems. In order to do that and ensure reproducibility, we considered five publicly available problems included in the CEC 2011 benchmark for real-world optimization (Das & Suganthan, 2010), namely P_1, P_2, P_3, P_4 and P_5, respectively with 6, 30, 1, 1, and 30 decision variables. It is important to

Table 1: CEC 2014 benchmark functions.

Function types	No.	Function	Optimum
Unimodal	1	Rotated High Conditioned Elliptic Function	100
	2	Rotated Bent Cigar Function	200
	3	Rotated Discus Function	300
Simple Multimodal	4	Shifted and Rotated Rosenbrock Function	400
	5	Shifted and Rotated Ackley Function	500
	6	Shifted and Rotated Weierstrass Function	600
	7	Shifted and Rotated Griewank Function	700
	8	Shifted Rastrigin Function	800
	9	Shifted and Rotated Rastrigin Function	900
	10	Shifted Schwefel Function	1,000
	11	Shifted and Rotated Schwefel Function	1,100
	12	Shifted and Rotated Katsuura Function	1,200
	13	Shifted and Rotated HappyCat Function	1,300
	14	Shifted and Rotated HGBat Function	1,400
	15	Shifted and Rotated Expanded Griewank plus Rosenbrock Function	1,500
	16	Shifted and Rotated Expanded Scaffer F6 Function	1,600
Hybrid	17	Hybrid Function 1 (N=3)	1,700
	18	Hybrid Function 2 (N=3)	1,800
	19	Hybrid Function 3 (N=4)	1,900
	20	Hybrid Function 4 (N=4)	2,000
	21	Hybrid Function 5 (N=5)	2,100
	22	Hybrid Function 6 (N=5)	2,200
Composition	23	Composition Function 1 (N=5)	2,300
	24	Composition Function 2 (N=3)	2,400
	25	Composition Function 3 (N=3)	2,500
	26	Composition Function 4 (N=5)	2600
	27	Composition Function 5 (N=5)	2,700
	28	Composition Function 6 (N=5)	2,800
	29	Composition Function 7 (N=3)	2,900
	30	Composition Function 8 (N=3)	3,000
Domain		$[-100, 100]^D$	

note that these functions are only box-constrained. A brief description of these five problems follows:

- P_1 (Parameter Estimation for Frequency-Modulated (FM) Sound Waves): This problem consists in optimizing 6 variables which describe a wave signal that correspond to generating a sound similar to a given target sound. The fitness function, to be minimized, is the sum of the squared errors between the estimated wave and the target wave. This problem is highly multimodal and has a strong epistasis.

- P_2 (Lennard-Jones Potential Problem): This problem consists in finding the position of N atoms to form the Lennard-Jones cluster having the lowest molecular potential energy. The resulting optimization problem requires to find the three coordinates x, y and z for each atom ($N = 10$ in this study, thus resulting into a 30-dimensional problem) and displays a number of local minima increasing exponentially with N.

- P_3 (Bifunctional Catalyst Blend Optimal Control Problem): This problem consists in optimizing a chemical process which converts methylcyclopentane to benzene in a tubular reactor. In particular, the goal is to find the optimal value of one single variable, the catalyst blend $u \in [0.6, 0.9]$ such that a performance metric J representing the benzene concentration at the exit of the reactor is *maximized*. This problem is reported to have as many as 300 local optima.

- P_4 (Optimal Control of a Non-Linear Stirred Tank Reactor): This problem consists in optimizing a first-order irreversible chemical reaction carried out in a continuous stirred tank reactor (CSTR). Similarly to P_3, the goal is to find the optimal value of one single variable, u so that a performance index J is minimized. It should be noted that this problem does not have any predefined bound for u, but the initial guess lies in $[0.0, 5.0]$. Also this problem is highly multimodal.

- P_5 (Tersoff Potential for model Si (B)): This problem consists in minimizing the Tersoff potential of a set of silicon atoms. Similarly to P_2, the problem requires to find the three coordinates x, y and z for each atom ($N = 10$ in this study, thus resulting into a 30-dimensional problem) and displays a number of local minima increasing exponentially with N.

5.2. Methodology

Except for the experiments on the real-world problems (see Subsection 5.7), for which we executed 25 independent runs for each tested algorithm on each problem, in all the experiments on the CEC 2014 benchmark we executed each tested algorithm for 51 independent runs on each test function[1]. For the algorithms that participated in the CEC 2014 competition, we considered the original raw data available online[2]. LJA was parametrized with a population of $P = 5 \times D$ solutions and a maximum number of generations $G = 2,000$, thus for a total of $10,000 \times D$ function evaluations as indicated in (Liang et al., 2013), where D is the problem dimensionality (depending on the experiments, either $D = 10$ or $D = 30$). As for the power-law index β, as we describe below, its value was chosen after performing a preliminary analysis to verify the existence of a statistical difference between three different values.

All the experiments have been conducted using the Stochastic Optimisation Software (SOS) Java platform (Caraffini & Iacca, 2020) on a workstation powered by an Intel® Core$^{\text{TM}}$ i9-7940x CPU @ 3.10GHz, 64GB RAM. The Java code was tested on Ubuntu 19.10 (kernel GNU/Linux 5.3.0-40-generic x86_64), while the Python 3 `numpy` v1.18.4 and `matplotlib` v3.2.2 libraries were used for data processing. The experimentation was conducted in various steps, which can be summarized as follows.

[1]We set the number of independent runs for each problem according to the indications in (Das & Suganthan, 2010) and (Liang et al., 2013), respectively for the real-world and the benchmark optimization problems.

[2]The raw data of the compared algorithms were taken from: `https://github.com/P-N-Suganthan/CEC2014`

Table 2: List of the compared metaheuristics from the CEC 2014 competition.

Algorithm	Acronym
Differential Evolution with Rotation-Invariant Mutation and Competing Strategies Adaptation (Bujok et al., 2014)	b3e3pbest
Covariance Matrix Leaning and Searching Preference (Chen et al., 2014)	CMLSP
Controlled Restart in Differential Evolution (Poláková et al., 2014)	DE-b6e6rl.w.r.
Differential Evolution Strategy based on the Constraint of Fitness Values Classification (Li et al., 2014)	FCDE
Memetic Differential Evolution Based on Fitness Euclidean-Distance Ratio (Qu et al., 2014)	FERDE
Fireworks Algorithm with Differential Mutation (Yu et al., 2014)	FWA-DM
Gaussian Adaptation based Parameter Adaptation for Differential Evolution (Mallipeddi et al., 2014)	GaAPADE
Linear Population Size Reduction SHADE (Tanabe & Fukunaga, 2014)	L-SHADE
Mean-Variance Mapping Optimization (Erlich et al., 2014)	MVMO
Bee-Inspired Algorithm (Maia et al., 2014)	OptBees
Partial Opposition-Based Adaptive Differential Evolution Algorithms (Hu et al., 2014)	POBL-ADE
Memetic Algorithm based on Regions and Local Search Chains (Molina et al., 2014)	rmalschcma
Non-Uniform Mapping in Real-Coded Genetic Algorithms (Yashesh et al., 2014)	NRGA
Differential Evolution with Replacement Strategy (Xu et al., 2014)	RSDE
Simultaneous Optimistic Optimization (Preux et al., 2014)	SOO
Simultaneous Optimistic Optimization with BOBYQA (Preux et al., 2014)	SOO+BOBYQA
United Multi-operator Evolutionary Algorithms (Elsayed et al., 2014)	UMOEAs

Firstly (see Subsection 5.3), we evaluated the performance of three different β values to choose the best one for the following steps. The performance was evaluated by measuring the error (the lower the better) between the best fitness obtained by the algorithms under comparison and the corresponding known optimum shown in Table 1, averaged across the 51 available runs on each of the 30 CEC 2014 benchmark functions (in this case, for $D = 10$). We then employed the pairwise Wilcoxon rank-sum test (Wilcoxon, 1945), with 5% significance level, as well as the sequentially rejective Holm-Bonferroni procedure (Garcia et al., 2008; Holm, 1979), described in Appendix A, to rank the mean errors of the runs for each value of β on the 30 CEC 2014 benchmark functions. Furthermore, we show the distribution of the errors across the runs, in terms of violin plots (Hintze & Nelson, 1998), and the convergence curves, in order to observe the difference in performance during the optimization process[3].

Secondly (see Subsection 5.4), we compared the original Jaya algorithm against the proposed LJA, again on the 30 CEC 2014 benchmark functions, in this case for $D = 10$ and $D = 30$. Since this specific comparison is between only two algorithms, in this case we used only the pairwise Wilcoxon rank-sum test and did not apply the Holm-Bonferroni procedure. Also in this case we report the violin plots and the convergence curves of the two algorithms.

Thirdly (see Subsection 5.5), we compared LJA against the 17 algorithms that participated in the CEC 2014 competition (Liang et al., 2013) (listed in Table 2). Also in this case we tested the algorithms on the 30 CEC 2014

[3]It should be noted that the CEC 2014 benchmark defines an accuracy of 1e-08, i.e. a problem is considered solved if the error is below this threshold. While some graphical methods have recently been proposed to show how many problems are solved by multiple algorithms at different budget values (Hare et al., 2018; Sergeyev et al., 2018), we have noted in our experimentation that these methods do not provide meaningful information on the CEC 2014 benchmark since none of the tested algorithms –apart from some of those that participated in the CEC 2014 competition– reach that accuracy on a significant number of test problems. For this reason, we preferred not to use these methods but report our comparisons in terms of statistical tests, violin plots, and convergence curves.

benchmark functions for $D = 10$ and $D = 30$, and we used the Wilcoxon rank-sum test as well as the Holm-Bonferroni procedure.

Then (see Subsection 5.6), we compared LJA against two deterministic global search optimization algorithms, namely DIRECT (Jones et al., 1993) and its locally-biased version DIRECT-L (Gablonsky & Kelley, 2001), again on the 30 CEC 2014 benchmark functions for $D = 10$ and $D = 30$, and applied the Wilcoxon rank-sum test as well as the Holm-Bonferroni procedure. This sort of comparison is meant to show how a simple metaheuristic such as LJA compares to much more complex optimizers that use a rather sophisticated search logic.

Finally, in order to show the applicability of LJA to real-world problems, we compared LJA, the original Jaya algorithm, DIRECT and DIRECT-L on the five selected industrial optimization problems described above. Also in this case we used the Wilcoxon rank-sum test and the Holm-Bonferroni procedure to assess the results in statistical terms.

5.3. Effect of the β parameter on LJA performance

We performed an initial analysis to investigate the behavior of the β parameter. As previously explained, this parameter affects the distribution of the step-sizes sampled from the Lévy distribution. Thus, it is important to adjust it to generate useful values that are neither too small nor too big, in the range $[1.0, 2.0]$ to ensure a stable distribution (Chechkin et al., 2008), and in particular with a value ≥ 1.5 to allow a fair compromise between small and big steps (Tran et al., 2014). In the following, we show the analysis for $\beta = \{1.6, 1.8, 2.0\}$ on the CEC 2014 benchmark functions for $D = 10$. The result of the Holm-Bonferroni procedure is reported in Table 3, while the detailed results in terms of average error $\pm$ std. dev. and the outcome of the Wilcoxon rank-sum test are reported in Table B.9 in the Appendix.

Table 3: Holm-Bonferroni procedure (reference: LJA ($\beta = 1.8$), Rank = 2.83e+00) for LJA ($\beta = 1.8$) against LJA ($\beta = 1.8$) and LJA ($\beta = 2.0$) on the CEC 2014 benchmark in 10 dimensions.

j	Optimizer	Rank	z_j	p_j	α/j	Hypothesis
1	LJA ($\beta = 1.6$)	2.17e+00	-3.65e+00	1.30e-04	5.00e-02	Rejected
2	LJA ($\beta = 2.0$)	1.00e+00	-1.00e+01	5.00e-24	2.50e-02	Rejected

Both the pairwise comparisons and the sequential rejection of the Holm-Bonferroni procedure show that superior optimization results can be obtained with $\beta = 1.8$. In particular, it is worth noticing that the value $\beta = 2.0$ leads the algorithm to premature convergence in several cases, as it can be noted in the violin plots and the convergence curves reported respectively in Figure 2 and 3, and it does not allow a sufficient exploitation. On the other hand, as it can be noted in the figures, the value $\beta = 1.6$ allows the algorithm to converge to better values, although these results are worse than those obtained with $\beta = 1.8$. Based on these observations, we chose $\beta = 1.8$ as the best parameter for the subsequent experiments. This value indeed seems to offer a good trade-off between small and big steps across the different CEC 2014 benchmark functions in 10 dimensions.

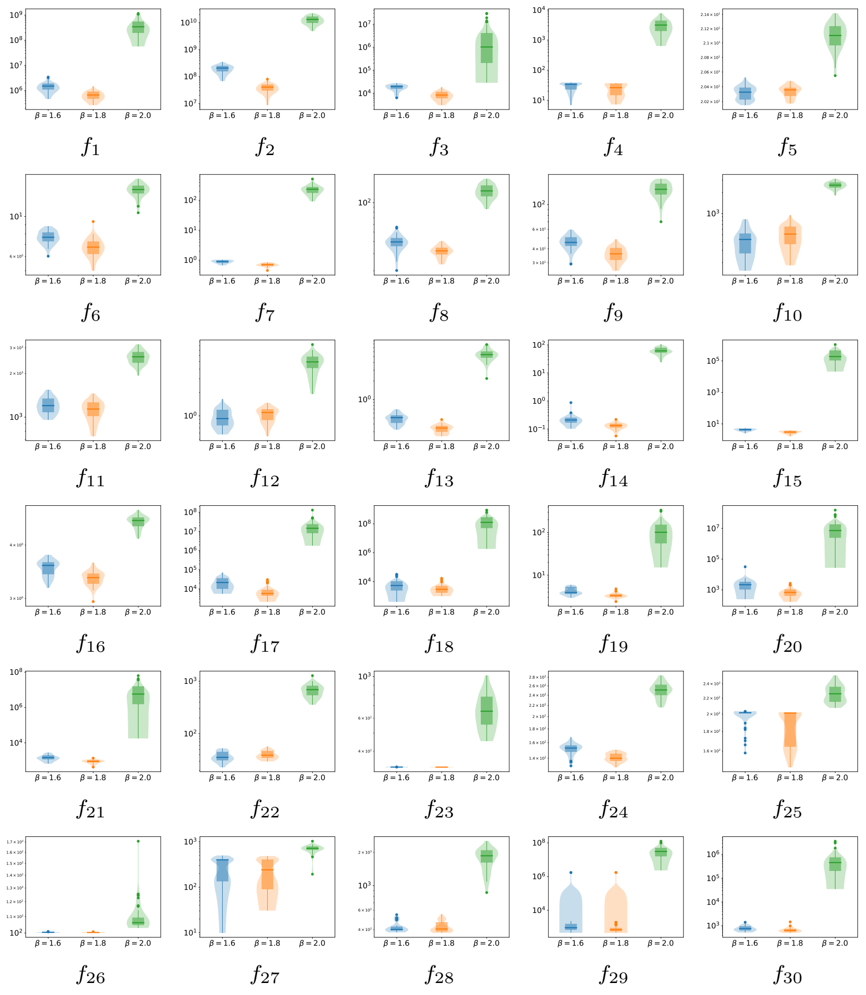

Figure 2: Violin plots for LJA with three values of β on the CEC 2014 benchmark functions with $D = 10$. The y-axis shows the fitness error (in log scale).

5.4. Comparison with Jaya

After choosing the proper value of β, we proceeded to the comparison between the original Jaya algorithm and the proposed LJA. To obtain a fair comparison, we parametrized Jaya with the same parameter setting of LJA, i.e. with a population of $P = 5 \times D$ solutions and a maximum number of generations $G = 2,000$. In this case we considered the CEC 2014 benchmark functions for

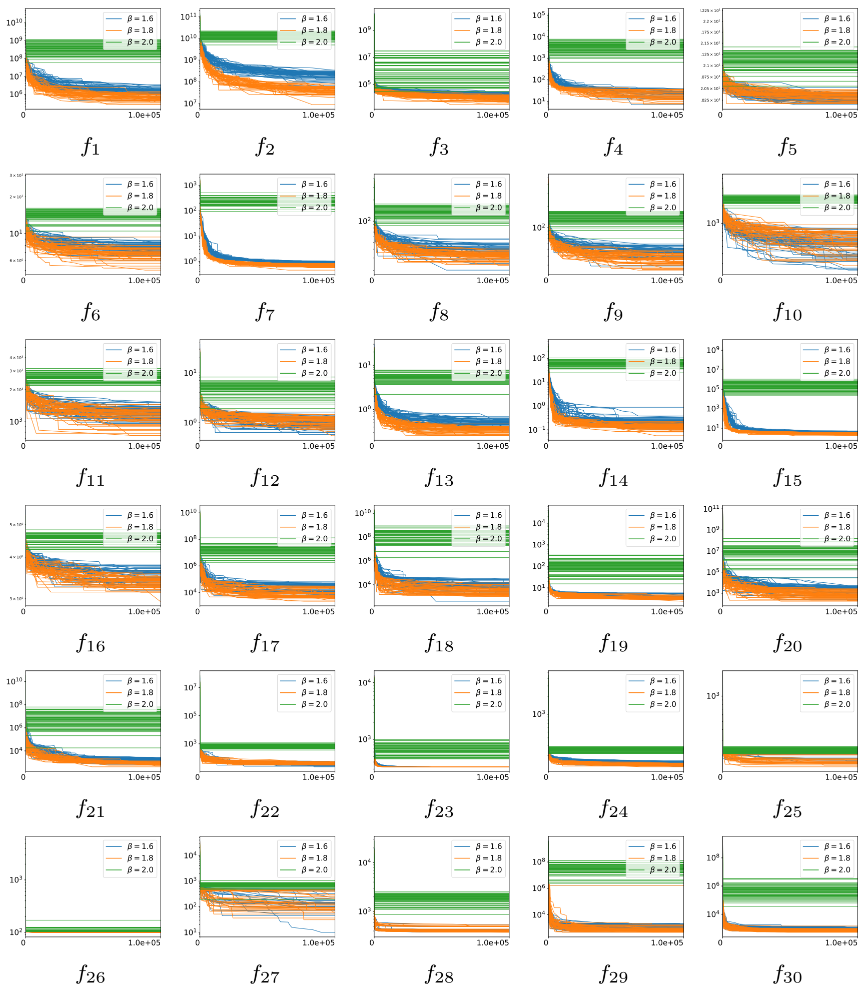

Figure 3: Convergence curves for LJA with three values of β on the CEC 2014 benchmark functions with $D = 10$. The x-axis shows the number of function evaluations. The y-axis shows the fitness error (in log scale).

both $D = 10$ and $D = 30$. The detailed results in terms of average error ± std. dev., together with the outcome of the Wilcoxon rank-sum test, are reported in Tables B.10-B.11 in the Appendix.

