Research Article

Mahamed G. H. Omran* and Giovanni Iacca

An improved Jaya optimization algorithm with ring topology and population size reduction

https://doi.org/10.1515/jisys-2022-0200
received May 14, 2022; accepted October 09, 2022

Abstract: An improved variant of the Jaya optimization algorithm, called Jaya2, is proposed to enhance the performance of the original Jaya sacrificing its algorithmic design. The proposed approach arranges the solutions in a ring topology to reduce the likelihood of premature convergence. In addition, the population size reduction is used to automatically adjust the population size during the optimization process. Moreover, the translation dependency problem of the original Jaya is discussed, and an alternative solution update operation is proposed. To test Jaya2, we compare it with nine different optimization methods on the CEC 2020 benchmark functions and the CEC 2011 real-world optimization problems. The results show that Jaya2 is highly competitive on the tested problems where it generally outperforms most approaches. Having an easy-to-implement approach with little parameter tuning is highly desirable since researchers from different disciplines with basic programming skills can use it to solve their optimization problems.

Keywords: Jaya algorithm, metaheuristics, stochastic search, real-world optimization, continuous optimization

MSC 2020: 90C26, 68T20

1 Introduction

Computational intelligence optimization is an active field of research that, over the years, has proposed a plethora of successful techniques variously inspired to biological or physical phenomena such as swarm intelligence (SI) and evolutionary computation (EC) [1]. Some recent noteworthy examples of metaphor-inspired techniques include, but are not limited to:

- novel bio-inspired algorithms, e.g., the Harris hawks optimization algorithm [2], the slime mould algorithm [3], the Hunger Games Search [4], monarch butterfly optimization [5], the moth search algorithm [6], or the iterative topographical global optimization algorithm [7];
- variations of existing techniques, such as variants of the flower pollination algorithm [8], differential evolution [9], fruit fly optimization algorithm [10], or backtracking search algorithm [11,12];
- hybrid algorithms obtained by combining, for instance ant colony optimization with chaotic sequences [13], particle swarm optimization (PSO) with local solvers [14], or the bat algorithm with extremal optimization [15].

While metaphor-based algorithms have shown successful results on a broad range of problems and massively increased the popularity of this field, there is a growing consensus in the research community on

* **Corresponding author: Mahamed G. H. Omran,** Centre for Applied Mathematics and Bioinformatics, and Computer Science Department, Gulf University for Science and Technology, Mubarak Al-Abdullah, Kuwait, e-mail: omran.m@gust.edu.kw
Giovanni Iacca: Department of Information Engineering and Computer Science, University of Trento, Via Sommarive 9, 38123 Povo (TN), Italy, e-mail: giovanni.iacca@unitn.it, tel: +39 0461 28 5220

the fact that nature-inspired metaphors should not be considered anymore as the only way to explain new algorithms [16]. In fact, even though such metaphors may provide very intuitive analogies to describe the concepts behind metaheuristics, some researchers now believe that using metaphor-free descriptions should be preferred, especially when addressing practitioners or researchers from other fields.

A current trend is therefore to design metaphor-less algorithms that, rather than being based on (sometimes complex) nature-inspired analogies, are built on the principle of *simplicity* [17], also referred to as the Ockham's razor ("entities are not to be multiplied without necessity"[1]) applied to algorithmic design [18]. It is important to note however that, while the simplicity of design can be seen as a kind of "universal goal" that can applied to various algorithmic aspects, this property has also an inherently subjective nature. Therefore, here, we will focus on two *quantifiable* aspects of simplicity: (1) the ease of implementation (e.g., measured in terms of number of mathematical operations), and (2) the use of as few algorithmic-specific parameters as possible, ideally none. These two features are intuitively useful *per se*, but also from the perspective of both algorithmic designers and practitioners.

From an algorithmic designers' point of view, the design of purposely simple (and, possibly metaphor-less) algorithms can be indeed an easier task. Moreover, studying the behavior of these algorithms, hybridizing them with other techniques, or modifying some of their inner principles to improve the performance, can also be facilitated.