From the tables, it results that for $D = 10$, LJA statistically outperforms Jaya in 22 out of 30 functions, with 6 ties and only two cases (f_{10} and f_{22}) in which Jaya outperforms LJA. For $D = 30$, the advantage of LJA is further clearer since it outperforms Jaya in 25 out of 30 functions, with 5 ties. Thus we can conclude that the proposed LJA is superior to the original Jaya algorithm.

To gain further insight in the difference between LJA and Jaya, we show also in this case the error distribution in terms of violin plots, see Figures 4-5, and the convergence curves, see Figures 6-7. From the violin plots, it can be seen that for most functions LJA reaches lower errors than Jaya. Furthermore, it can be noted that, in the case of LJA, the median errors (represented by the horizontal bar inside each violin plot) are usually lower than the mean errors, which indicates an asymmetric distribution of the error (tending to lowest values). As for the std. dev., the values for both algorithms are usually low, suggesting that both algorithms are able to converge to similar solutions in most of their respective runs.

As for the convergence curves, it can be observed that the two algorithms tend to converge in most cases with a similar dynamic behavior, apart from some cases such as f_{10} and f_{27} for $D = 10$ where it can noted that a few occasional runs of Jaya reach better results. In general though, it can observed that in most cases LJA tends to converge faster and to better results w.r.t. the original Jaya algorithm.

5.5. Comparison with the metaheuristics from the CEC 2014 competition

The next step of our experimentation consisted in comparing the proposed LJA with the metaheuristics from the CEC 2014 competition. In this case, we ran LJA for $D = 10$ and $D = 30$, to assess its scalability in comparison with the algorithms from the competition. We present in Table 4 the results of the Holm-Bonferroni procedure, while the detailed results in terms of average error $\pm$ std. dev., together with the outcome of the Wilcoxon rank-sum test, are reported in Tables B.14-B.17 in the Appendix.

It is important to remember that the proposal of this paper is to investigate the performance of a different sampling strategy in Jaya. Therefore, the fundamental comparison is the one presented in the previous section. Given the complexity of Jaya and LJA, it is not reasonable to expect it to outperform the CEC 2014 competitors. Nevertheless, the results presented in the current section are useful for understanding how far LJA is from the state-of-the-art algorithms that participated in the CEC 2014 competition.

In Table 4, one can notice that the reference algorithm is UMOEAs because it is the one with the best rank. LJA is the last in the table because it obtained the worst rank. Only the top three algorithms in that table have a performance that is statistically similar to UMOEAs. Therefore, taking UMOEAs as a reference, LJA did not achieve overall competitive results when all functions are considered together. However, one might evaluate individual results to look for scenarios where LJA performs well.

When considering the 1e-08 solution quality error threshold required by the CEC 2014 competition, many competitors can solve at least one of the problems. On the other hand, the best result for LJA is on function f_{14}. However, when making pairwise comparisons, one can notice that LJA does reach better solutions than some competitors. Below, we highlight some examples where LJA statistically outperforms other algorithms.

For the experiment with 10 dimensions, LJA did surprisingly well on several functions and defeated even some of the best algorithms in these cases:

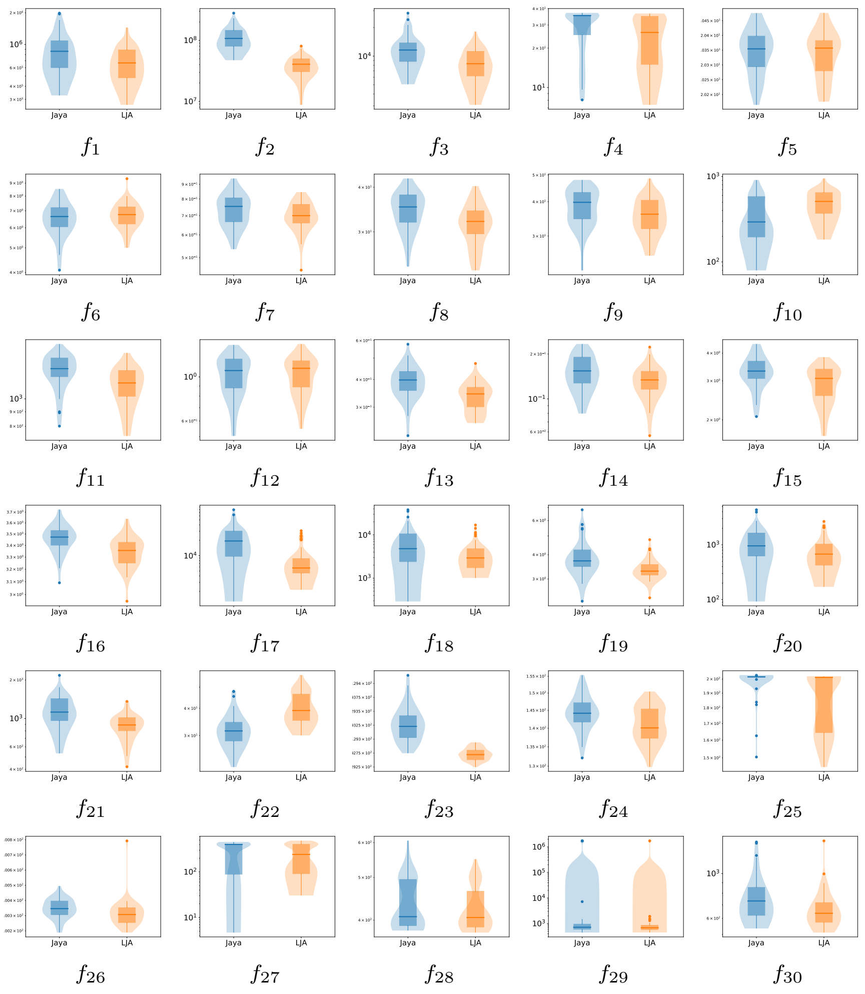

Figure 4: Violin plots for LJA vs Jaya on the CEC 2014 benchmark functions with $D = 10$. The y-axis shows the fitness error (in log scale).

- b3e3pbest: f_4, f_{14}, f_{23}.
- CMLSP: f_{14}, f_{22}.
- L-SHADE: f_4, f_{23}.
- NRGA: $f_{14}, f_{18}, f_{21}, f_{23}, f_{28}, f_{30}$.
- POBL-ADE: f_{14}, f_{23}.
- SOO: $f_1, f_{17}, f_{18}, f_{20}, f_{21}, f_{22}$.

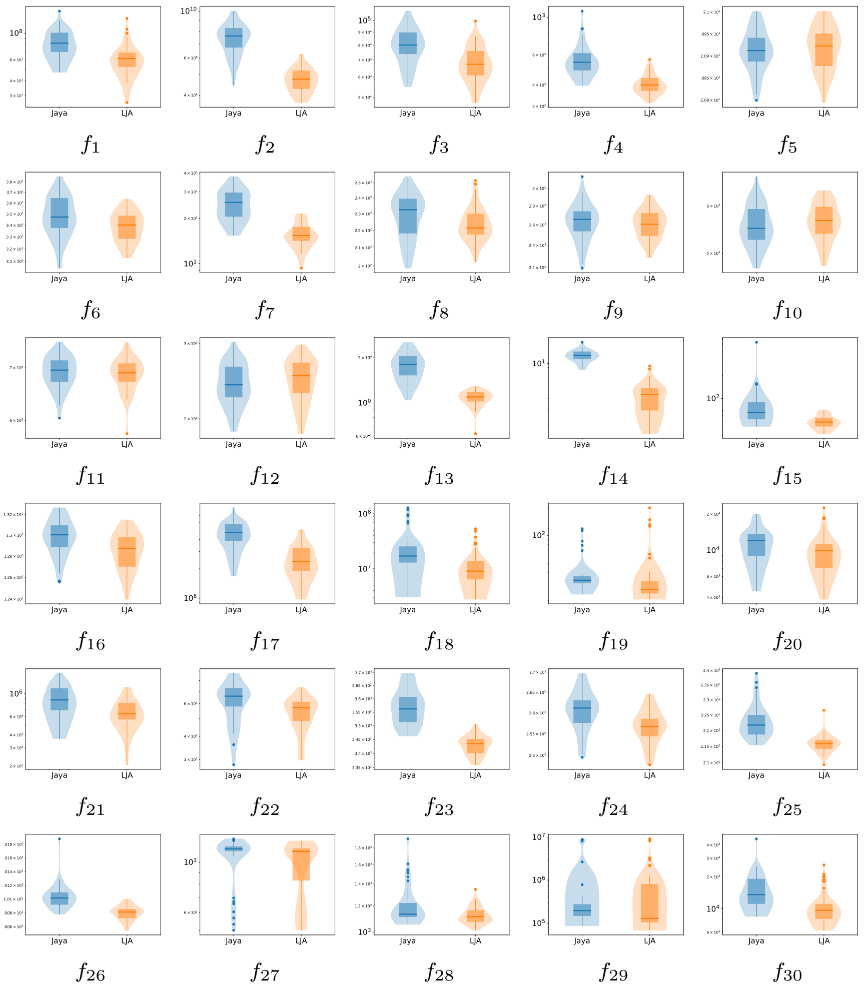

Figure 5: Violin plots for LJA vs Jaya on the CEC 2014 benchmark functions with $D = 30$. The y-axis shows the fitness error (in log scale).

For the experiment with 30 dimensions, LJA only achieved better results than SOO and its variant with BOBYQA:

- SOO: $f_{17}, f_{19}, f_{20}, f_{21}, f_{22}, f_{26}$.
- SOO+BOBYQA: f_{20}, f_{22}, f_{26}.

As it can be noticed, although ranking last in the overall comparison, LJA is still able to find better solutions than other high-quality algorithms, including the top-performer ones such as CMLSP and L-SHADE. However, the functions

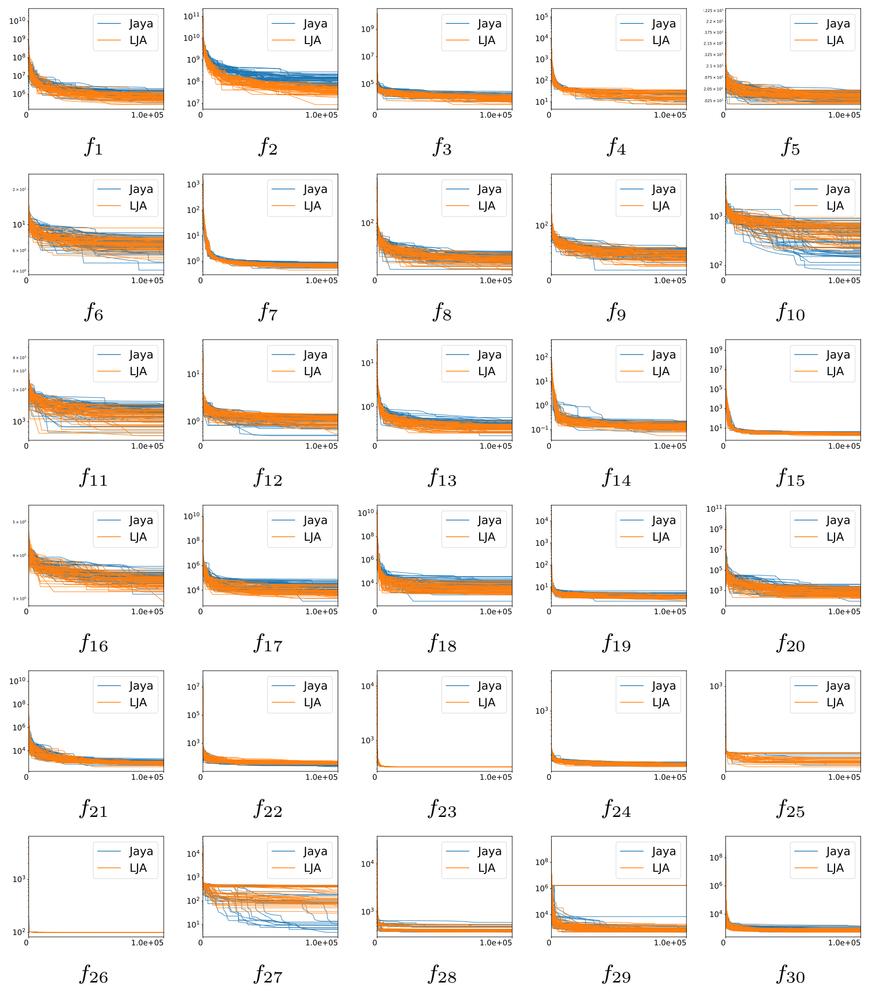

Figure 6: Convergence curves for LJA vs Jaya on the CEC 2014 benchmark functions with $D = 10$. The x-axis shows the number of function evaluations. The y-axis shows the fitness error (in log scale).

above have different characteristics and there is no clear pattern to explain why LJA works better than the competitors on these functions. On the other hand these results are a strong motivation to improve LJA further.

5.6. Comparison with deterministic global optimization algorithms

An often overlooked aspect in the study of metaheuristics is to evaluate how these compare w.r.t. alternative approaches such as mathematical programming methods for global optimization (Kvasov & Mukhametzhanov, 2018). For this reason, we included in our analysis a comparison between the proposed LJA and

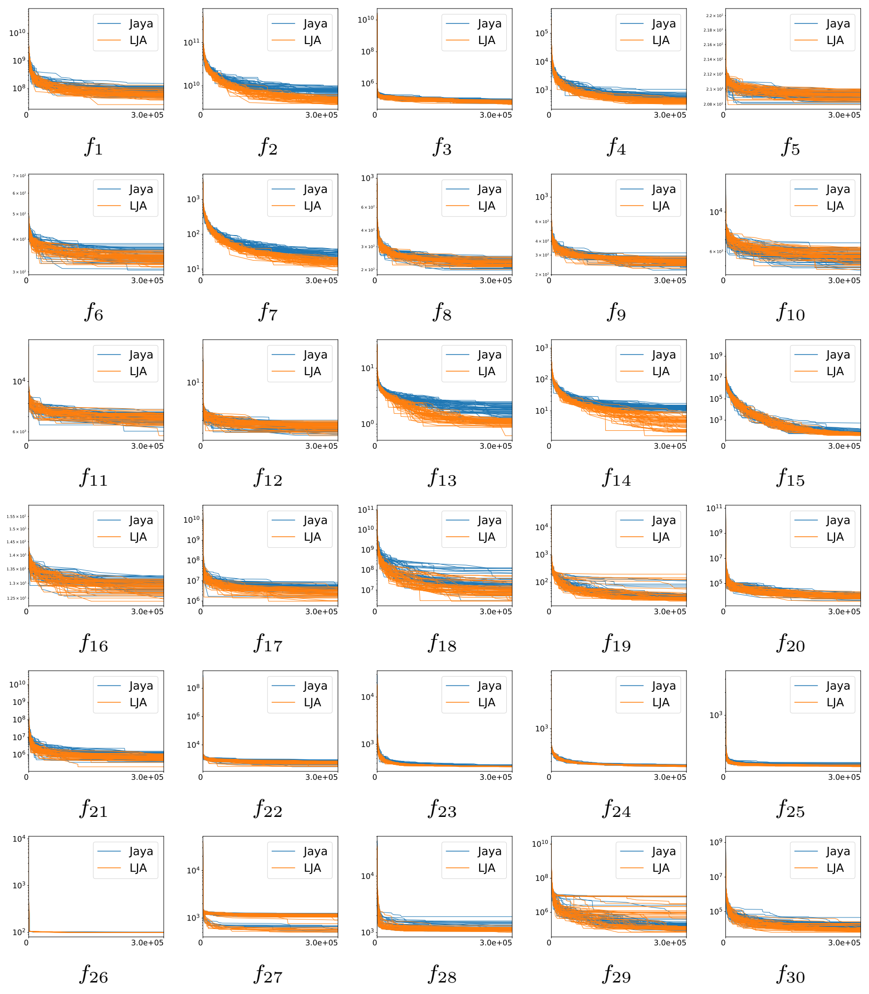

Figure 7: Convergence curves for LJA vs Jaya on the CEC 2014 benchmark functions with $D = 30$. The x-axis shows the number of function evaluations. The y-axis shows the fitness error (in log scale).

two well-known deterministic global search optimization algorithms, namely DIRECT (Jones et al., 1993) and its locally-biased version DIRECT-L (Gablonsky & Kelley, 2001), for which we used a publicly available Java implementation[4]. Both DIRECT variants have been applied to the CEC 2014 benchmark for

[4]Available at: `https://github.com/mike-gimelfarb/optim4j/blob/master/src/main/java/opt/multivariate/unconstrained/order0/direct/DirectAlgorithm.java`. It should be noted that this Java implementation is a direct porting of the original FORTRAN code by (Gablonsky & Kelley, 2001).

Table 4: Holm-Bonferroni procedure (reference: UMOEAs, Rank = 1.44e+01) for LJA against the CEC 2014 competitors on the CEC 2014 benchmark in 10 and 30 dimensions.

j	Optimizer	Rank	z_j	p_j	α/j	Hypothesis
1	L-SHADE	1.42e+01	-1.81e-01	4.28e-01	5.00e-02	Accepted
2	MVMO	1.28e+01	-1.75e+00	3.98e-02	2.50e-02	Accepted
3	CMLSP	1.26e+01	-1.90e+00	2.88e-02	1.67e-02	Accepted
4	GaAPADE	1.22e+01	-2.35e+00	9.38e-03	1.25e-02	Rejected
5	DE-b6e6rl.w.r.	1.21e+01	-2.48e+00	6.63e-03	1.00e-02	Rejected
6	rmalschcma	1.18e+01	-2.78e+00	2.69e-03	8.33e-03	Rejected
7	FERDE	1.12e+01	-3.49e+00	2.42e-04	7.14e-03	Rejected
8	RSDE	9.33e+00	-5.48e+00	2.16e-08	6.25e-03	Rejected
9	FWA-DM	9.07e+00	-5.77e+00	4.04e-09	5.56e-03	Rejected
10	SOO+BOBYQA	8.97e+00	-5.88e+00	2.11e-09	5.00e-03	Rejected
11	SOO	7.87e+00	-7.07e+00	7.84e-13	4.55e-03	Rejected
12	OptBees	7.82e+00	-7.12e+00	5.30e-13	4.17e-03	Rejected
13	POBL-ADE	7.05e+00	-7.95e+00	9.02e-16	3.85e-03	Rejected
14	NRGA	5.50e+00	-9.64e+00	2.84e-22	3.57e-03	Rejected
15	FCDE	4.82e+00	-1.04e+01	1.59e-25	3.33e-03	Rejected
16	b3e3pbest	3.33e+00	-1.20e+01	2.12e-33	3.13e-03	Rejected
17	LJA	2.37e+00	-1.30e+01	3.92e-39	2.94e-03	Rejected

$D = 10$ and $D = 30$ with the default parametrization suggested in (Gablonsky, 2001): $tolerance = 0.0$, $volper = 0.0$, $sigmaper = 0.0$, $maxDim = 64$, $maxDivs = 3,000$. As for $maxDepth$, we used the default value 600 for DIRECT, while we used $6,000$ for DIRECT-L.

In Table 5, we report the result of the Holm-Bonferroni procedure while the detailed results in terms of average error ± std. dev. and Wilcoxon rank-sum are shown also in this case in the Appendix, Tables B.12-B.13.

Table 5 shows that LJA ranks last in the Holm-Bonferroni procedure, and both DIRECT versions obtain statistically similar results. At first, that is precisely the result we were expecting to obtain, given how these algorithms work. Nonetheless, when the pairwise comparisons are analyzed, the results show that the deterministic method cannot perform well in all problems, getting stuck in local optima many times. In those cases, the stochastic nature of LJA allows it to escape from poor-quality regions and achieve better results.

Table 5: Holm-Bonferroni procedure (reference: DIRECT-L, Rank = 2.15e+00) for LJA against DIRECT and DIRECT-L on the CEC 2014 benchmark in 10 and 30 dimensions.

j	Optimizer	Rank	z_j	p_j	α/j	Hypothesis
1	DIRECT	1.97e+00	-1.42e+00	7.78e-02	5.00e-02	Accepted
2	LJA	1.57e+00	-4.52e+00	3.11e-06	2.50e-02	Rejected

The next analysis compares the results in terms of Wins, Ties, and Losses for each dimension (10 and 30) and problem type. Table 6 contains the values for

DIRECT and DIRECT-L compared with LJA. What is important in the table is to investigate when LJA wins. For instance, LJA is superior to DIRECT and DIRECT-L even on f_1, which is a unimodal function, where the LJA results are one order of magnitude better than those of DIRECT and DIRECT-L. The most notable results are those for the hybrid function type: LJA wins in 5 out of 6 functions. A possible explanation is that these functions are characterized by different properties for each of their function sub-components, leading the DIRECT-based algorithms to the wrong location, or making them stop earlier. This suggests that an exciting investigation for future work would be a hybrid optimizer combining LJA and DIRECT, in order to leverage the best features of the two algorithms.

Table 6: Number of Wins, Ties, and Losses (w/t/l) for DIRECT and DIRECT-L in each dimension and problem type. The reference is LJA.

D	Problem type	DIRECT	DIRECT-L
10	Unimodal Functions	2/0/1	2/0/1
	Simple Multimodal Functions	10/1/2	10/1/2
	Hybrid Function 1	1/0/5	1/0/5
	Composition Functions	5/2/1	6/1/1
30	Unimodal Functions	2/0/1	2/0/1
	Simple Multimodal Functions	13/0/0	13/0/0
	Hybrid Function 1	1/0/5	1/0/5
	Composition Functions	7/0/1	7/0/1

5.7. Real-world industrial optimization problems

For this last part of the experimentation, we considered the five industrial optimization problems described in Subsection 5.1. It is important to remember that these functions are only box-constrained. Constraint handling in LJA is a possible topic for a future investigation. On these problems, we compared the proposed LJA, the original Jaya algorithm, DIRECT, and DIRECT-L.

Table 7: Average fitness ± standard deviation and statistic comparison (reference: LJA) for LJA against Jaya, DIRECT and DIRECT-L on the five selected industrial optimization problems taken from the CEC 2011 benchmark. The boldface indicates the lowest average fitness per each problem.

	LJA	DIRECT		DIRECT-L		Jaya	
P_1	$1.46e+01 \pm 3.49e+00$	$2.11e+01 \pm 0.00e+00$	+	$2.10e+01 \pm 3.55e-15$	+	$\mathbf{1.24e+01 \pm 6.42e+00}$	=
P_2	$-5.99e+00 \pm 1.34e+00$	$5.94e+06 \pm 2.91e+07$	+	$8.47e+04 \pm 4.15e+05$	+	$\mathbf{-6.28e+00 \pm 1.57e+00}$	=
P_3	$\mathbf{1.15e-05 \pm 1.63e-19}$	$1.15e-05 \pm 3.39e-21$	+	$1.15e-05 \pm 3.39e-21$	+	$1.15e-05 \pm 1.77e-19$	=
P_4	$1.01e-02 \pm 4.94e-02$	$\mathbf{0.00e+00 \pm 0.00e+00}$	-	$\mathbf{0.00e+00 \pm 0.00e+00}$	-	$\mathbf{0.00e+00 \pm 0.00e+00}$	-
P_5	$\mathbf{0.00e+00 \pm 0.00e+00}$	$3.27e+01 \pm 3.80e+01$	+	$6.97e+01 \pm 1.35e+02$	+	$\mathbf{0.00e+00 \pm 0.00e+00}$	=

Tables 7 and 8 report respectively the results in terms of average fitness ± std. dev. and Wilcoxon rank-sum test on the five problems (it should be noted

Table 8: Holm-Bonferroni procedure (reference: Jaya, Rank = 3.20e+00) for LJA against Jaya, DIRECT and DIRECT-L on the five selected industrial optimization problems from the CEC 2011 benchmark.

j	Optimizer	Rank	z_j	p_j	α/j	Hypothesis
1	LJA	2.80e+00	-6.32e-01	2.64e-01	5.00e-02	Accepted
2	DIRECT-L	1.80e+00	-2.21e+00	1.34e-02	2.50e-02	Rejected
3	DIRECT	1.40e+00	-2.85e+00	2.21e-03	1.67e-02	Rejected

that the optimum is unknown for all problems, so the error cannot be computed in this case), and the outcome of the Holm-Bonferroni procedure.

From the tables, it can be observed that LJA is statistically on par with Jaya in four out of five problems, while Jaya is superior in the case of P_4. In general, on these problems Jaya appears slightly more efficient in terms of average fitness. In fact, LJA obtains a lower fitness than Jaya only on P_3. It is also interesting to note that the two deterministic global optimization algorithms, DIRECT and DIRECT-L, seem to perform relatively poorly on these problems, which is probably due to their extremely high multimodality. In fact, LJA results are statistically superior to both of them in four out of five problems, i.e. except P_4. One possible explanation for the slightly worse performance of LJA on P_4 is likely the lower exploitation capability due to Lévy flight, whose jumps might, in some cases, interrupt the exploitation phase. These observations are also confirmed by the Holm-Bonferroni procedure, where it can be seen that LJA ranks second after Jaya, but before both DIRECT-L and DIRECT (in this order). On the other hand, the null-hypotheses of statistical equivalence is accepted on the comparison between LJA and Jaya, while it is rejected in the comparison against DIRECT-L and DIRECT.

As a final note, such a “jump effect” could be controlled by tuning the β parameter, which was not done in our experimentation because we decided to employ the same configuration for all the experiments. Nevertheless, as we discuss below it opens the possibility of self-adapting β during the optimization process.

6. Conclusions and future works

In this paper, we proposed the Lévy flight Jaya Algorithm (LJA) for global continuous optimization. The algorithm was obtained by employing the Lévy distribution to determine the step-size and thus control the balance between exploration and exploitation. This modification, while still keeping the original simple algorithmic philosophy of Jaya, has a dramatic impact on its performance as it allows the algorithm to escape from local optima by performing occasional “jumps” in the search space. Of note, such behavior occurs without having an explicit restart mechanism, but only relying on the way random numbers are sampled from the Lévy distribution.

We assessed the performance of the proposed LJA on the CEC 2014 benchmark and compared it against that of the original Jaya, as well as the metaheuristics that participated in the CEC 2014 competition. Furthermore, we compared LJA against two deterministic DIRECT-based global optimization algorithms on the CEC 2014 benchmark functions and five industrial optimization problems. Our results showed that, thanks to Lévy flight, LJA consistently outperforms the canonical Jaya algorithm, but, apart from a few cases, it results less efficient than the most advanced metaheuristics that participated in the CEC 2014 competition. For instance, this happened on the CEC 2014 unimodal functions, where we observed that Lévy flight hampered a full exploitation due to its occasional big steps. On the other hand, the improvement w.r.t. Jaya on most of the highly multimodal functions in the CEC 2014 benchmark was quite evident, both in terms of lower fitness error and higher robustness (i.e., std. dev. across multiple runs). These superior results confirm the advantage of using Lévy flight to escape local optima.

Apart from the specific numerical results reported here, we should remark that one of the main advantages of LJA is its simplicity: LJA is much simpler to implement and tune (in fact, it inherits the two Jaya's parameters, i.e. the population size and the number of generations, but it adds a third one, β) than most of the compared algorithms from the state-of-the-art. As such, it can represent a very useful optimizer not only for practitioners – who might need to use an optimization algorithm without having sufficient time and knowledge for tuning its parameters– but also for algorithm experts who might want to embed LJA into more complex algorithmic schemes and/or hybridizing it with other algorithms. In this sense, this work paves the road to several possible future research directions. For instance, it would be interesting to further modify the Jaya scheme by using different mechanisms for handling the boundary constraints (see lines 15-18 in Algorithm 1), as suggested e.g. in (Zhang et al., 2004), or to design a mechanism to automatically adapt the β parameter and/or the population size (thus making the algorithm truly parameter-less). Furthermore, another possible research direction would be to investigate how other kinds of distribution –alternative to Lévy flight– may affect the search. Finally, it might be worth hybridizing LJA with other algorithms, such as deterministic search methods for solving Lipschitz continuous functions (Sergeyev & Kvasov, 2015; Kvasov & Sergeyev, 2015; Paulavičius et al., 2020), or including specific techniques borrowed from mathematical programming (Zhigljavsky & Zilinskas, 2007; Paulavičius & Žilinskas, 2014) for dimensionality reduction and balancing of exploitation and exploration (Lera & Sergeyev, 2018).

Acknowledgments

This paper was supported by FAPESP grant (#2017/20844-0) to V.V.M. and CAPES to V.C.S.Jr.

References

Aslan, M., Gunduz, M., & Kiran, M. S. (2019). Jayax: Jaya algorithm with xor operator for binary optimization. *Appl. Soft Comput.*, *82*, 105576.

Bekdaş, G. (2019). Optimum design of post-tensioned axially symmetric cylindrical walls using novel hybrid metaheuristic methods. *Struct. Des. Tall Spec. Build.*, *28*, e1550.

Bujok, P., Tvrdík, J., & Poláková, R. (2014). Differential evolution with rotation-invariant mutation and competing-strategies adaptation. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 2253–2258).

Caraffini, F., & Iacca, G. (2020). The SOS platform: designing, tuning and statistically benchmarking optimisation algorithms. *Mathematics*, *8*, 785.

Caraffini, F., Neri, F., Passow, B. N., & Iacca, G. (2013). Re-sampled inheritance search: high performance despite the simplicity. *Soft Comput.*, *17*, 2235–2256.

Chechkin, A. V., Metzler, R., Klafter, J., Gonchar, V. Y. et al. (2008). Introduction to the theory of Lévy flights. In *Anomalous transport: Foundations and applications* (pp. 431–451). John Wiley & Sons.

Chen, L., Zheng, Z., Liu, H. L., & Xie, S. (2014). An evolutionary algorithm based on covariance matrix leaning and searching preference for solving CEC 2014 benchmark problems. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 2672–2677).

Das, S., & Suganthan, P. N. (2010). *Problem definitions and evaluation criteria for CEC 2011 competition on testing evolutionary algorithms on real world optimization problems*. Technical Report Jadavpur University, Nanyang Technological University, Kolkata.

Elsayed, S. M., Sarker, R. A., Essam, D. L., & Hamza, N. M. (2014). Testing united multi-operator evolutionary algorithms on the CEC 2014 real-parameter numerical optimization. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 1650–1657).

Erlich, I., Rueda, J. L., Wildenhues, S., & Shewarega, F. (2014). Evaluating the mean-variance mapping optimization on the CEC 2014 test suite. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 1625–1632).

Gablonsky, J. M. (2001). *Modifications of the DIRECT Algorithm*. Ph.D. thesis North Carolina State University.

Gablonsky, J. M., & Kelley, C. T. (2001). A locally-biased form of the direct algorithm. *J. Global Optim.*, *21*, 27–37.

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 Evol. Comput.*, *37*, 58–72.

Garcia, S., Fernandez, A., Luengo, J., & Herrera, F. (2008). A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. *Soft Comput.*, *13*, 959–977.

Hare, W., Loeppky, J., & Xie, S. (2018). Methods to compare expensive stochastic optimization algorithms with random restarts. *J. Global Optim.*, *72*, 781–801.

Hariya, Y., Kurihara, T., Shindo, T., & Jin'no, K. (2015). Lévy flight PSO. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 2678–2684).

Heidari, A. A., & Pahlavani, P. (2017). An efficient modified grey wolf optimizer with lévy flight for optimization tasks. *Appl. Soft Comput.*, *60*, 115–134.

Hintze, J. L., & Nelson, R. D. (1998). Violin plots: a box plot-density trace synergism. *Am. Stat.*, *52*, 181–184.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scand. J. Stat.*, *6*, 65–70.

Hu, Z., Bao, Y., & Xiong, T. (2014). Partial opposition-based adaptive differential evolution algorithms: Evaluation on the CEC 2014 benchmark set for real-parameter optimization. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 2259–2265).

Iacca, G., Neri, F., Mininno, E., Ong, Y.-S., & Lim, M.-H. (2012). Ockham's razor in memetic computing: Three stage optimal memetic exploration. *Inform. Sciences*, *188*, 17–43.

Jones, D. R., Perttunen, C. D., & Stuckman, B. E. (1993). Lipschitzian optimization without the Lipschitz constant. *J. Optim. Theory App.*, *79*, 157–181.

Kalantzis, G., Shang, C., Lei, Y., & Leventouri, T. (2016). Investigations of a GPU-based Lévy-firefly algorithm for constrained optimization of radiation therapy treatment planning. *Swarm Evol. Comput.*, *26*, 191–201.

Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *J. Global Optim.*, *39*, 459–471.

Kennedy, J. (2010). Particle swarm optimization. In C. Sammut, & G. I. Webb (Eds.), *Encyclopedia of Machine Learning* (pp. 760–766). Springer.

Kvasov, D. E., & Mukhametzhanov, M. S. (2018). Metaheuristic vs. deterministic global optimization algorithms: The univariate case. *Appl. Math. Comput.*, *318*, 245–259.

Kvasov, D. E., & Sergeyev, Y. D. (2015). Deterministic approaches for solving practical black-box global optimization problems. *Adv. Eng. Softw.*, *80*, 58–66.

Lera, D., & Sergeyev, Y. D. (2018). GOSH: derivative-free global optimization using multi-dimensional space-filling curves. *J. Global Optim.*, *71*, 193–211.

Li, Z., Shang, Z., Qu, B. Y., & Liang, J. J. (2014). Differential evolution strategy based on the constraint of fitness values classification. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 1454–1460).

Liang, J. J., Qu, B. Y., & Suganthan, P. N. (2013). *Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization*. Technical Report Zhengzhou University and Nanyang Technological University.

Maia, R. D., de Castro, L. N., & Caminhas, W. M. (2014). Real-parameter optimization with OptBees. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 2649–2655).