From the practitioners' perspective, these algorithms are attractive since they do not require a strong background in optimization, and thus, they can be applied easily to different problems in domains that are far away from computer science and applied mathematics. Moreover, practitioners may find easier to adapt, if needed, these algorithms, e.g., by incorporating domain knowledge. Or, they may find easier to port the code of the optimizer to a specific hardware platform, operating system, programming language, etc. Another practical consequence is that these algorithms may use less data structures, thus less memory, and simpler mathematical operations, hence being suitable also for hardware-constrained devices such as embedded systems [19,20] and wireless sensor networks [21]. As for the use of few or no parameters, this is desirable in that it relieves the user from a possibly time-demanding parameter tuning.

The two aforementioned features are the reasons for the popularity of a population-based algorithm such as DE [22], which in the original form uses only two algorithm-specific parameters (the scale factor and crossover rate) and has an extremely simple implementation. Another well-known example of this kind of algorithms can be considered simulated annealing [23], along with some of its variants such as that proposed in ref. [24]. Further instances are single particle algorithms [25–27], although these tend to have limited performances, especially on large-scale problems. The second issue that quite often affects these algorithms is premature convergence, i.e., they tend to quickly lose diversity [28] and obtain stuck in local optima. This is, probably, the main drawback that comes with simplicity, such that one may say that, to some extent, there is a trade-off between simplicity and performance.

One algorithm that follows this design philosophy aiming at simplicity, and that has recently attracted a growing interest, is Jaya [29]. Jaya is a population-based optimization algorithm whose main design principle is to use as few algorithm-specific parameters as possible. In particular, it only requires two parameters, i.e., the population size and the maximum number of generations, and it is extremely easy to implement. Despite these merits, Jaya is known to suffer from premature convergence, as shown, e.g., in refs [30,31]. Several modifications have been proposed to overcome this issue, but, as we will discuss later, most of these introduce various algorithmic components, more or less sophisticated, that somehow contradict the original simplicity of Jaya.

1.1 Motivation and contributions

The main motivation of this study is to propose a simple yet effective optimization algorithm. In order to that, we start from an already effective simple algorithm like Jaya, on which we apply only a few simple

1 The original quote is in Latin: "*entia non sunt multiplicanda praeter necessitatem*."

modifications. In doing that, we aim to avoid the risks of (1) increasing the algorithmic complexity and (2) making the algorithm well performing only on some benchmark problems (i.e., overfitting) while performing poorly on real-world problems. For example, in ref. [32], they found that the canonical and simple DE outperformed its more complex variants when applied to real-world problems. Given this context, the main contributions of this article are as follows:

(1) Fixing a mathematical inconsistency in the original Jaya;
(2) Proposing a new variant of Jaya, called Jaya2, while preserving its simplicity. The proposed variant uses a ring topology for the population to maintain its diversity and linearly decreases the population size as the optimization process progresses.
(3) Comparing the proposed approach with nine metaheuristic approaches of different complexities.
(4) Evaluating the proposed approach on both benchmark functions used in the IEEE competitions and real-world problems.

Obviously, to improve Jaya2 even further, one could add more complex algorithmic components (e.g., a local search algorithm). However, as we will see later, the proposed modifications are enough to make Jaya2 perform better than Jaya on most of the tested problems and show in several cases comparable results with respect to some of the most complex, state-of-the-art optimization algorithms.

The rest of this article is organized as follows. Section 2 describes the original Jaya algorithm and its variants. In Section 3, the proposed approach along with its time complexity are presented. Next, in Section 4, the results of the proposed algorithm, in comparison with eight other approaches from the literature on the IEEE CEC 2020 benchmark functions and IEEE CEC 2011 real-world problems, are presented and discussed. This analysis also includes a study of the effect of the modifications introduced in Jaya2. A brief discussion of the algorithmic overhead of the competing algorithm is also included. Finally, Section 5 concludes this article and gives directions for the future work.

2 Background: Jaya and its variants

Similar to other metaheuristics such as PSO [33] and DE, Jaya operates on a population of candidate solutions, each one assimilated to a *particle* with a given *position* in the search space, determined by the solution's variables. The algorithm proceeds iteratively by updating these positions to obtain them to the most promising regions of the search space, i.e., those characterized by better objective function values. In essence, the rationale of the Jaya search process is to move at each iteration (generation) each candidate solution far away from the worst solution in the current population, and closer to the best one.[2] This logic is controlled by the following equation:

$$x'_{i,j} = x_{i,j} + r_1(x_{\text{best},j} - |x_{i,j}|) - r_2(x_{\text{worst},j} - |x_{i,j}|), \tag{2}$$

where $x_{i,j}$ is the value of the jth variable of the ith particle, $x_{\text{best},j}$ and $x_{\text{worst},j}$ are, respectively, the value of the jth variable of the best and worst solutions in the current population, r_1 and r_2 are two random numbers

2 In what follows, we consider bound-constrained continuous optimization problems of the following general form:

$$\min f(\boldsymbol{x}), \quad \text{s.t.} \boldsymbol{x} \in \mathcal{S}, \tag{1}$$

where $\boldsymbol{x}$ is a candidate solution to the given optimization problem (we use the boldface to indicate vectors), i.e., a D-dimensional vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_D)$, where each x_i is a variable, $f(\cdot)$ is the objective function (that we assume, without loss of generality, to be minimized), and $\mathcal{S}$ is the search space domain, that in bound-constrained optimization is a hyper-rectangle defined by lower and upper bounds, respectively $\boldsymbol{l} = (l_1, l_2, \ldots, l_D)$ and $\boldsymbol{u} = (u_1, u_2, \ldots, u_D)$, s.t. $l_i \leq x_i \leq u_i \ \forall i \in \{1, 2, \ldots, D\}$. Furthermore, we consider black-box (zeroth-order) optimization settings where no specific assumption (e.g., of continuity) is made on $f(\cdot)$, thus no gradient information is available.

sampled uniformly in $[0, 1]$, and $x'_{i,j}$ is the jth variable of the new solution to be evaluated. Of note, after the application of equation (2), possibly out-of-bounds variables are corrected by saturating them at their respective lower and upper bounds, l_j and u_j. While various works [34,35] have shown that this correction mechanism can introduce a bias in an evolutionary search process, here (and in our proposal, described in the next section) we stick to this mechanism, as it is used in the original Jaya implementation [29].

After the solution update operation, in case of improvement, the new solution, $\boldsymbol{x}'_i$, replaces the old one, $\boldsymbol{x}_i$. This process is applied to each solution in the current population and is continued for a maximum number of generations.

For completeness, the pseudo-code of Jaya is given in Algorithm 1, where it can be seen that the algorithm does not require any other parameter except the population size (P) and the maximum number of generations (G). Furthermore, its implementation in terms of SLOC (source lines of code) is rather straightforward.

As discussed earlier, these two advantages have recently made Jaya quite popular in the field of numerical optimization, as also shown in various studies on engineering applications [36,37], urban traffic light scheduling [38], image classification [39,40], and binary problems [41]. Nevertheless, several works [30,31] have collected evidence on the fact that this algorithm has an inherent tendency to premature convergence. Therefore, various improved Jaya variants have been proposed in the recent literature, in the attempt to alleviate this issue. In ref. [42], a self-adaptive multi-population Jaya variant (SAMP-Jaya) was proposed, which uses a relatively complex mechanism for dividing the population into multiple sub-populations based on the quality of the solutions. This algorithm was later extended in ref. [43] to include elitism. An even more complex multi-population Jaya was proposed lately in ref. [44], where the number of sub-populations is adapted at runtime based on a convergence criterion, and each sub-population uses a different perturbation mechanism (in addition to equation (2)), the worst of which is eventually updated. A slightly simpler approach was instead proposed in ref. [30], where a Jaya variant uses three modified versions of equation (2) where the two uniform random numbers r_1 and r_2 are replaced by numbers generated by a chaotic map. In ref. [45], Cellular Automata (CA) [46] was used to organize the solutions as a tripodal mesh. Each solution has a number of neighbors determined by the topological structure used; $\boldsymbol{x}_{\text{best}}$ and $\boldsymbol{x}_{\text{worst}}$ are chosen from the solution's neighbors to avoid premature convergence. The approach generally outperforms Jaya on a set of relatively simple functions. Finally, a Lévy Flight Jaya algorithm (LJA) was recently introduced in ref. [31], which replaces r_1 and r_2 with random numbers sampled from the Lévy distribution to overcome premature convergence by allowing the algorithm to perform occasional "jumps": notably, this is one of the few recent Jaya variants that do not really add complexity to the original algorithm. In fact, several other and much more complex variants exist; however, reviewing all of them is out of the scope of this article. We refer the interested reader to the recent survey [47].