Majumder, A., & Laha, D. (2016). A new cuckoo search algorithm for 2-machine robotic cell scheduling problem with sequence-dependent setup times. *Swarm Evol. Comput.*, *28*, 131–143.

Mallipeddi, R., Wu, G., Lee, M., & Suganthan, P. N. (2014). Gaussian adaptation based parameter adaptation for differential evolution. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 1760–1767).

de Melo, V. V., & Iacca, G. (2014). A modified covariance matrix adaptation evolution strategy with adaptive penalty function and restart for constrained optimization. *Expert Syst. Appl.*, *41*, 7077–7094.

Molina, D., Lacroix, B., & Herrera, F. (2014). Influence of regions on the memetic algorithm for the CEC 2014 special session on real-parameter single objective optimisation. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 1633–1640).

Neri, F., Iacca, G., & Mininno, E. (2013). Compact optimization. In *Handbook of Optimization: From Classical to Modern Approach* (pp. 337–364). Springer.

Nguyen, T. T., & Vo, D. N. (2017). Modified cuckoo search algorithm for multiobjective short-term hydrothermal scheduling. *Swarm Evol. Comput.*, *37*, 73–89.

Opara, K., & Arabas, J. (2018). Comparison of mutation strategies in differential evolution – a probabilistic perspective. *Swarm Evol. Comput.*, *39*, 53–69.

Pandey, N., Verma, O. P., & Kumar, A. (2019). Nature inspired power optimization in smartphones. *Swarm Evol. Comput.*, *44*, 470–479.

Paulavičius, R., Sergeyev, Y. D., Kvasov, D. E., & Žilinskas, J. (2020). Globally-biased BIRECT algorithm with local accelerators for expensive global optimization. *Expert Syst. Appl.*, *144*, 113052.

Paulavičius, R., & Žilinskas, J. (2014). *Simplicial global optimization*. Springer.

Piotrowski, A. P., & Napiorkowski, J. J. (2018). Some metaheuristics should be simplified. *Inform. Sciences*, *427*, 32–62.

Poláková, R., Tvrdík, J., & Bujok, P. (2014). Controlled restart in differential evolution applied to CEC 2014 benchmark functions. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 2230–2236).

Preux, P., Munos, R., & Valko, M. (2014). Bandits attack function optimization. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 2245–2252).

Qu, B. Y., Liang, J. J., Xiao, J. M., & Shang, Z. G. (2014). Memetic differential evolution based on fitness Euclidean-distance ratio. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 2266–2273).

Rao, R. V. (2016). Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *Int. J. Ind. Eng. Comput.*, *7*, 19–34.

Rao, R. V., & Saroj, A. (2017). A self-adaptive multi-population based Jaya algorithm for engineering optimization. *Swarm Evol. Comput.*, *37*, 1–26.

Rao, R. V., Savsani, V. J., & Vakharia, D. P. (2011). Teaching-learning-based optimization: A novel method for constrained mechanical design optimization problems. *Comput. Aided Des.*, *43*, 303–315.

Sergeyev, Y. D., & Kvasov, D. E. (2015). A deterministic global optimization using smooth diagonal auxiliary functions. *Commun. Nonlinear Sci.*, *21*, 99–111.

Sergeyev, Y. D., Kvasov, D. E., & Mukhametzhanov, M. S. (2018). On the efficiency of nature-inspired metaheuristics in expensive global optimization with limited budget. *Sci. Rep.*, *8*, 1–9.

Sharma, H., Bansal, J. C., Arya, K. V., & Yang, X.-S. (2016). Lévy flight artificial bee colony algorithm. *Int. J. Syst. Sci.*, *47*, 2652–2670.

Tanabe, R., & Fukunaga, A. S. (2014). Improving the search performance of SHADE using linear population size reduction. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 1658–1665).

Tighzert, L., Fonlupt, C., & Mendil, B. (2018). A set of new compact firefly algorithms. *Swarm Evol. Comput.*, *40*, 92–115.

Tran, T., Nguyen, T. T., & Nguyen, H. L. (2014). Global optimization using Lévy flights. *CoRR*, *abs/1407.5739*.

Viswanathan, G. M., Buldyrev, S. V., Havlin, S., Da Luz, M., Raposo, E., & Stanley, H. E. (1999). Optimizing the success of random searches. *Nature*, *401*, 911.

Wang, G., Guo, L., & Gandomi, A. H. (2013). Lévy-flight krill herd algorithm. *Math. Probl. Eng.*, .

Wang, R., Nguyen, T. T., Li, C., Jenkinson, I., Yang, Z., & Kavakeb, S. (2019). Optimising discrete dynamic berth allocations in seaports using a Lévy flight based meta-heuristic. *Swarm Evol. Comput.*, *44*, 1003–1017.

Wang, S., Rao, R. V., Chen, P., Zhang, Y., Liu, A., & Wei, L. (2017). Abnormal breast detection in mammogram images by feed-forward neural network trained by Jaya algorithm. *Fundam. Inform.*, *151*, 191–211.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, *1*, 80–83.

Xu, C., Huang, H., & Ye, S. (2014). A differential evolution with replacement strategy for real-parameter numerical optimization. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 1617–1624).

Yang, X.-S. (2010a). Firefly algorithm, Lévy flights and global optimization. In M. Bramer, R. Ellis, & M. Petridis (Eds.), *Research and Development in Intelligent Systems XXVI* (pp. 209–218). Springer.

Yang, X.-S. (2010b). Firefly algorithm, stochastic test functions and design optimisation. *Int. J. Bio-Inspir. Com.*, *2*, 78–84.

Yang, X.-S. (2012). Flower pollination algorithm for global optimization. In J. Durand-Lose, & N. Jonoska (Eds.), *International Conference on Unconventional Computation and Natural Computation (UCNC)* (pp. 240–249).

Yang, X.-S., & Deb, S. (2009). Cuckoo search via Lévy flights. In *World Congress on Nature Biologically Inspired Computing (NaBIC)* (pp. 210–214).

Yashesh, D., Deb, K., & Bandaru, S. (2014). Non-uniform mapping in real-coded genetic algorithms. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 2237–2244).

Yu, C., Kelley, L., Zheng, S., & Tan, Y. (2014). Fireworks algorithm with differential mutation for solving the CEC 2014 competition problems. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 3238–3245).

Zhang, W.-J., Xie, X.-F., & Bi, D.-C. (2004). Handling boundary constraints for numerical optimization by particle swarm flying in periodic search space. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 2307–2311). volume 2.

Zhang, Y., Yang, X., Cattani, C., Rao, R. V., Wang, S., & Phillips, P. (2016). Tea category identification using a novel fractional fourier entropy and Jaya algorithm. *Entropy*, *18*.

Zhigljavsky, A., & Zilinskas, A. (2007). *Stochastic global optimization* volume 9. Springer Science & Business Media.

Appendix A. Holm-Bonferroni procedure

The sequentially rejective Holm-Bonferroni procedure (Garcia et al., 2008; Holm, 1979) is a multiple comparisons test that applies the to Bonferroni correction to counteract the fact that the more null-hypotheses are checked, the higher the probability of obtaining Type I errors (false positives). The Holm–Bonferroni therefore adjusts the rejection criterion for each of the individual hypotheses. In a nutshell, the procedure consists of the following: considering the error obtained by all the algorithms at the end of the computational budget, averaged across all the available runs, for each problem a score R_i is assigned to each algorithm, for $i = 1, 2, \dots, N_A$ (where N_A is the number of algorithms under analysis), being N_A the score of the algorithm displaying the best performance on that problem, $N_A - 1$ the score of the second-best, and so on. The algorithm displaying the worst performance scores 1. These scores are then averaged, for each algorithm, over the whole set of test problems. The algorithms are sorted based on these average scores. Indicating with R_0 the Rank of the algorithm displaying the highest average score, which is considered as the *reference algorithm*, and with R_j for $j = 1, 2, \dots, N_A - 1$ the Ranks of the remaining $N_A - 1$ algorithms, the values z_j are calculated as:

$$z_j = \frac{R_j - R_0}{\sqrt{\frac{N_A(N_A+1)}{6N_{TP}}}}$$

where N_{TP} is the number of test problems in consideration. By means of the z_j values, the corresponding cumulative normal distribution values p_j are derived. Finally, the p_j values are compared to the corresponding δ/j where δ is the confidence interval, set to 0.05: if $p_j < \delta/j$, the null-hypothesis (that the reference algorithm has the same performance as the j-th algorithm) is rejected, otherwise is accepted as well as all the subsequent tests.

Appendix B. Detailed results on CEC 2014 benchmark

Table B.9: Average error ± standard deviation and statistic comparison (reference: LJA ($\beta = 1.8$)) for LJA ($\beta = 1.8$) against LJA ($\beta = 1.8$) and LJA ($\beta = 2.0$) on the CEC 2014 benchmark in 10 dimensions. The boldface indicates the lowest average error per each problem.

	LJA ($\beta = 1.8$)	LJA ($\beta = 1.6$)		LJA ($\beta = 2.0$)	
f_1	$\mathbf{6.99e+05 \pm 2.56e+05}$	$1.56e+06 \pm 6.27e+05$	+	$4.13e+08 \pm 2.74e+08$	+
f_2	$\mathbf{4.06e+07 \pm 1.36e+07}$	$2.05e+08 \pm 6.26e+07$	+	$1.35e+10 \pm 4.02e+09$	+
f_3	$\mathbf{8.81e+03 \pm 3.29e+03}$	$1.88e+04 \pm 4.81e+03$	+	$4.03e+06 \pm 6.74e+06$	+
f_4	$\mathbf{2.43e+01 \pm 9.43e+00}$	$2.97e+01 \pm 8.11e+00$	+	$3.38e+03 \pm 1.68e+03$	+
f_5	$2.03e+01 \pm 7.94e-02$	$\mathbf{2.03e+01 \pm 8.78e-02}$	=	$2.11e+01 \pm 1.75e-01$	+
f_6	$\mathbf{6.77e+00 \pm 7.64e-01}$	$7.69e+00 \pm 6.03e-01$	+	$1.40e+01 \pm 1.14e+00$	+
f_7	$\mathbf{7.03e-01 \pm 7.65e-02}$	$8.84e-01 \pm 9.77e-02$	+	$2.43e+02 \pm 7.97e+01$	+
f_8	$\mathbf{3.18e+01 \pm 4.01e+00}$	$3.95e+01 \pm 6.68e+00$	+	$1.32e+02 \pm 2.11e+01$	+
f_9	$\mathbf{3.59e+01 \pm 5.24e+00}$	$4.61e+01 \pm 6.41e+00$	+	$1.37e+02 \pm 2.09e+01$	+
f_{10}	$5.00e+02 \pm 1.74e+02$	$\mathbf{4.23e+02 \pm 1.78e+02}$	-	$2.57e+03 \pm 2.64e+02$	+
f_{11}	$\mathbf{1.12e+03 \pm 1.70e+02}$	$1.22e+03 \pm 1.49e+02$	+	$2.60e+03 \pm 2.63e+02$	+
f_{12}	$1.06e+00 \pm 2.07e-01$	$\mathbf{9.76e-01 \pm 2.45e-01}$	=	$4.90e+00 \pm 1.28e+00$	+
f_{13}	$\mathbf{3.37e-01 \pm 4.97e-02}$	$4.97e-01 \pm 8.71e-02$	+	$5.46e+00 \pm 1.07e+00$	+
f_{14}	$\mathbf{1.36e-01 \pm 3.06e-02}$	$2.26e-01 \pm 1.09e-01$	+	$6.41e+01 \pm 1.62e+01$	+
f_{15}	$\mathbf{2.96e+00 \pm 5.19e-01}$	$4.23e+00 \pm 6.23e-01$	+	$2.80e+05 \pm 2.29e+05$	+
f_{16}	$\mathbf{3.34e+00 \pm 1.24e-01}$	$3.53e+00 \pm 1.49e-01$	+	$4.53e+00 \pm 1.48e-01$	+
f_{17}	$\mathbf{8.54e+03 \pm 6.77e+03}$	$2.48e+04 \pm 1.55e+04$	+	$1.91e+07 \pm 1.96e+07$	+
f_{18}	$\mathbf{4.03e+03 \pm 3.30e+03}$	$7.49e+03 \pm 7.14e+03$	+	$1.95e+08 \pm 2.02e+08$	+
f_{19}	$\mathbf{3.37e+00 \pm 4.03e-01}$	$4.24e+00 \pm 8.15e-01$	+	$1.08e+02 \pm 6.88e+01$	+
f_{20}	$\mathbf{8.13e+02 \pm 5.63e+02}$	$2.80e+03 \pm 4.29e+03$	+	$1.67e+07 \pm 2.68e+07$	+
f_{21}	$\mathbf{8.97e+02 \pm 1.85e+02}$	$1.55e+03 \pm 5.25e+02$	+	$1.11e+07 \pm 1.32e+07$	+
f_{22}	$4.01e+01 \pm 6.78e+00$	$\mathbf{3.77e+01 \pm 7.64e+00}$	=	$6.99e+02 \pm 1.82e+02$	+
f_{23}	$\mathbf{3.29e+02 \pm 1.04e-02}$	$3.29e+02 \pm 7.43e-02$	+	$6.71e+02 \pm 1.41e+02$	+
f_{24}	$\mathbf{1.41e+02 \pm 4.93e+00}$	$1.52e+02 \pm 7.16e+00$	+	$2.51e+02 \pm 1.64e+01$	+
f_{25}	$\mathbf{1.84e+02 \pm 1.91e+01}$	$1.98e+02 \pm 1.03e+01$	+	$2.26e+02 \pm 1.15e+01$	+
f_{26}	$\mathbf{1.00e+02 \pm 8.62e-02}$	$1.00e+02 \pm 1.20e-01$	+	$1.10e+02 \pm 1.04e+01$	+
f_{27}	$\mathbf{2.51e+02 \pm 1.56e+02}$	$2.83e+02 \pm 1.47e+02$	=	$7.12e+02 \pm 1.22e+02$	+
f_{28}	$4.24e+02 \pm 4.91e+01$	$\mathbf{4.22e+02 \pm 4.38e+01}$	=	$1.82e+03 \pm 3.62e+02$	+
f_{29}	$\mathbf{6.84e+04 \pm 3.35e+05}$	$6.86e+04 \pm 3.35e+05$	+	$3.69e+07 \pm 2.74e+07$	+
f_{30}	$\mathbf{6.76e+02 \pm 1.57e+02}$	$8.00e+02 \pm 1.97e+02$	+	$6.69e+05 \pm 7.61e+05$	+

Table B.10: Average error ± standard deviation and statistic comparison (reference: LJA) for LJA against Jaya on the CEC 2014 benchmark in 10 dimensions. The boldface indicates the lowest average error per each problem.

	LJA	Jaya	
f_1	$\mathbf{6.99e+05 \pm 2.56e+05}$	$9.13e+05 \pm 4.02e+05$	+
f_2	$\mathbf{4.06e+07 \pm 1.36e+07}$	$1.19e+08 \pm 5.04e+07$	+
f_3	$\mathbf{8.81e+03 \pm 3.29e+03}$	$1.23e+04 \pm 4.91e+03$	+
f_4	$\mathbf{2.43e+01 \pm 9.43e+00}$	$3.04e+01 \pm 8.53e+00$	+
f_5	$\mathbf{2.03e+01 \pm 7.94e-02}$	$2.03e+01 \pm 7.70e-02$	=
f_6	$6.77e+00 \pm 7.64e-01$	$\mathbf{6.66e+00 \pm 9.37e-01}$	=
f_7	$\mathbf{7.03e-01 \pm 7.65e-02}$	$7.39e-01 \pm 9.16e-02$	+
f_8	$\mathbf{3.18e+01 \pm 4.01e+00}$	$3.48e+01 \pm 4.04e+00$	+
f_9	$\mathbf{3.59e+01 \pm 5.24e+00}$	$3.90e+01 \pm 5.58e+00$	+
f_{10}	$5.00e+02 \pm 1.74e+02$	$\mathbf{3.74e+02 \pm 2.21e+02}$	-
f_{11}	$\mathbf{1.12e+03 \pm 1.70e+02}$	$1.27e+03 \pm 1.59e+02$	+
f_{12}	$1.06e+00 \pm 2.07e-01$	$\mathbf{1.06e+00 \pm 2.21e-01}$	=
f_{13}	$\mathbf{3.37e-01 \pm 4.97e-02}$	$3.98e-01 \pm 6.31e-02$	+
f_{14}	$\mathbf{1.36e-01 \pm 3.06e-02}$	$1.59e-01 \pm 3.90e-02$	+
f_{15}	$\mathbf{2.96e+00 \pm 5.19e-01}$	$3.36e+00 \pm 5.23e-01$	+
f_{16}	$\mathbf{3.34e+00 \pm 1.24e-01}$	$3.46e+00 \pm 1.15e-01$	+
f_{17}	$\mathbf{8.54e+03 \pm 6.77e+03}$	$2.30e+04 \pm 1.61e+04$	+
f_{18}	$\mathbf{4.03e+03 \pm 3.30e+03}$	$7.77e+03 \pm 8.05e+03$	+
f_{19}	$\mathbf{3.37e+00 \pm 4.03e-01}$	$3.94e+00 \pm 8.27e-01$	+
f_{20}	$\mathbf{8.13e+02 \pm 5.63e+02}$	$1.23e+03 \pm 8.69e+02$	+
f_{21}	$\mathbf{8.97e+02 \pm 1.85e+02}$	$1.18e+03 \pm 3.21e+02$	+
f_{22}	$4.01e+01 \pm 6.78e+00$	$\mathbf{3.23e+01 \pm 5.63e+00}$	-
f_{23}	$\mathbf{3.29e+02 \pm 1.04e-02}$	$3.29e+02 \pm 2.94e-02$	+
f_{24}	$\mathbf{1.41e+02 \pm 4.93e+00}$	$1.44e+02 \pm 4.73e+00$	+
f_{25}	$\mathbf{1.84e+02 \pm 1.91e+01}$	$1.99e+02 \pm 9.45e+00$	+
f_{26}	$\mathbf{1.00e+02 \pm 8.62e-02}$	$1.00e+02 \pm 6.29e-02$	+
f_{27}	$\mathbf{2.51e+02 \pm 1.56e+02}$	$2.83e+02 \pm 1.72e+02$	=
f_{28}	$\mathbf{4.24e+02 \pm 4.91e+01}$	$4.39e+02 \pm 5.94e+01$	=
f_{29}	$\mathbf{6.84e+04 \pm 3.35e+05}$	$1.70e+05 \pm 5.13e+05$	=
f_{30}	$\mathbf{6.76e+02 \pm 1.57e+02}$	$7.68e+02 \pm 2.15e+02$	+

Table B.11: Average error ± standard deviation and statistic comparison (reference: LJA) for LJA against Jaya on the CEC 2014 benchmark in 30 dimensions. The boldface indicates the lowest average error per each problem.