Algorithm 1: The Jaya algorithm

input : D: problem dimensions; $(\boldsymbol{l}, \boldsymbol{u})$: problem bounds;
$f(\cdot)$: objective function;
P: population size;
G: maximum number of generations
output: $\boldsymbol{x}_{\min}$ best solution; $f_{\min}$: best function value

```
1  g ← 0
2  X ← ∅
3  for i ← 1 to P do      //randomly generate P solutions within the bounds l and u
4  |  for j ← 1 to D do
5  |  |  x_{i,j} ~ U(l_j, u_j)
6  |  X ← X ∪ {x_i}
7  |  f_i ← f(x_i)        //evaluate the P solutions
8  while g < G do
```

set to $\boldsymbol{x}_{\text{best}}$ and $\boldsymbol{x}_{\text{worst}}$ the best and worst solutions in the current population
for $i \leftarrow 1$ **to** P **do**
  for $j \leftarrow 1$ **to** D **do**
    $x'_{i,j} \leftarrow x_{i,j} + r_1(x_{\text{best},j} - |x_{i,j}|) - r_2(x_{\text{worst},j} - |x_{i,j}|)$
    if $x'_{i,j} < l_j$ **then**
      $x'_{i,j} \leftarrow l_j$
    else if $x'_{i,j} > u_j$ **then**
      $x'_{i,j} \leftarrow u_j$
  $f'_i \leftarrow f(\boldsymbol{x}'_i)$
  if $f'_i < f_i$ **then**
    $\boldsymbol{x}_i \leftarrow \boldsymbol{x}'_i$
    $f_i \leftarrow f'_i$
  update $\boldsymbol{x}_{\text{min}}$ and f_{min}
  $g \leftarrow g + 1$
return $\boldsymbol{x}_{\text{min}}$, f_{min}

3 The proposed approach: Jaya2

As seen earlier, Jaya is easy to implement and requires no algorithm-specific parameters, besides the population size P and the maximum number of generations G. However, the performance of Jaya suffers from premature convergence. The main reason for this behavior is the use of equation (2), which guides each solution to move toward the best solution in the population and move away from the worst one. This behavior is similar to that of the *gbest* PSO [48], where all solutions are directly connected to the best solution in the population (i.e., a *star* topology). The star topology (Figure 1) allows each solution in the population to share information globally. In a global neighborhood, information is instantaneously disseminated to all solutions, quickly attracting the entire population to the same search region with the high risk of premature convergence. To tackle this problem, the *ring* topology (Figure 2) makes use of local neighborhoods like the one used in the *lbest* PSO [48], i.e., each particle communicates only with its immediate neighbors. Therefore, the population needs more iterations to converge, but better solutions can be found. For example, Figure 2 shows that solution $\boldsymbol{x}_1$ has two neighbors, $\boldsymbol{x}_2$ and $\boldsymbol{x}_6$, while solution $\boldsymbol{x}_4$ has $\boldsymbol{x}_3$ and $\boldsymbol{x}_5$ as its neighbors. Thus, each solution interacts with its two neighbors (rather than with the whole population). Hence, the ring lattice, known as *lbest*, is an indirect communication pattern where the best solution and worst solution, respectively, $\boldsymbol{x}_{\text{best}}$ and $\boldsymbol{x}_{\text{worst}}$, are chosen from the adjacent neighbors of each solution (i.e., for $\boldsymbol{x}_i$, $\boldsymbol{x}_{\text{best}}$, and $\boldsymbol{x}_{\text{worst}}$ are chosen from $\{\boldsymbol{x}_{i-1}, \boldsymbol{x}_i, \boldsymbol{x}_{i+1}\}$). This way we purposely "slow down" the convergence speed of Jaya, but reduce the possibility of premature convergence. In fact, the ring splits the population into many small-sized neighborhoods (of size 3), such that the exchange of information occurs only within each neighborhood, rather than at the global

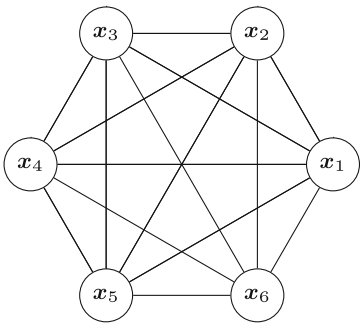

Figure 1: A star topology with six solutions.

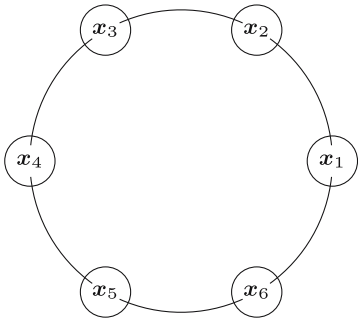

Figure 2: A ring topology with six solutions.

level. So, while in the star topology (i.e., a fully connected topology with one single neighborhood encompassing the entire population), the effect of biasing the search toward the global best would lead to a greedy/exploitative approach, in the ring topology, exploration is favored over exploitation, and hence, in this sense, this topology "slows down" the convergence, the advantage being avoiding premature convergence. This advantage has made the use of the ring topology quite popular in the metaheuristics research. Recently, Lynn et al. [49] identified the use of population topologies, including the ring one, as one of the main aspects to focus research on. Some early attempts at using the ring topology beyond PSO have been made in refs [50,51], where ring neighborhoods were used in self-adaptive DE schemes, and in ref. [52], where the ring topology was used in a hybrid PSO-DE algorithm called "bare bones" DE. In all cases, the ring topology was proved to produce performance improvements in terms of optimization results. Yet, to the best of our knowledge, the ring topology has never been applied to Jaya.

Another modification we introduce here is the use of a linearly decreasing population size. This mechanism has been used to improve the performance of many metaheuristics, see, e.g., [53,54], since it allows a stronger exploration pressure at the early stages of the optimization process, and a stronger exploitation behavior as long as the process advances. Hence, to improve the performance of Jaya, we reduce the population size linearly as a function of the number of objective function evaluations, as in L-SHADE [54]. More specifically, after each generation $g = 1, 2, \ldots, G - 1$, the new population size P_{g+1} is defined as follows:

$$P_{g+1} = \text{round}\left(\frac{P_{\min} - P_{\max}}{nfe_{\max}} \times nfe + P_{\max}\right), \tag{3}$$

where $P_{\max}$ indicates the initial population size and $P_{\min}$ indicates the population size at the last generation. The latter is set to 3, which is the smallest possible value for Jaya (given that equation (2) requires three solutions). nfe is the current number of objective function evaluations, and $nfe_{\max}$ is the maximum number of objective function evaluations. Whenever it occurs that $P_{g+1} < P_g$, the population is sorted and the worst $(P_g - P_{g+1})$ are discarded.

One last modification is related to the absolute value operator in equation (2). In fact, the use of the absolute value operator makes Jaya not translation independent, i.e., the moves and the final solution are dependent on the coordinate system. To show this, we use the approach proposed by ref. [55], where the optimization algorithm is tested on the two *identical* landscapes of the following functions:

- Function 1: $f(x) = x^2$, $x \in [-100, 100]$.
- Function 2: $f(x) = (x + 100)^2$, $x \in [-200, 0]$.

We run Jaya under the following conditions [55]:

- population size = 25;
- number of runs = 15; and
- number of iterations/run = 5.