	LJA	Jaya	
f_1	$\mathbf{6.31e+07 \pm 1.87e+07}$	$8.47e+07 \pm 2.25e+07$	$+$
f_2	$\mathbf{4.77e+09 \pm 6.03e+08}$	$7.55e+09 \pm 1.18e+09$	$+$
f_3	$\mathbf{6.91e+04 \pm 1.07e+04}$	$8.10e+04 \pm 1.24e+04$	$+$
f_4	$\mathbf{4.08e+02 \pm 5.38e+01}$	$5.69e+02 \pm 1.29e+02$	$+$
f_5	$2.09e+01 \pm 4.97e-02$	$\mathbf{2.09e+01 \pm 4.71e-02}$	$=$
f_6	$\mathbf{3.39e+01 \pm 1.29e+00}$	$3.48e+01 \pm 1.77e+00$	$+$
f_7	$\mathbf{1.58e+01 \pm 2.80e+00}$	$2.59e+01 \pm 5.83e+00$	$+$
f_8	$\mathbf{2.24e+02 \pm 9.93e+00}$	$2.29e+02 \pm 1.34e+01$	$+$
f_9	$\mathbf{2.61e+02 \pm 1.47e+01}$	$2.64e+02 \pm 1.85e+01$	$=$
f_{10}	$5.68e+03 \pm 3.95e+02$	$\mathbf{5.59e+03 \pm 4.35e+02}$	$=$
f_{11}	$\mathbf{6.88e+03 \pm 3.12e+02}$	$6.91e+03 \pm 3.16e+02$	$=$
f_{12}	$2.49e+00 \pm 2.73e-01$	$\mathbf{2.44e+00 \pm 2.63e-01}$	$=$
f_{13}	$\mathbf{1.08e+00 \pm 1.19e-01}$	$1.80e+00 \pm 3.59e-01$	$+$
f_{14}	$\mathbf{4.33e+00 \pm 1.70e+00}$	$1.23e+01 \pm 1.82e+00$	$+$
f_{15}	$\mathbf{5.05e+01 \pm 9.36e+00}$	$8.39e+01 \pm 7.03e+01$	$+$
f_{16}	$\mathbf{1.28e+01 \pm 1.78e-01}$	$1.30e+01 \pm 1.70e-01$	$+$
f_{17}	$\mathbf{2.63e+06 \pm 9.76e+05}$	$4.69e+06 \pm 1.36e+06$	$+$
f_{18}	$\mathbf{1.26e+07 \pm 1.06e+07}$	$2.97e+07 \pm 3.19e+07$	$+$
f_{19}	$\mathbf{3.78e+01 \pm 3.46e+01}$	$3.85e+01 \pm 1.91e+01$	$+$
f_{20}	$\mathbf{9.92e+03 \pm 3.69e+03}$	$1.16e+04 \pm 3.70e+03$	$+$
f_{21}	$\mathbf{6.94e+05 \pm 2.03e+05}$	$9.02e+05 \pm 3.08e+05$	$+$
f_{22}	$\mathbf{5.47e+02 \pm 1.05e+02}$	$6.45e+02 \pm 1.38e+02$	$+$
f_{23}	$\mathbf{3.43e+02 \pm 3.41e+00}$	$3.57e+02 \pm 6.69e+00$	$+$
f_{24}	$\mathbf{2.57e+02 \pm 4.04e+00}$	$2.61e+02 \pm 4.74e+00$	$+$
f_{25}	$\mathbf{2.16e+02 \pm 2.58e+00}$	$2.23e+02 \pm 5.18e+00$	$+$
f_{26}	$\mathbf{1.01e+02 \pm 1.02e-01}$	$1.01e+02 \pm 1.70e-01$	$+$
f_{27}	$\mathbf{9.86e+02 \pm 2.48e+02}$	$1.08e+03 \pm 1.96e+02$	$+$
f_{28}	$\mathbf{1.13e+03 \pm 6.63e+01}$	$1.21e+03 \pm 1.70e+02$	$+$
f_{29}	$\mathbf{9.82e+05 \pm 2.07e+06}$	$1.57e+06 \pm 3.06e+06$	$+$
f_{30}	$\mathbf{1.09e+04 \pm 4.24e+03}$	$1.56e+04 \pm 6.41e+03$	$+$

Table B.12: Average error ± standard deviation and statistic comparison (reference: LJA) for LJA against DIRECT and DIRECT-L on the CEC 2014 benchmark in 10 dimensions. The boldface indicates the lowest average error per each problem.

	LJA	DIRECT		DIRECT-L	
f_1	**6.99e+05 ± 2.56e+05**	7.26e+06 ± 6.52e−09	+	7.32e+06 ± 9.31e−10	+
f_2	4.06e+07 ± 1.36e+07	**8.88e+03 ± 9.09e−12**	-	1.64e+07 ± 1.49e−08	-
f_3	8.81e+03 ± 3.29e+03	6.65e+03 ± 3.64e−12	-	**6.65e+03 ± 6.37e−12**	-
f_4	2.43e+01 ± 9.43e+00	**2.07e−01 ± 1.71e−13**	-	1.69e+00 ± 0.00e+00	-
f_5	2.03e+01 ± 7.94e−02	**2.00e+01 ± 3.41e−13**	-	2.00e+01 ± 3.41e−13	-
f_6	6.77e+00 ± 7.64e−01	**3.90e+00 ± 3.41e−13**	-	3.90e+00 ± 3.41e−13	-
f_7	7.03e−01 ± 7.65e−02	2.56e−01 ± 4.55e−13	-	**1.19e−01 ± 2.27e−13**	-
f_8	**3.18e+01 ± 4.01e+00**	3.18e+01 ± 1.14e−13	=	3.18e+01 ± 1.14e−13	=
f_9	3.59e+01 ± 5.24e+00	**3.08e+01 ± 9.09e−13**	-	3.18e+01 ± 6.82e−13	-
f_{10}	**5.00e+02 ± 1.74e+02**	6.04e+02 ± 6.82e−13	+	6.04e+02 ± 6.82e−13	+
f_{11}	1.12e+03 ± 1.70e+02	**1.04e+03 ± 2.27e−12**	-	1.06e+03 ± 2.27e−12	-
f_{12}	1.06e+00 ± 2.07e−01	3.16e−01 ± 0.00e+00	-	**2.67e−01 ± 4.55e−13**	-
f_{13}	3.37e−01 ± 4.97e−02	**1.88e−01 ± 4.55e−13**	-	1.88e−01 ± 4.55e−13	-
f_{14}	**1.36e−01 ± 3.06e−02**	2.25e−01 ± 2.27e−13	+	1.96e−01 ± 4.55e−13	+
f_{15}	2.96e+00 ± 5.19e−01	1.54e+00 ± 2.27e−13	-	**1.54e+00 ± 9.09e−13**	-
f_{16}	3.34e+00 ± 1.24e−01	3.11e+00 ± 1.59e−12	-	**3.11e+00 ± 0.00e+00**	-
f_{17}	**8.54e+03 ± 6.77e+03**	5.60e+05 ± 4.66e−10	+	5.60e+05 ± 3.49e−10	+
f_{18}	**4.03e+03 ± 3.30e+03**	1.29e+04 ± 1.27e−11	+	1.29e+04 ± 7.28e−12	+
f_{19}	3.37e+00 ± 4.03e−01	3.03e+00 ± 1.59e−12	-	**3.03e+00 ± 1.36e−12**	-
f_{20}	**8.13e+02 ± 5.63e+02**	9.06e+03 ± 9.09e−12	+	9.06e+03 ± 3.64e−12	+
f_{21}	**8.97e+02 ± 1.85e+02**	2.46e+04 ± 2.18e−11	+	2.46e+04 ± 1.46e−11	+
f_{22}	**4.01e+01 ± 6.78e+00**	5.18e+01 ± 1.36e−12	+	5.18e+01 ± 1.36e−12	+
f_{23}	3.29e+02 ± 1.04e−02	**2.00e+02 ± 0.00e+00**	-	2.00e+02 ± 0.00e+00	-
f_{24}	**1.41e+02 ± 4.93e+00**	1.43e+02 ± 1.36e−12	+	1.43e+02 ± 2.27e−12	+
f_{25}	1.84e+02 ± 1.91e+01	1.57e+02 ± 1.82e−12	-	**1.52e+02 ± 0.00e+00**	-
f_{26}	1.00e+02 ± 8.62e−02	1.00e+02 ± 1.36e−12	=	**1.00e+02 ± 1.82e−12**	-
f_{27}	2.51e+02 ± 1.56e+02	**2.00e+02 ± 0.00e+00**	=	2.00e+02 ± 0.00e+00	=
f_{28}	4.24e+02 ± 4.91e+01	**2.00e+02 ± 0.00e+00**	-	2.00e+02 ± 0.00e+00	-
f_{29}	6.84e+04 ± 3.35e+05	**2.00e+02 ± 0.00e+00**	-	2.00e+02 ± 0.00e+00	-
f_{30}	6.76e+02 ± 1.57e+02	**2.00e+02 ± 0.00e+00**	-	2.00e+02 ± 0.00e+00	-

Table B.13: Average error ± standard deviation and statistic comparison (reference: LJA) for LJA against DIRECT and DIRECT-L on the CEC 2014 benchmark in 30 dimensions. The boldface indicates the lowest average error per each problem.

	LJA	DIRECT		DIRECT-L	
f_1	**6.31e+07 ± 1.87e+07**	1.87e+08 ± 2.98e−08	+	1.99e+08 ± 1.19e−07	+
f_2	4.77e+09 ± 6.03e+08	3.62e+08 ± 2.98e−07	-	**3.62e+08 ± 5.96e−08**	-
f_3	6.91e+04 ± 1.07e+04	**1.10e+04 ± 9.09e−12**	-	1.10e+04 ± 9.09e−12	-
f_4	4.08e+02 ± 5.38e+01	1.62e+02 ± 3.41e−13	-	**8.90e+01 ± 2.27e−13**	-
f_5	2.09e+01 ± 4.97e−02	**2.00e+01 ± 3.41e−13**	-	2.00e+01 ± 3.41e−13	-
f_6	3.39e+01 ± 1.29e+00	9.00e+00 ± 6.82e−13	-	**9.00e+00 ± 5.68e−13**	-
f_7	1.58e+01 ± 2.80e+00	9.33e−01 ± 5.68e−13	-	**9.33e−01 ± 4.55e−13**	-
f_8	2.24e+02 ± 9.93e+00	9.55e+01 ± 4.55e−13	-	**9.55e+01 ± 5.68e−13**	-
f_9	2.61e+02 ± 1.47e+01	1.32e+02 ± 6.82e−13	-	**1.32e+02 ± 6.82e−13**	-
f_{10}	5.68e+03 ± 3.95e+02	3.17e+03 ± 2.73e−12	-	**3.17e+03 ± 4.55e−12**	-
f_{11}	6.88e+03 ± 3.12e+02	**3.98e+03 ± 0.00e+00**	-	3.98e+03 ± 2.73e−12	-
f_{12}	2.49e+00 ± 2.73e−01	1.92e−01 ± 6.82e−13	-	**1.92e−01 ± 1.36e−12**	-
f_{13}	1.08e+00 ± 1.19e−01	5.16e−01 ± 6.82e−13	-	**5.16e−01 ± 9.09e−13**	-
f_{14}	4.33e+00 ± 1.70e+00	2.81e−01 ± 4.55e−13	-	**2.81e−01 ± 9.09e−13**	-
f_{15}	5.05e+01 ± 9.36e+00	**2.11e+01 ± 2.27e−13**	-	3.91e+01 ± 0.00e+00	-
f_{16}	1.28e+01 ± 1.78e−01	**1.08e+01 ± 0.00e+00**	-	1.22e+01 ± 1.36e−12	-
f_{17}	**2.63e+06 ± 9.76e+05**	1.73e+07 ± 1.12e−08	+	1.73e+07 ± 0.00e+00	+
f_{18}	1.26e+07 ± 1.06e+07	3.96e+03 ± 2.73e−12	-	**3.96e+03 ± 5.46e−12**	-
f_{19}	**3.78e+01 ± 3.46e+01**	4.77e+01 ± 9.09e−13	+	4.77e+01 ± 9.09e−13	+
f_{20}	**9.92e+03 ± 3.69e+03**	2.97e+04 ± 2.18e−11	+	4.90e+04 ± 3.64e−11	+
f_{21}	**6.94e+05 ± 2.03e+05**	1.70e+06 ± 2.33e−10	+	4.85e+06 ± 4.66e−09	+
f_{22}	**5.47e+02 ± 1.05e+02**	1.04e+03 ± 4.55e−12	+	1.04e+03 ± 4.55e−12	+
f_{23}	3.43e+02 ± 3.41e+00	**2.00e+02 ± 0.00e+00**	-	2.00e+02 ± 0.00e+00	-
f_{24}	2.57e+02 ± 4.04e+00	**2.00e+02 ± 0.00e+00**	-	2.00e+02 ± 0.00e+00	-
f_{25}	2.16e+02 ± 2.58e+00	**2.00e+02 ± 0.00e+00**	-	2.00e+02 ± 0.00e+00	-
f_{26}	**1.01e+02 ± 1.02e−01**	2.00e+02 ± 0.00e+00	+	2.00e+02 ± 0.00e+00	+
f_{27}	9.86e+02 ± 2.48e+02	**2.00e+02 ± 0.00e+00**	-	2.00e+02 ± 0.00e+00	-
f_{28}	1.13e+03 ± 6.63e+01	**2.00e+02 ± 0.00e+00**	-	2.00e+02 ± 0.00e+00	-
f_{29}	9.82e+05 ± 2.07e+06	**2.00e+02 ± 0.00e+00**	-	2.00e+02 ± 0.00e+00	-
f_{30}	1.09e+04 ± 4.24e+03	**2.00e+02 ± 0.00e+00**	-	2.00e+02 ± 0.00e+00	-

Table B.14: Average error ± standard deviation and statistic comparison for LJA (reference) versus CEC2014 competitors in 10 dimensions (part 1).