On Function 1, the average best solution found by Jaya is 0.1300263, while on Function 2, it is only 1.686978.

To fix this problem, we propose to replace equation (2) with the following equation:

$$x'_{i,j} = x_{i,j} + r_1(x_{\text{best},j} - x_{i,j}) - r_2(x_{\text{worst},j} - x_{i,j}). \tag{4}$$

Now when testing Jaya using equation (4) (instead of equation (2)) on Function 1 and Function 2, we obtain the same result (namely, 0.2853698). Thus, the translation dependency problem of Jaya has been removed.

The new proposed variant is called Jaya2, and its detailed pseudo-code is listed in Algorithm 2. A flowchart describing the Jaya2 algorithm is shown in Figure 3.

Algorithm 2: The Jaya2 algorithm

input : D: problem dimensions; $(\boldsymbol{l}, \boldsymbol{u})$: problem bounds;
$f(\cdot)$: objective function;
$P_{\max}$: maximum population size;
$P_{\min}$: minimum population size (set to 3);
G: maximum number of generations
output: $\boldsymbol{x}_{\min}$ best solution; $f_{\min}$: best function value

$g \leftarrow 0$
$P \leftarrow P_{\max}$
$\boldsymbol{X} \leftarrow \varnothing$
for $i \leftarrow 1$ **to** P //randomly generate P solutions within the bounds $\boldsymbol{l}$ and $\boldsymbol{u}$
  for $j \leftarrow 1$ **to** D **do**
  $x_{i,j} \sim U(l_j, u_j)$
  $\boldsymbol{X} \leftarrow \boldsymbol{X} \cup \{\boldsymbol{x}_i\}$
  $f_i \leftarrow f(\boldsymbol{x}_i)$ // evaluate the P solutions
while $g < G$ **do**
  for $i \leftarrow 1$ **to** P **do**
    set to $\boldsymbol{x}_{\text{best}}$ and $\boldsymbol{x}_{\text{worst}}$ the best and worst solutions in $\{\boldsymbol{x}_{i-1}, \boldsymbol{x}_i, \boldsymbol{x}_{i+1}\}$
    for $j \leftarrow 1$ **to** D **do**
      $x'_{i,j} \leftarrow x_{i,j} + r_1(x_{\text{best},j} - x_{i,j}) - r_2(x_{\text{worst},j} - x_{i,j})$
      if $x'_{i,j} < l_j$ **then**
        $x'_{i,j} \leftarrow l_j$
      else if $x'_{i,j} > u_j$ **then**
        $x'_{i,j} \leftarrow u_j$
    $f'_i \leftarrow f(\boldsymbol{x}'_i)$
    if $f'_i < f_i$ **then**
      $\boldsymbol{x}_i \leftarrow \boldsymbol{x}'_i$
      $f_i \leftarrow f'_i$
    update $\boldsymbol{x}_{\min}$ and $f_{\min}$
  $P_{g+1} = \text{round}\left(\frac{P_{\min} - P_{\max}}{nfe_{\max}} \times nfe + P_{\max}\right)$
  if $P_g < P_{g+1}$ **then**
    sort solutions in $\boldsymbol{X}$ based on their objective function values
    delete the worst $P_g - P_{g+1}$ solutions
    randomly reshuffle the remaining solutions
  $g \leftarrow g + 1$
return $\boldsymbol{x}_{\min}$, $f_{\min}$

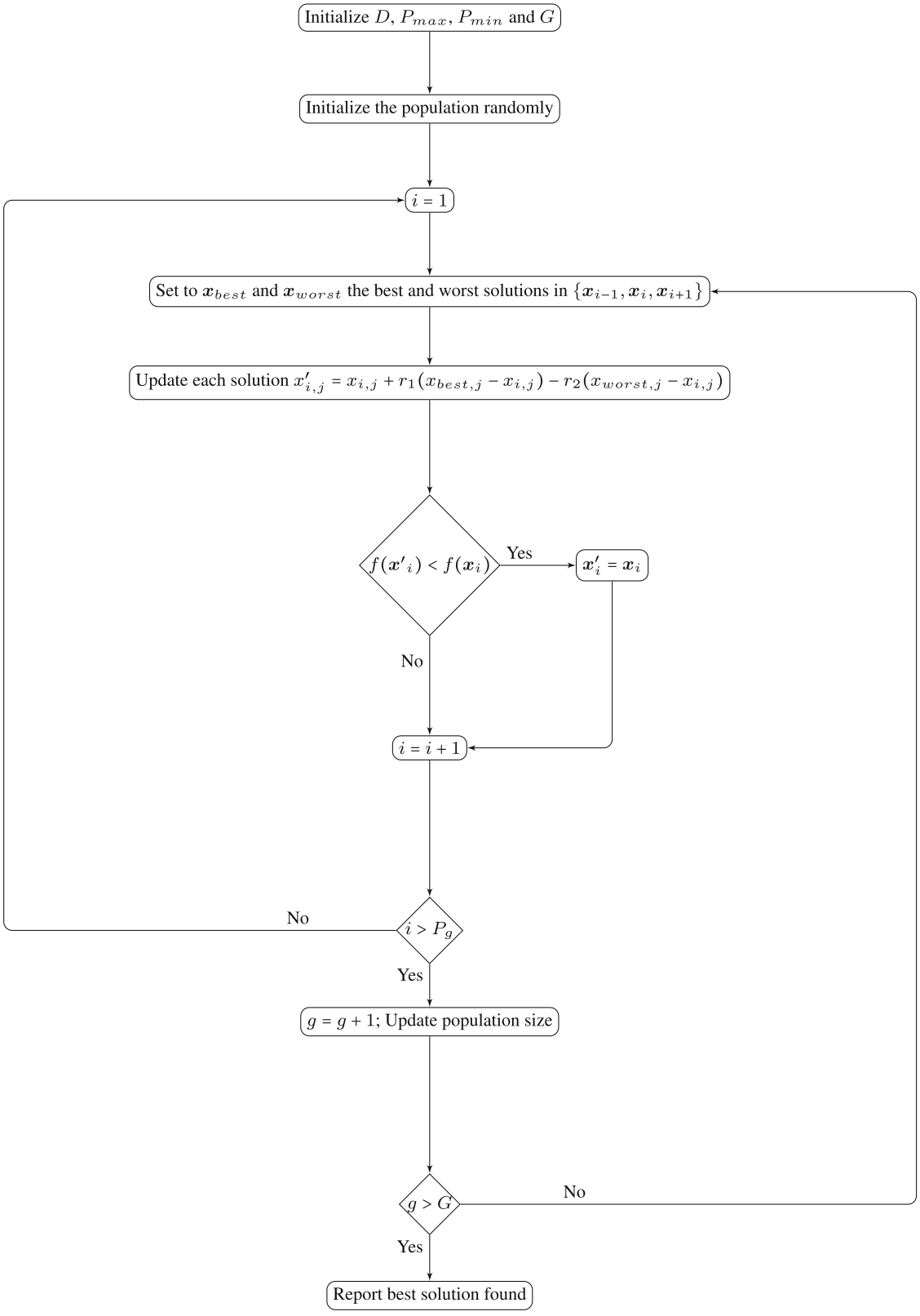

Figure 3: Flowchart of the Jaya2 algorithm.

3.1 Time complexity

The complexity of the proposed algorithm, represented using the big-O notation, is a function of the population size (P), the generations (G), the number of problem dimensions (D), and the cost of the function evaluation (c). More specifically, $O(\text{Jaya2})$ can be computed as follows:

$$O(\text{initialization}) + O(G \times P \times \text{function evaluations}) + O(G \times \text{population update}) + O(G \times \text{population reduction}), \tag{5}$$

which can be rewritten as follows:

$$O(P \cdot D + G \cdot P \cdot c + G \cdot P \cdot D + G \cdot P \cdot \log(P)) \cong O(G \cdot (P \cdot c + P \cdot D + P \cdot \log(P))). \tag{6}$$

Hence, the complexity of Jaya2 is of a polynomial order. An empirical investigation of the computational overhead of Jaya2 appears in Section 4.6.