	LJA	rmalschcma		DE-b6e6rl.w.r.		MVMO		UMOEAs		POBL-ADE		CMLSP		b3e3pbest		OptBees		L-SHADE	
f_1	6.99e+05 ± 2.56e+05	**0.00e+00 ± 0.00e+00**	-	0.00e+00 ± 0.00e+00	-	4.95e−04 ± 7.28e−04	-	0.00e+00 ± 0.00e+00	-	1.62e+04 ± 3.72e+04	-	1.77e−07 ± 5.98e−07	-	2.63e+06 ± 9.20e+06	-	7.84e+02 ± 6.89e+02	-	0.00e+00 ± 0.00e+00	-
f_2	4.06e+07 ± 1.36e+07	**0.00e+00 ± 0.00e+00**	-	0.00e+00 ± 0.00e+00	-	7.10e−09 ± 2.26e−08	-	0.00e+00 ± 0.00e+00	-	2.27e+03 ± 3.24e+03	-	0.00e+00 ± 5.63e−15	-	2.24e+08 ± 7.38e+08	-	9.88e−03 ± 6.34e−02	-	0.00e+00 ± 0.00e+00	-
f_3	8.81e+03 ± 3.29e+03	1.03e−07 ± 7.25e−07	-	**0.00e+00 ± 0.00e+00**	-	9.85e−11 ± 4.36e−10	-	0.00e+00 ± 0.00e+00	-	5.74e−04 ± 1.89e−03	-	1.06e−04 ± 1.81e−04	-	4.31e+03 ± 1.09e+04	-	9.21e−01 ± 2.91e+00	-	0.00e+00 ± 0.00e+00	-
f_4	2.43e+01 ± 9.43e+00	8.50e−02 ± 6.01e−01	-	1.13e+00 ± 4.93e+00	-	9.55e+00 ± 1.48e+01	-	**0.00e+00 ± 0.00e+00**	-	2.55e+01 ± 1.40e+01	=	0.00e+00 ± 1.38e−14	-	5.25e+01 ± 5.95e+01	+	2.69e+00 ± 2.85e+00	-	2.94e+01 ± 1.25e+01	+
f_5	2.03e+01 ± 7.94e−02	**1.37e+01 ± 9.24e+00**	-	1.85e+01 ± 4.38e+00	-	1.66e+01 ± 7.26e+00	-	1.68e+01 ± 7.11e+00	-	1.91e+01 ± 3.56e+00	-	1.69e+01 ± 7.27e+00	-	2.02e+01 ± 1.86e−01	-	2.00e+01 ± 1.27e−04	-	1.41e+01 ± 8.68e+00	-
f_6	6.77e+00 ± 7.64e−01	1.48e−04 ± 3.87e−04	-	**0.00e+00 ± 0.00e+00**	-	3.44e−03 ± 1.05e−02	-	0.00e+00 ± 0.00e+00	-	1.04e+00 ± 7.75e−01	-	6.20e−02 ± 2.91e−01	-	2.79e+00 ± 3.60e+00	-	3.02e+00 ± 1.27e+00	-	1.75e−02 ± 1.24e−01	-
f_7	7.03e−01 ± 7.65e−02	**0.00e+00 ± 0.00e+00**	-	1.69e−02 ± 1.28e−02	-	1.86e−02 ± 1.43e−02	-	0.00e+00 ± 0.00e+00	-	1.63e−01 ± 1.92e−01	-	0.00e+00 ± 0.00e+00	-	3.31e+00 ± 9.34e+00	-	1.56e−01 ± 1.39e−01	-	3.04e−03 ± 6.44e−03	-
f_8	3.18e+01 ± 4.01e+00	**0.00e+00 ± 0.00e+00**	-	0.00e+00 ± 0.00e+00	-	0.00e+00 ± 2.76e−14	-	0.00e+00 ± 0.00e+00	-	7.81e+00 ± 3.83e+00	-	2.07e+00 ± 1.71e+00	-	1.32e+01 ± 2.34e+01	-	0.00e+00 ± 1.29e−13	-	0.00e+00 ± 0.00e+00	-
f_9	3.59e+01 ± 5.24e+00	3.32e+00 ± 1.63e+00	-	4.89e+00 ± 1.07e+00	-	3.49e+00 ± 1.44e+00	-	2.73e+00 ± 2.33e+00	-	7.63e+00 ± 4.03e+00	-	**1.66e+00 ± 1.34e+00**	-	2.10e+01 ± 1.95e+01	-	2.08e+01 ± 7.50e+00	-	2.34e+00 ± 8.32e−01	-
f_{10}	5.00e+02 ± 1.74e+02	7.68e+00 ± 2.32e+01	-	**1.22e−03 ± 8.66e−03**	-	2.14e+00 ± 2.32e+00	-	3.74e−01 ± 7.91e−01	-	1.53e+02 ± 1.15e+02	-	1.96e+02 ± 9.05e+01	-	3.31e+02 ± 5.73e+02	-	2.19e+02 ± 1.03e+02	-	8.57e−03 ± 2.15e−02	-
f_{11}	1.12e+03 ± 1.70e+02	**2.01e+01 ± 3.23e+01**	-	1.97e+02 ± 7.92e+01	-	9.63e+01 ± 7.57e+01	-	1.44e+02 ± 1.45e+02	-	2.08e+02 ± 1.42e+02	-	1.53e+02 ± 1.08e+02	-	7.83e+02 ± 4.98e+02	-	3.93e+02 ± 1.62e+02	-	3.21e+01 ± 3.79e+01	-
f_{12}	1.06e+00 ± 2.07e−01	1.65e−02 ± 2.38e−02	-	2.93e−01 ± 5.58e−02	-	4.22e−02 ± 2.79e−02	-	**0.00e+00 ± 0.00e+00**	-	2.69e−01 ± 5.80e−02	-	3.03e−02 ± 7.15e−02	-	8.03e−01 ± 6.47e−01	-	1.30e−01 ± 7.77e−02	-	6.82e−02 ± 1.90e−02	-
f_{13}	3.37e−01 ± 4.97e−02	3.29e−02 ± 1.57e−02	-	1.28e−01 ± 3.31e−02	-	3.55e−02 ± 1.43e−02	-	**9.44e−03 ± 6.84e−03**	-	1.31e−01 ± 4.64e−02	-	2.72e−02 ± 1.09e−02	-	3.12e−01 ± 3.40e−01	-	4.16e−01 ± 1.81e−01	=	5.16e−02 ± 1.49e−02	-
f_{14}	1.36e−01 ± 3.06e−02	1.26e−01 ± 3.36e−02	=	1.11e−01 ± 2.88e−02	-	8.91e−02 ± 3.34e−02	-	1.10e−01 ± 3.81e−02	-	2.60e−01 ± 1.25e−01	+	1.89e−01 ± 5.13e−02	+	6.88e−01 ± 1.96e+00	+	3.69e−01 ± 1.92e−01	+	**8.14e−02 ± 2.53e−02**	-
f_{15}	2.96e+00 ± 5.19e−01	4.71e−01 ± 9.60e−02	-	8.32e−01 ± 1.56e−01	-	4.35e−01 ± 1.40e−01	-	6.67e−01 ± 1.72e−01	-	7.12e−01 ± 2.38e−01	-	8.97e−01 ± 2.25e−01	-	4.70e+01 ± 1.83e+02	-	2.44e+00 ± 1.23e+00	-	**3.66e−01 ± 6.85e−02**	-
f_{16}	3.34e+00 ± 1.24e−01	**1.05e+00 ± 4.70e−01**	-	1.87e+00 ± 2.53e−01	-	1.45e+00 ± 4.03e−01	-	1.53e+00 ± 5.62e−01	-	1.41e+00 ± 5.16e−01	-	1.55e+00 ± 6.82e−01	-	2.83e+00 ± 6.29e−01	-	2.64e+00 ± 3.90e−01	-	1.24e+00 ± 3.00e−01	-
f_{17}	8.54e+03 ± 6.77e+03	7.83e+01 ± 9.49e+01	-	1.40e+00 ± 5.52e+00	-	9.36e+00 ± 1.01e+01	-	8.48e+00 ± 1.37e+01	-	2.57e+02 ± 1.62e+02	-	3.13e+02 ± 1.62e+02	-	1.01e+05 ± 4.19e+05	-	6.84e+02 ± 9.31e+02	-	**9.77e−01 ± 1.07e+00**	-
f_{18}	4.03e+03 ± 3.30e+03	5.22e+00 ± 4.59e+00	-	6.21e−01 ± 5.71e−01	-	7.83e−01 ± 7.43e−01	-	7.84e−01 ± 6.17e−01	-	3.32e+01 ± 3.33e+01	-	3.09e+01 ± 1.54e+01	-	2.25e+05 ± 1.41e+06	-	3.35e+01 ± 4.55e+01	-	2.44e−01 ± 3.11e−01	-
f_{19}	3.37e+00 ± 4.03e−01	**7.66e−02 ± 4.40e−02**	-	1.42e−01 ± 9.09e−02	-	1.58e−01 ± 1.71e−01	-	2.00e−01 ± 2.85e−01	-	2.09e+00 ± 1.08e+00	-	1.25e+00 ± 5.41e−01	-	2.37e+00 ± 3.55e+00	-	9.33e−01 ± 3.74e−01	-	7.73e−02 ± 6.34e−02	-
f_{20}	8.13e+02 ± 5.63e+02	8.06e+00 ± 1.26e+01	-	**5.59e−02 ± 8.18e−02**	-	3.13e−01 ± 3.91e−01	-	3.71e−01 ± 3.32e−01	-	1.26e+01 ± 1.16e+01	-	1.99e+01 ± 1.37e+01	-	5.69e+03 ± 1.80e+04	-	8.96e+00 ± 2.21e+01	-	1.85e−01 ± 1.78e−01	-
f_{21}	8.97e+02 ± 1.85e+02	4.93e+01 ± 7.05e+01	-	7.87e−01 ± 2.35e+00	-	1.93e+00 ± 4.95e+00	-	5.40e−01 ± 2.83e−01	-	1.03e+02 ± 1.12e+02	-	3.64e+01 ± 5.21e+01	-	5.00e+03 ± 1.56e+04	-	5.71e+01 ± 6.20e+01	-	**4.08e−01 ± 3.06e−01**	-
f_{22}	4.01e+01 ± 6.78e+00	8.47e+00 ± 1.12e+01	-	1.54e−01 ± 1.71e−01	-	2.63e−01 ± 2.76e−01	-	2.45e−01 ± 1.93e−01	-	3.00e+01 ± 3.34e+01	-	8.95e+01 ± 5.66e+01	+	6.76e+01 ± 1.30e+02	-	1.70e+01 ± 7.40e+00	-	**4.41e−02 ± 2.79e−02**	-
f_{23}	3.29e+02 ± 1.04e−02	3.29e+02 ± 9.09e−13	+	3.29e+02 ± 2.27e−12	+	3.29e+02 ± 2.27e−12	+	3.29e+02 ± 2.27e−12	+	3.29e+02 ± 2.52e−04	+	2.02e+02 ± 1.29e+01	-	3.34e+02 ± 1.45e+01	+	2.72e+02 ± 1.01e+02	+	3.29e+02 ± 9.09e−13	+
f_{24}	1.41e+02 ± 4.93e+00	1.08e+02 ± 2.95e+00	-	1.12e+02 ± 2.05e+00	-	1.09e+02 ± 2.95e+00	-	1.08e+02 ± 2.53e+00	-	1.24e+02 ± 2.39e+01	-	1.10e+02 ± 1.91e+01	-	1.31e+02 ± 2.40e+01	-	1.37e+02 ± 9.82e+00	-	**1.07e+02 ± 2.26e+00**	-
f_{25}	1.84e+02 ± 1.91e+01	1.75e+02 ± 3.23e+01	-	1.21e+02 ± 1.47e+01	-	**1.16e+02 ± 7.23e+00**	-	1.26e+02 ± 1.39e+01	-	1.86e+02 ± 2.63e+01	=	1.28e+02 ± 2.38e+01	-	1.86e+02 ± 3.12e+01	=	1.46e+02 ± 1.29e+01	-	1.33e+02 ± 4.00e+01	-
f_{26}	1.00e+02 ± 8.62e−02	1.00e+02 ± 1.40e−02	-	1.00e+02 ± 3.23e−02	-	1.00e+02 ± 1.12e−02	-	**1.00e+02 ± 1.10e−02**	-	1.00e+02 ± 4.91e−02	-	1.00e+02 ± 1.02e−02	-	1.00e+02 ± 3.34e−01	-	1.00e+02 ± 1.91e−01	=	1.00e+02 ± 1.61e−02	-
f_{27}	2.51e+02 ± 1.56e+02	1.85e+02 ± 1.55e+02	-	6.16e+01 ± 1.23e+02	-	1.72e+01 ± 7.74e+01	-	2.55e+01 ± 8.01e+01	-	2.56e+02 ± 1.66e+02	=	4.11e+01 ± 1.18e+02	-	1.59e+02 ± 1.91e+02	-	**7.42e+00 ± 2.39e+00**	-	5.81e+01 ± 1.33e+02	-
f_{28}	4.24e+02 ± 4.91e+01	3.89e+02 ± 8.07e+01	-	3.63e+02 ± 1.46e+01	-	3.61e+02 ± 4.44e+01	-	3.13e+02 ± 1.84e+01	-	4.23e+02 ± 5.53e+01	=	2.80e+02 ± 7.01e+01	-	4.20e+02 ± 8.39e+01	-	3.07e+02 ± 1.94e−01	-	3.81e+02 ± 3.14e+01	-
f_{29}	6.84e+04 ± 3.35e+05	2.27e+02 ± 1.28e+01	-	2.18e+02 ± 2.04e+01	-	**1.81e+02 ± 3.85e+01**	-	1.95e+02 ± 1.85e+01	-	3.55e+05 ± 9.31e+05	-	2.00e+02 ± 0.00e+00	-	1.68e+04 ± 6.96e+04	-	2.20e+02 ± 2.20e+01	-	2.22e+02 ± 4.58e−01	-
f_{30}	6.76e+02 ± 1.57e+02	5.85e+02 ± 6.48e+01	-	4.67e+02 ± 1.39e+01	-	4.92e+02 ± 2.28e+01	-	2.34e+02 ± 3.27e+01	-	6.38e+02 ± 1.62e+02	-	2.16e+02 ± 3.69e+01	-	1.32e+03 ± 1.84e+03	-	3.89e+02 ± 7.88e+01	-	4.65e+02 ± 1.32e+01	-

Table B.15: Average error ± standard deviation and statistic comparison for LJA (reference) versus CEC2014 competitors in 10 dimensions (part 2).

	LJA	FWA-DM		NRGA		FCDE		SOO+BOBYQA		SOO		GaAPADE		FERDE		RSDE	
f_1	6.99e+05 ± 2.56e+05	5.01e+03 ± 1.67e+04	-	2.79e+04 ± 3.10e+04	-	0.00e+00 ± 0.00e+00	-	4.57e+03 ± 0.00e+00	-	8.81e+06 ± 0.00e+00	+	0.00e+00 ± 0.00e+00	-	2.37e+00 ± 8.46e+00	-	0.00e+00 ± 0.00e+00	-
f_2	4.06e+07 ± 1.36e+07	1.34e−04 ± 9.49e−04	-	9.15e+02 ± 1.09e+03	-	0.00e+00 ± 0.00e+00	-	3.60e−02 ± 2.84e−14	-	6.34e+00 ± 8.53e−14	-	0.00e+00 ± 0.00e+00	-	6.29e−05 ± 1.11e−04	-	0.00e+00 ± 0.00e+00	-
f_3	8.81e+03 ± 3.29e+03	1.88e−09 ± 9.02e−09	-	1.52e+03 ± 1.50e+03	-	0.00e+00 ± 0.00e+00	-	5.84e+03 ± 1.82e−12	-	6.64e+03 ± 0.00e+00	-	0.00e+00 ± 0.00e+00	-	1.35e−03 ± 1.21e−03	-	0.00e+00 ± 0.00e+00	-
f_4	2.43e+01 ± 9.43e+00	1.41e+00 ± 1.60e+00	-	1.54e+01 ± 1.69e+01	-	1.84e+01 ± 1.68e+01	-	0.00e+00 ± 0.00e+00	-	6.78e−01 ± 5.68e−14	-	3.07e+01 ± 1.12e+01	+	0.00e+00 ± 0.00e+00	-	2.81e+00 ± 8.16e+00	-
f_5	2.03e+01 ± 7.94e−02	2.00e+01 ± 4.17e−02	-	1.96e+01 ± 2.77e+00	-	2.03e+01 ± 8.99e−02	=	2.00e+01 ± 0.00e+00	-	2.00e+01 ± 0.00e+00	-	1.97e+01 ± 1.71e+00	-	1.91e+01 ± 2.69e+00	-	1.92e+01 ± 3.88e+00	-
f_6	6.77e+00 ± 7.64e−01	7.06e−01 ± 6.40e−01	-	2.45e+00 ± 1.26e+00	-	3.57e+00 ± 1.40e+00	-	2.00e−03 ± 3.41e−13	-	2.00e−03 ± 3.41e−13	-	1.48e−01 ± 3.86e−01	-	8.89e−01 ± 2.75e−01	-	5.29e−02 ± 2.11e−01	-
f_7	7.03e−01 ± 7.65e−02	9.48e−02 ± 4.92e−02	-	2.03e−01 ± 1.03e−01	-	1.96e−01 ± 1.44e−01	-	4.90e−02 ± 5.68e−13	-	4.90e−02 ± 5.68e−13	-	3.16e−03 ± 4.50e−03	-	1.88e−02 ± 1.53e−02	-	3.55e−02 ± 3.09e−02	-
f_8	3.18e+01 ± 4.01e+00	2.54e−01 ± 8.09e−01	-	5.58e+00 ± 3.75e+00	-	1.61e+01 ± 5.32e+00	-	1.89e+01 ± 0.00e+00	-	1.89e+01 ± 0.00e+00	-	0.00e+00 ± 0.00e+00	-	0.00e+00 ± 0.00e+00	-	6.61e−01 ± 9.21e−01	-
f_9	3.59e+01 ± 5.24e+00	6.01e+00 ± 2.45e+00	-	8.69e+00 ± 3.92e+00	-	2.10e+01 ± 7.13e+00	-	8.96e+00 ± 1.02e−12	-	8.96e+00 ± 1.02e−12	-	3.38e+00 ± 1.03e+00	-	5.64e+00 ± 1.69e+00	-	8.52e+00 ± 3.67e+00	-
f_{10}	5.00e+02 ± 1.74e+02	1.59e+00 ± 2.08e+00	-	1.19e+02 ± 1.02e+02	-	2.92e+02 ± 2.00e+02	-	1.30e+02 ± 2.27e−13	-	1.30e+02 ± 2.27e−13	-	1.52e−01 ± 6.51e−02	-	3.67e−02 ± 3.95e−02	-	6.84e+01 ± 6.59e+01	-
f_{11}	1.12e+03 ± 1.70e+02	3.72e+02 ± 1.53e+02	-	5.76e+02 ± 2.98e+02	-	6.61e+02 ± 3.04e+02	-	3.49e+02 ± 1.14e−12	-	3.49e+02 ± 1.14e−12	-	1.83e+02 ± 1.04e+02	-	7.55e+01 ± 7.56e+01	-	2.91e+02 ± 1.91e+02	-
f_{12}	1.06e+00 ± 2.07e−01	4.25e−02 ± 4.78e−02	-	1.24e−01 ± 8.31e−02	-	7.73e−01 ± 3.65e−01	-	0.00e+00 ± 0.00e+00	-	0.00e+00 ± 0.00e+00	-	1.40e−01 ± 4.12e−02	-	1.23e−01 ± 6.98e−02	-	2.21e−01 ± 1.36e−01	-
f_{13}	3.37e−01 ± 4.97e−02	1.21e−01 ± 7.18e−02	-	1.58e−01 ± 6.15e−02	-	3.56e−01 ± 1.66e−01	=	3.00e−02 ± 4.55e−13	-	3.00e−02 ± 4.55e−13	-	6.01e−02 ± 1.38e−02	-	1.16e−01 ± 4.37e−02	-	1.28e−01 ± 3.15e−02	-
f_{14}	1.36e−01 ± 3.06e−02	2.14e−01 ± 1.20e−01	+	2.54e−01 ± 6.74e−02	+	3.47e−01 ± 1.95e−01	+	1.30e−01 ± 6.82e−13	=	1.30e−01 ± 6.82e−13	=	9.42e−02 ± 3.17e−02	-	9.36e−02 ± 2.70e−02	-	1.36e−01 ± 4.32e−02	=
f_{15}	2.96e+00 ± 5.19e−01	7.75e−01 ± 2.63e−01	-	1.02e+00 ± 4.98e−01	-	1.59e+00 ± 9.62e−01	-	4.20e−01 ± 1.14e−12	-	4.40e−01 ± 1.14e−12	-	6.06e−01 ± 9.48e−02	-	6.72e−01 ± 2.16e−01	-	9.83e−01 ± 3.67e−01	-
f_{16}	3.34e+00 ± 1.24e−01	1.76e+00 ± 4.68e−01	-	2.75e+00 ± 4.88e−01	-	3.19e+00 ± 2.65e−01	-	2.52e+00 ± 2.27e−13	-	2.52e+00 ± 2.27e−13	-	1.98e+00 ± 2.78e−01	-	1.53e+00 ± 4.59e−01	-	2.23e+00 ± 4.28e−01	-
f_{17}	8.54e+03 ± 6.77e+03	2.55e+02 ± 1.77e+02	-	1.61e+04 ± 1.75e+04	=	3.03e+02 ± 1.84e+02	-	4.23e+02 ± 2.73e−12	-	3.12e+06 ± 0.00e+00	+	9.91e+00 ± 5.41e+00	-	8.23e+00 ± 9.69e+00	-	4.77e+01 ± 5.47e+01	-
f_{18}	4.03e+03 ± 3.30e+03	2.52e+01 ± 1.83e+01	-	7.42e+03 ± 5.07e+03	+	2.27e+01 ± 1.60e+01	-	3.95e+03 ± 1.82e−12	+	1.29e+04 ± 1.46e−11	+	**2.23e−01 ± 1.87e−01**	-	2.73e+00 ± 1.29e+00	-	2.00e+00 ± 1.09e+00	-
f_{19}	3.37e+00 ± 4.03e−01	1.30e+00 ± 7.75e−01	-	2.09e+00 ± 7.73e−01	-	2.46e+00 ± 1.63e+00	-	5.50e−01 ± 1.82e−12	-	5.50e−01 ± 1.82e−12	-	2.57e−01 ± 1.17e−01	-	5.09e−01 ± 1.74e−01	-	1.03e+00 ± 3.52e−01	-
f_{20}	8.13e+02 ± 5.63e+02	1.34e+01 ± 1.16e+01	-	1.72e+03 ± 1.91e+03	=	1.78e+01 ± 1.22e+01	-	6.93e+03 ± 7.28e−12	+	9.36e+03 ± 3.64e−12	+	4.32e−01 ± 1.52e−01	-	1.70e+00 ± 7.48e−01	-	7.21e−01 ± 6.16e−01	-
f_{21}	8.97e+02 ± 1.85e+02	9.46e+01 ± 9.80e+01	-	4.82e+03 ± 4.19e+03	+	1.48e+02 ± 1.27e+02	-	1.94e+03 ± 4.55e−12	+	2.47e+04 ± 7.28e−12	+	5.09e−01 ± 2.52e−01	-	8.54e+00 ± 2.64e+01	-	1.21e+00 ± 3.30e+00	-
f_{22}	4.01e+01 ± 6.78e+00	3.41e+01 ± 4.40e+01	-	3.76e+01 ± 3.98e+01	-	2.75e+01 ± 1.36e+01	-	1.26e+02 ± 9.09e−13	+	1.26e+02 ± 2.27e−12	+	3.25e+00 ± 9.20e−01	-	3.24e+00 ± 3.92e+00	-	1.17e+01 ± 9.65e+00	-
f_{23}	3.29e+02 ± 1.04e−02	3.29e+02 ± 6.27e−08	+	3.29e+02 ± 9.32e−06	+	3.29e+02 ± 1.07e−12	+	**2.00e+02 ± 0.00e+00**	-	2.00e+02 ± 0.00e+00	-	3.29e+02 ± 2.73e−12	+	3.29e+02 ± 3.22e−11	+	3.29e+02 ± 2.27e−12	+
f_{24}	1.41e+02 ± 4.93e+00	1.27e+02 ± 2.90e+01	-	1.31e+02 ± 1.55e+01	-	1.37e+02 ± 9.86e+00	-	1.16e+02 ± 2.27e−12	-	1.16e+02 ± 2.27e−12	-	1.09e+02 ± 1.75e+00	-	1.15e+02 ± 2.51e+00	-	1.19e+02 ± 6.53e+00	-
f_{25}	1.84e+02 ± 1.91e+01	1.79e+02 ± 2.76e+01	-	1.84e+02 ± 2.05e+01	=	1.84e+02 ± 2.42e+01	=	1.39e+02 ± 9.09e−13	-	1.45e+02 ± 2.27e−12	-	1.64e+02 ± 4.11e+01	-	1.36e+02 ± 8.25e+00	-	1.30e+02 ± 1.92e+01	-
f_{26}	1.00e+02 ± 8.62e−02	1.00e+02 ± 7.47e−02	-	1.00e+02 ± 6.21e−02	-	1.00e+02 ± 1.46e−01	=	1.00e+02 ± 1.36e−12	-	1.00e+02 ± 1.36e−12	-	1.00e+02 ± 1.29e−02	-	1.00e+02 ± 4.16e−02	-	1.00e+02 ± 3.61e−02	-
f_{27}	2.51e+02 ± 1.56e+02	3.21e+02 ± 1.21e+02	=	2.81e+02 ± 1.56e+02	=	4.75e+01 ± 1.14e+02	-	2.00e+02 ± 0.00e+00	=	2.00e+02 ± 0.00e+00	=	8.97e+01 ± 1.60e+02	-	2.88e+01 ± 7.51e+01	-	9.12e+01 ± 1.38e+02	-
f_{28}	4.24e+02 ± 4.91e+01	3.47e+02 ± 4.76e+01	-	4.77e+02 ± 1.09e+02	+	4.57e+02 ± 7.90e+01	=	**2.00e+02 ± 0.00e+00**	-	2.00e+02 ± 0.00e+00	-	3.83e+02 ± 3.09e+01	-	3.66e+02 ± 7.31e+00	-	3.87e+02 ± 4.83e+01	-
f_{29}	6.84e+04 ± 3.35e+05	2.12e+02 ± 2.08e+01	-	4.13e+02 ± 7.22e+01	-	3.41e+04 ± 2.39e+05	-	2.00e+02 ± 0.00e+00	-	2.00e+02 ± 0.00e+00	-	2.22e+02 ± 6.63e−01	-	3.18e+02 ± 5.54e+01	-	2.13e+02 ± 2.56e+01	-
f_{30}	6.76e+02 ± 1.57e+02	3.94e+02 ± 1.18e+02	-	1.73e+03 ± 3.12e+02	+	8.67e+02 ± 3.55e+02	+	**2.00e+02 ± 0.00e+00**	-	2.00e+02 ± 0.00e+00	-	4.67e+02 ± 1.82e+01	-	5.35e+02 ± 6.00e+01	-	5.05e+02 ± 1.05e+02	-

Table B.16: Average error ± standard deviation and statistic comparison for LJA (reference) versus CEC2014 competitors in 30 dimensions (part 1).

	LJA	rmalschcma		DE-b6e6rl.w.r.		MVMO		UMOEAs		POBL-ADE		CMLSP		b3e3pbest		OptBees		L-SHADE	
f_1	6.31e+07 ± 1.87e+07	**0.00e+00 ± 0.00e+00**	-	4.05e+04 ± 3.40e+04	-	1.07e−03 ± 1.50e−04	-	0.00e+00 ± 0.00e+00	-	1.60e+04 ± 1.20e+04	-	3.98e−10 ± 8.57e−10	-	1.32e+07 ± 5.09e+07	-	8.57e+04 ± 3.01e+05	-	0.00e+00 ± 0.00e+00	-
f_2	4.77e+09 ± 6.03e+08	**0.00e+00 ± 0.00e+00**	-	0.00e+00 ± 0.00e+00	-	2.38e−05 ± 3.62e−05	-	0.00e+00 ± 0.00e+00	-	3.14e+02 ± 7.44e+02	-	0.00e+00 ± 0.00e+00	-	5.17e+08 ± 2.01e+09	-	3.21e−12 ± 1.13e−11	-	0.00e+00 ± 0.00e+00	-
f_3	6.91e+04 ± 1.07e+04	2.62e+01 ± 6.59e+01	-	**0.00e+00 ± 0.00e+00**	-	1.11e−03 ± 6.89e−04	-	0.00e+00 ± 0.00e+00	-	6.43e−10 ± 4.55e−09	-	1.23e−08 ± 2.12e−08	-	5.61e+03 ± 2.19e+04	-	8.41e−03 ± 3.73e−02	-	0.00e+00 ± 0.00e+00	-
f_4	4.08e+02 ± 5.38e+01	**0.00e+00 ± 0.00e+00**	-	1.62e−01 ± 9.37e−02	-	2.27e−13 ± 2.53e−13	-	0.00e+00 ± 0.00e+00	-	6.34e+01 ± 2.61e+01	-	2.17e−06 ± 2.58e−06	-	6.34e+01 ± 1.42e+02	-	1.26e+01 ± 1.36e+01	-	0.00e+00 ± 0.00e+00	-
f_5	2.09e+01 ± 4.97e−02	2.00e+01 ± 2.53e−04	-	2.03e+01 ± 2.74e−02	-	2.00e+01 ± 6.94e−04	-	2.02e+01 ± 1.63e−01	-	2.06e+01 ± 5.06e−02	-	**2.00e+01 ± 4.88e−05**	-	2.05e+01 ± 2.09e−01	-	2.00e+01 ± 1.01e−05	-	2.01e+01 ± 3.65e−02	-
f_6	3.39e+01 ± 1.29e+00	1.14e+00 ± 1.00e+00	-	1.21e+01 ± 3.41e+00	-	3.62e+00 ± 2.12e+00	-	**0.00e+00 ± 0.00e+00**	-	5.19e+00 ± 1.63e+00	-	0.00e+00 ± 0.00e+00	-	2.20e+01 ± 8.21e+00	-	1.64e+01 ± 3.41e+00	-	1.38e−07 ± 9.79e−07	-
f_7	1.58e+01 ± 2.80e+00	1.93e−04 ± 1.37e−03	-	2.30e−10 ± 1.63e−09	-	2.99e−03 ± 9.67e−03	-	**0.00e+00 ± 0.00e+00**	-	2.37e−02 ± 2.29e−02	-	0.00e+00 ± 0.00e+00	-	4.09e+00 ± 1.51e+01	-	3.75e−02 ± 3.78e−02	-	0.00e+00 ± 0.00e+00	-
f_8	2.24e+02 ± 9.93e+00	1.95e−02 ± 1.38e−01	-	**0.00e+00 ± 0.00e+00**	-	8.58e−01 ± 7.64e−01	-	1.95e+00 ± 1.68e+00	-	5.59e+01 ± 1.09e+01	-	9.84e+00 ± 2.93e+01	-	5.11e+01 ± 8.63e+01	-	1.14e−13 ± 2.70e−13	-	0.00e+00 ± 0.00e+00	-
f_9	2.61e+02 ± 1.47e+01	1.79e+01 ± 3.97e+00	-	4.41e+01 ± 6.16e+00	-	2.51e+01 ± 7.59e+00	-	8.45e+00 ± 2.11e+00	-	8.46e+01 ± 8.97e+00	-	**2.19e+00 ± 2.35e+00**	-	1.04e+02 ± 6.98e+01	-	1.37e+02 ± 3.21e+01	-	6.78e+00 ± 1.47e+00	-
f_{10}	5.68e+03 ± 3.95e+02	8.12e+01 ± 1.02e+02	-	3.51e−02 ± 2.49e−02	-	1.79e+01 ± 3.19e+01	-	1.36e+01 ± 2.36e+01	-	2.17e+03 ± 4.87e+02	-	1.47e+03 ± 4.73e+02	-	1.46e+03 ± 2.37e+03	-	1.04e+03 ± 2.50e+02	-	**1.63e−02 ± 1.56e−02**	-
f_{11}	6.88e+03 ± 3.12e+02	1.55e+03 ± 5.82e+02	-	1.98e+03 ± 2.18e+02	-	1.54e+03 ± 3.36e+02	-	1.58e+03 ± 7.34e+02	-	3.86e+03 ± 3.49e+02	-	1.82e+03 ± 7.49e+02	-	3.88e+03 ± 1.60e+03	-	2.72e+03 ± 5.63e+02	-	**1.23e+03 ± 1.81e+02**	-
f_{12}	2.49e+00 ± 2.73e−01	1.60e−02 ± 8.12e−03	-	3.39e−01 ± 4.65e−02	-	7.21e−02 ± 4.08e−02	-	2.15e−03 ± 2.22e−03	-	9.51e−01 ± 1.34e−01	-	**1.46e−04 ± 4.14e−04**	-	9.90e−01 ± 7.30e−01	-	1.81e−01 ± 6.06e−02	-	1.61e−01 ± 2.27e−02	-
f_{13}	1.08e+00 ± 1.19e−01	1.38e−01 ± 2.05e−02	-	3.36e−01 ± 5.20e−02	-	1.57e−01 ± 3.12e−02	-	5.68e−02 ± 1.85e−02	-	2.86e−01 ± 6.04e−02	-	**4.81e−02 ± 2.27e−02**	-	3.84e−01 ± 1.71e−01	-	5.61e−01 ± 1.46e−01	-	1.24e−01 ± 1.73e−02	-
f_{14}	4.33e+00 ± 1.70e+00	2.22e−01 ± 3.44e−02	-	2.43e−01 ± 2.51e−02	-	1.99e−01 ± 2.23e−02	-	2.11e−01 ± 4.29e−02	-	2.26e−01 ± 4.24e−02	-	3.12e−01 ± 5.08e−02	-	1.02e+00 ± 3.16e+00	-	4.00e−01 ± 2.29e−01	-	2.42e−01 ± 2.95e−02	-
f_{15}	5.05e+01 ± 9.36e+00	2.45e+00 ± 4.28e−01	-	5.81e+00 ± 7.13e−01	-	2.86e+00 ± 6.44e−01	-	3.09e+00 ± 6.62e−01	-	7.73e+00 ± 1.03e+00	-	3.02e+00 ± 1.22e+00	-	1.82e+02 ± 7.89e+02	-	1.27e+01 ± 6.86e+00	-	**2.15e+00 ± 2.49e−01**	-
f_{16}	1.28e+01 ± 1.78e−01	9.65e+00 ± 9.33e−01	-	9.54e+00 ± 3.60e−01	-	1.02e+01 ± 1.25e+00	-	1.05e+01 ± 6.37e−01	-	1.04e+01 ± 4.54e−01	-	1.28e+01 ± 6.98e−01	=	1.12e+01 ± 1.22e+00	-	1.09e+01 ± 6.84e−01	-	**8.50e+00 ± 4.54e−01**	-
f_{17}	2.63e+06 ± 9.76e+05	6.98e+02 ± 3.23e+02	-	2.12e+03 ± 1.91e+03	-	9.01e+02 ± 3.83e+02	-	1.01e+03 ± 3.55e+02	-	1.10e+03 ± 4.10e+02	-	9.35e+02 ± 3.84e+02	-	5.20e+05 ± 2.61e+06	-	2.74e+04 ± 4.00e+04	-	**1.88e+02 ± 7.42e+01**	-
f_{18}	1.26e+07 ± 1.06e+07	5.67e+02 ± 6.91e+02	-	2.87e+01 ± 1.56e+01	-	2.89e+01 ± 2.04e+01	-	2.32e+01 ± 1.28e+01	-	1.10e+02 ± 3.77e+01	-	7.54e+01 ± 1.60e+01	-	3.17e+06 ± 1.36e+07	-	1.96e+02 ± 4.72e+02	-	**5.91e+00 ± 2.86e+00**	-
f_{19}	3.78e+01 ± 3.46e+01	5.82e+00 ± 1.34e+00	-	4.39e+00 ± 9.69e−01	-	**3.08e+00 ± 7.40e−01**	-	3.81e+00 ± 9.38e−01	-	8.88e+00 ± 1.20e+01	-	3.88e+00 ± 6.05e−01	-	1.01e+01 ± 1.52e+01	-	7.90e+00 ± 1.86e+00	-	3.68e+00 ± 6.74e−01	-
f_{20}	9.92e+03 ± 3.69e+03	1.99e+02 ± 8.39e+01	-	1.54e+01 ± 6.47e+00	-	1.09e+02 ± 1.08e+02	-	1.17e+01 ± 5.70e+00	-	3.89e+01 ± 2.19e+01	-	9.79e+00 ± 4.64e+00	-	3.23e+03 ± 1.23e+04	-	8.53e+02 ± 7.71e+02	-	**3.08e+00 ± 1.45e+00**	-
f_{21}	6.94e+05 ± 2.03e+05	5.73e+02 ± 2.63e+02	-	2.43e+02 ± 1.42e+02	-	4.67e+02 ± 2.32e+02	-	3.18e+02 ± 1.90e+02	-	3.86e+02 ± 1.90e+02	-	2.30e+02 ± 1.17e+02	-	4.35e+05 ± 1.95e+06	-	1.74e+04 ± 1.80e+04	-	**8.68e+01 ± 8.90e+01**	-
f_{22}	5.47e+02 ± 1.05e+02	1.59e+02 ± 6.17e+01	-	6.36e+01 ± 6.57e+01	-	1.45e+02 ± 8.06e+01	-	9.59e+01 ± 8.47e+01	-	2.31e+02 ± 8.08e+01	-	6.09e+01 ± 5.68e+01	-	4.33e+02 ± 4.85e+02	-	2.32e+02 ± 9.16e+01	-	**2.76e+01 ± 1.77e+01**	-
f_{23}	3.43e+02 ± 3.41e+00	3.15e+02 ± 8.01e−05	-	3.15e+02 ± 2.27e−12	-	3.15e+02 ± 2.73e−12	-	3.14e+02 ± 1.36e−12	-	3.15e+02 ± 2.73e−12	-	**2.00e+02 ± 0.00e+00**	-	3.19e+02 ± 1.39e+01	-	3.15e+02 ± 6.59e−02	-	3.15e+02 ± 1.36e−12	-
f_{24}	2.57e+02 ± 4.04e+00	2.22e+02 ± 6.34e+00	-	2.23e+02 ± 9.21e−01	-	2.25e+02 ± 1.05e+00	-	2.24e+02 ± 5.93e+00	-	2.22e+02 ± 7.40e+00	-	2.00e+02 ± 5.00e−04	-	2.28e+02 ± 1.30e+01	-	2.36e+02 ± 5.42e+00	-	2.24e+02 ± 1.05e+00	-
f_{25}	2.16e+02 ± 2.58e+00	2.06e+02 ± 2.53e+00	-	2.03e+02 ± 4.67e−01	-	2.03e+02 ± 5.43e−01	-	2.00e+02 ± 1.38e−03	-	2.04e+02 ± 3.19e+00	-	**2.00e+02 ± 0.00e+00**	-	2.05e+02 ± 3.97e+00	-	2.01e+02 ± 1.67e−01	-	2.03e+02 ± 4.91e−02	-
f_{26}	1.01e+02 ± 1.02e−01	1.02e+02 ± 1.38e+01	-	1.00e+02 ± 4.77e−02	-	1.00e+02 ± 3.31e−02	-	**1.00e+02 ± 2.93e−02**	-	1.39e+02 ± 4.86e+01	=	1.24e+02 ± 4.23e+01	-	1.00e+02 ± 2.43e−01	-	1.01e+02 ± 1.71e−01	-	1.00e+02 ± 1.54e−02	-
f_{27}	9.86e+02 ± 2.48e+02	3.28e+02 ± 3.38e+01	-	3.52e+02 ± 4.41e+01	-	4.01e+02 ± 3.64e−01	-	3.01e+02 ± 5.20e+00	-	4.21e+02 ± 4.59e+01	-	**2.00e+02 ± 0.00e+00**	-	4.17e+02 ± 2.21e+02	-	4.02e+02 ± 9.67e−01	-	3.00e+02 ± 2.55e−13	-
f_{28}	1.13e+03 ± 6.63e+01	8.23e+02 ± 1.13e+02	-	8.20e+02 ± 3.69e+01	-	8.77e+02 ± 3.71e+01	-	3.68e+02 ± 4.76e+00	-	9.16e+02 ± 1.61e+02	-	**2.00e+02 ± 0.00e+00**	-	9.31e+02 ± 3.52e+02	-	4.31e+02 ± 1.51e+01	-	8.40e+02 ± 1.39e+01	-
f_{29}	9.82e+05 ± 2.07e+06	1.15e+03 ± 2.38e+02	-	7.87e+02 ± 1.11e+02	-	7.36e+02 ± 4.61e+01	-	2.06e+02 ± 1.33e+00	-	3.39e+05 ± 2.39e+06	-	**2.00e+02 ± 0.00e+00**	-	4.34e+05 ± 2.07e+06	-	2.16e+02 ± 1.16e+00	-	7.17e+02 ± 5.08e+00	-
f_{30}	1.09e+04 ± 4.24e+03	2.04e+03 ± 4.93e+02	-	1.32e+03 ± 6.62e+02	-	2.00e+03 ± 6.27e+02	-	4.30e+02 ± 1.55e+02	-	1.29e+03 ± 5.09e+02	-	**2.00e+02 ± 0.00e+00**	-	1.29e+04 ± 3.87e+04	-	5.93e+02 ± 9.77e+01	-	1.25e+03 ± 6.14e+02	-

Table B.17: Average error ± standard deviation and statistic comparison for LJA (reference) versus CEC2014 competitors in 30 dimensions (part 2).

	LJA	FWA-DM		NRGA		FCDE		SOO+BOBYQA		SOO		GaAPADE		FERDE		RSDE	
f_1	6.31e+07 ± 1.87e+07	2.76e+05 ± 1.82e+05	-	5.74e+05 ± 2.87e+05	-	1.05e+05 ± 9.01e+04	-	2.67e+06 ± 0.00e+00	-	2.16e+08 ± 0.00e+00	+	0.00e+00 ± 1.19e−14	-	5.02e+04 ± 5.01e+04	-	1.50e+03 ± 1.68e+03	-
f_2	4.77e+09 ± 6.03e+08	0.00e+00 ± 0.00e+00	-	9.28e+03 ± 3.91e+03	-	0.00e+00 ± 0.00e+00	-	9.96e+01 ± 3.41e−13	-	3.14e+04 ± 0.00e+00	-	0.00e+00 ± 6.89e−15	-	2.39e−03 ± 3.21e−03	-	1.19e−09 ± 5.93e−09	-
f_3	6.91e+04 ± 1.07e+04	0.00e+00 ± 0.00e+00	-	4.58e+03 ± 3.76e+03	-	1.09e+02 ± 4.01e+02	-	7.84e+03 ± 9.09e−12	-	1.08e+04 ± 3.64e−12	-	0.00e+00 ± 1.13e−14	-	1.13e−03 ± 7.29e−04	-	4.74e−02 ± 1.15e−01	-
f_4	4.08e+02 ± 5.38e+01	2.04e+01 ± 1.91e+01	-	8.06e+01 ± 3.10e+01	-	2.99e+01 ± 3.02e+01	-	3.68e+01 ± 0.00e+00	-	1.09e+02 ± 2.84e−13	-	2.79e−12 ± 1.32e−11	-	2.89e+00 ± 5.10e+00	-	3.05e+00 ± 1.33e+01	-
f_5	2.09e+01 ± 4.97e−02	2.05e+01 ± 5.36e−02	-	2.00e+01 ± 1.10e−04	-	2.09e+01 ± 7.59e−02	=	2.00e+01 ± 0.00e+00	-	2.00e+01 ± 0.00e+00	-	2.00e+01 ± 3.32e−03	-	2.00e+01 ± 7.10e−05	-	2.03e+01 ± 9.78e−02	-
f_6	3.39e+01 ± 1.29e+00	1.29e+01 ± 8.25e+00	-	1.78e+01 ± 2.18e+00	-	2.14e+01 ± 3.52e+00	-	1.91e+00 ± 5.68e−13	-	1.90e+00 ± 5.68e−13	-	6.13e−01 ± 1.16e+00	-	1.81e+01 ± 1.75e+00	-	5.16e+00 ± 1.99e+00	-
f_7	1.58e+01 ± 2.80e+00	8.55e−03 ± 9.81e−03	-	1.59e−02 ± 1.59e−02	-	2.79e−02 ± 2.71e−02	-	4.09e−01 ± 2.27e−13	-	9.96e−01 ± 4.55e−13	-	0.00e+00 ± 1.59e−14	-	0.00e+00 ± 0.00e+00	-	8.46e−04 ± 1.57e−03	-
f_8	2.24e+02 ± 9.93e+00	1.14e−13 ± 4.58e−13	-	2.66e+01 ± 7.72e+00	-	9.49e+01 ± 2.21e+01	-	9.25e+01 ± 3.41e−13	-	9.25e+01 ± 3.41e−13	-	1.75e+00 ± 1.28e+00	-	0.00e+00 ± 0.00e+00	-	2.04e+01 ± 6.97e+00	-
f_9	2.61e+02 ± 1.47e+01	5.66e+01 ± 1.08e+01	-	4.57e+01 ± 1.33e+01	-	1.35e+02 ± 3.11e+01	-	5.97e+01 ± 0.00e+00	-	5.97e+01 ± 1.14e−12	-	1.70e+01 ± 3.29e+00	-	5.50e+01 ± 1.03e+01	-	5.80e+01 ± 1.63e+01	-
f_{10}	5.68e+03 ± 3.95e+02	8.53e+00 ± 2.42e+00	-	1.07e+03 ± 4.56e+02	-	2.27e+03 ± 5.30e+02	-	2.13e+03 ± 9.09e−13	-	2.31e+03 ± 2.27e−12	-	8.14e+00 ± 4.89e+00	-	1.29e+00 ± 1.59e+00	-	3.29e+02 ± 2.44e+02	-
f_{11}	6.88e+03 ± 3.12e+02	2.63e+03 ± 2.48e+02	-	3.41e+03 ± 6.43e+02	-	3.49e+03 ± 6.73e+02	-	2.09e+03 ± 9.09e−13	-	2.15e+03 ± 0.00e+00	-	1.90e+03 ± 2.80e+02	-	2.72e+03 ± 4.64e+02	-	2.74e+03 ± 6.38e+02	-
f_{12}	2.49e+00 ± 2.73e−01	3.71e−01 ± 6.66e−02	-	1.51e−01 ± 7.04e−02	-	1.37e+00 ± 5.44e−01	-	3.00e−02 ± 6.82e−13	-	3.00e−02 ± 6.82e−13	-	2.03e−01 ± 1.28e−01	-	5.64e−01 ± 1.72e−01	-	4.44e−01 ± 1.64e−01	-
f_{13}	1.08e+00 ± 1.19e−01	3.89e−01 ± 5.51e−02	-	2.81e−01 ± 5.59e−02	-	5.68e−01 ± 1.15e−01	-	3.40e−01 ± 1.59e−12	-	3.50e−01 ± 9.09e−13	-	1.45e−01 ± 2.41e−02	-	2.81e−01 ± 4.39e−02	-	3.05e−01 ± 5.45e−02	-
f_{14}	4.33e+00 ± 1.70e+00	2.69e−01 ± 7.76e−02	-	**1.87e−01 ± 2.64e−02**	-	4.44e−01 ± 2.25e−01	-	2.80e−01 ± 4.55e−13	-	2.90e−01 ± 2.27e−13	-	2.11e−01 ± 3.12e−02	-	2.13e−01 ± 2.67e−02	-	2.36e−01 ± 3.33e−02	-
f_{15}	5.05e+01 ± 9.36e+00	7.37e+00 ± 8.46e−01	-	1.37e+01 ± 4.78e+00	-	1.54e+01 ± 6.63e+00	-	2.17e+01 ± 1.14e−12	-	2.25e+01 ± 0.00e+00	-	3.06e+00 ± 4.15e−01	-	4.17e+00 ± 7.71e−01	-	5.92e+00 ± 2.56e+00	-
f_{16}	1.28e+01 ± 1.78e−01	1.10e+01 ± 2.71e−01	-	1.15e+01 ± 6.67e−01	-	1.22e+01 ± 5.68e−01	-	9.81e+00 ± 1.14e−12	-	9.86e+00 ± 4.55e−13	-	9.88e+00 ± 4.48e−01	-	1.12e+01 ± 4.53e−01	-	1.06e+01 ± 7.62e−01	-
f_{17}	2.63e+06 ± 9.76e+05	6.29e+03 ± 5.95e+03	-	2.35e+05 ± 1.18e+05	-	6.54e+03 ± 8.63e+03	-	4.21e+04 ± 1.46e−11	-	2.81e+07 ± 0.00e+00	+	2.00e+02 ± 1.35e+02	-	1.74e+03 ± 8.90e+02	-	1.24e+03 ± 3.76e+02	-
f_{18}	1.26e+07 ± 1.06e+07	7.67e+01 ± 3.66e+01	-	5.50e+02 ± 7.09e+02	-	1.23e+02 ± 6.63e+01	-	4.16e+01 ± 1.59e−12	-	2.85e+03 ± 2.73e−12	-	9.32e+00 ± 4.70e+00	-	2.26e+01 ± 6.37e+00	-	9.54e+01 ± 4.30e+01	-
f_{19}	3.78e+01 ± 3.46e+01	9.95e+00 ± 1.93e+00	-	1.37e+01 ± 1.29e+00	-	1.32e+01 ± 1.16e+01	-	1.63e+01 ± 1.82e−12	-	1.84e+02 ± 9.09e−13	+	3.62e+00 ± 8.50e−01	-	7.65e+00 ± 7.24e−01	-	5.65e+00 ± 1.45e+00	-
f_{20}	9.92e+03 ± 3.69e+03	4.28e+01 ± 2.61e+01	-	1.14e+04 ± 5.56e+03	=	1.33e+02 ± 7.91e+01	-	3.44e+04 ± 2.18e−11	+	3.81e+04 ± 3.64e−11	+	5.59e+00 ± 2.87e+00	-	2.86e+01 ± 1.06e+01	-	3.73e+01 ± 2.52e+01	-
f_{21}	6.94e+05 ± 2.03e+05	7.29e+02 ± 9.49e+02	-	1.81e+05 ± 9.42e+04	-	3.01e+03 ± 3.29e+03	-	1.54e+04 ± 0.00e+00	-	1.63e+07 ± 0.00e+00	+	1.18e+02 ± 8.15e+01	-	6.87e+02 ± 2.09e+02	-	4.71e+02 ± 2.32e+02	-
f_{22}	5.47e+02 ± 1.05e+02	1.46e+02 ± 8.83e+01	-	4.07e+02 ± 1.29e+02	-	4.57e+02 ± 1.66e+02	-	9.56e+02 ± 9.09e−13	+	1.02e+03 ± 1.82e−12	+	6.26e+01 ± 4.95e+01	-	1.15e+02 ± 7.21e+01	-	1.91e+02 ± 1.18e+02	-
f_{23}	3.43e+02 ± 3.41e+00	3.14e+02 ± 4.55e−13	-	3.15e+02 ± 1.35e−03	-	3.15e+02 ± 2.24e−12	-	2.00e+02 ± 0.00e+00	-	2.00e+02 ± 0.00e+00	-	3.15e+02 ± 1.82e−12	-	3.14e+02 ± 2.19e−04	-	3.15e+02 ± 1.39e−06	-
f_{24}	2.57e+02 ± 4.04e+00	2.26e+02 ± 3.59e+00	-	2.28e+02 ± 4.21e+00	-	2.50e+02 ± 6.79e+00	-	**2.00e+02 ± 0.00e+00**	-	2.00e+02 ± 0.00e+00	-	2.24e+02 ± 1.73e+00	-	2.25e+02 ± 5.51e−01	-	2.24e+02 ± 1.64e+00	-
f_{25}	2.16e+02 ± 2.58e+00	2.01e+02 ± 1.98e−01	-	2.11e+02 ± 1.68e+00	-	2.07e+02 ± 3.97e+00	-	2.00e+02 ± 0.00e+00	-	2.00e+02 ± 0.00e+00	-	2.03e+02 ± 7.75e−02	-	2.00e+02 ± 1.85e−01	-	2.03e+02 ± 1.16e−01	-
f_{26}	1.01e+02 ± 1.02e−01	1.00e+02 ± 5.35e−02	-	1.00e+02 ± 9.24e−02	-	1.01e+02 ± 1.18e−01	-	2.00e+02 ± 0.00e+00	+	2.00e+02 ± 0.00e+00	+	1.00e+02 ± 2.25e−02	-	1.00e+02 ± 7.52e−02	-	1.00e+02 ± 4.10e−02	-
f_{27}	9.86e+02 ± 2.48e+02	4.01e+02 ± 3.06e+01	-	5.88e+02 ± 1.70e+02	-	6.47e+02 ± 2.48e+02	-	2.00e+02 ± 0.00e+00	-	2.00e+02 ± 0.00e+00	-	3.28e+02 ± 4.23e+01	-	3.84e+02 ± 2.29e+01	-	4.69e+02 ± 9.37e+01	-
f_{28}	1.13e+03 ± 6.63e+01	3.93e+02 ± 1.46e+01	-	1.59e+03 ± 5.70e+02	+	1.60e+03 ± 5.86e+02	+	2.00e+02 ± 0.00e+00	-	2.00e+02 ± 0.00e+00	-	8.34e+02 ± 2.75e+01	-	8.05e+02 ± 2.46e+01	-	9.05e+02 ± 1.20e+02	-
f_{29}	9.82e+05 ± 2.07e+06	2.11e+02 ± 2.90e+00	-	1.32e+03 ± 2.08e+02	-	7.55e+05 ± 2.59e+06	-	2.00e+02 ± 0.00e+00	-	2.00e+02 ± 0.00e+00	-	5.00e+05 ± 2.00e+06	-	1.19e+03 ± 1.05e+02	-	6.52e+05 ± 2.63e+06	-
f_{30}	1.09e+04 ± 4.24e+03	4.51e+02 ± 1.96e+02	-	2.89e+03 ± 5.43e+02	-	2.98e+03 ± 1.25e+03	-	2.00e+02 ± 0.00e+00	-	2.00e+02 ± 0.00e+00	-	1.65e+03 ± 6.84e+02	-	1.15e+03 ± 3.05e+02	-	1.70e+03 ± 8.58e+02	-