4 Experimental results

Two different sets of problems, namely, the 10 benchmark functions used at the CEC 2020 competition on single-objective bound-constrained numerical optimization [56] and the 22 real-world optimization problems from the CEC 2011 competition on testing evolutionary algorithms on real-world optimization problems [57], have been used in the experiments.

Table 1 summarizes the 10 CEC 2020 benchmark functions. The first four functions are shifted and rotated well-known functions. The next three functions are hybrid, which means they are linear combinations of several basic functions. Finally, the last three functions are compositions, i.e., couplings between the previous functions considered in different parts of their optimization domains. All functions are scalable, which means that they can be computed for different numbers of problem dimensions D. Accordingly, the search range for all functions is $[-100, 100]^D$. In our experiments, we set D to 10. The mathematical description of each function can be found in ref. [56].

Table 2 summarizes the 22 CEC 2011 real-world problems. These include various engineering problems, such as optimization of chemical systems, optimal design of TLC devices, optimal control, and spacecraft trajectory optimization. In this case, the number of problem dimensions depends on the specific problem, ranging from 1 (T_3, T_4) to 215 ($T_{11.2}$), and apart from a few cases, the function type and its optimum are in general unknown. We should note that we did not use T_3 in our experiments due to its large computing time.

Table 1: Summary of the CEC 2020 benchmark functions

Function	Type	Optimum value
f_1	Unimodal function	100
f_2	Basic function	1,100
f_3	Basic function	700
f_4	Basic function	1,900
f_5	Hybrid function	1,700
f_6	Hybrid function	1,600
f_7	Hybrid function	2,100
f_8	Composition function	2,200
f_9	Composition function	2,400
f_{10}	Composition function	2,500

Table 2: Summary of the CEC 2011 real-world problems

Problem	Description	*D*	Constraints
T_1	A frequency-modulated sound waves problem	6	Bound constrained
T_2	A Lennard–Jones potential problem	30	Bound constrained
T_3	A bifunctional catalyst blend control problem	1	Bound constrained
T_4	A stirred tank reactor control problem	1	Unconstrained
T_5	A Tersoff potential minimization problem	30	Bound constrained
T_6	A Tersoff potential minimization problem	30	Bound constrained
T_7	A radar polyphase code design problem	20	Bound constrained
T_8	A transmission network expansion problem	7	Equality/inequality constraints
T_9	A transmission pricing problem	126	Linear equality constraints
T_{10}	An antenna array design problem	12	Bound constrained
$T_{11.1}$	A dynamic economic dispatch problem	120	Inequality constraints
$T_{11.2}$	A dynamic economic dispatch problem	216	Inequality constraints
$T_{11.3}$	A static economic dispatch problem	6	Inequality constraints
$T_{11.4}$	A static economic dispatch problem	13	Inequality constraints
$T_{11.5}$	A static economic dispatch problem	15	Inequality constraints
$T_{11.6}$	A static economic dispatch problem	40	Inequality constraints
$T_{11.7}$	A static economic dispatch problem	140	Inequality constraints
$T_{11.8}$	A hydrothermal scheduling problem	96	Inequality constraints
$T_{11.9}$	A hydrothermal scheduling problem	96	Inequality constraints
$T_{11.10}$	A hydrothermal scheduling problem	96	Inequality constraints
T_{12}	A spacecraft trajectory optimization problem	26	Bound constrained
T_{13}	A spacecraft trajectory optimization problem	22	Bound constrained

However, this exclusion does not affect the overall results of our comparative analysis reported next. Furthermore, as detailed in Table 2, some problems include equality and/or inequality constraints other than the bound constraints. These were handled with a static penalty function proportionate to the sum of the constraint violations, as in the original Matlab formulation provided by the CEC 2011 competition organizers. For more details, the interested reader is referred to ref. [57].

We have chosen these two benchmark sets for the following reasons:

- The CEC 2020 benchmark offers a broad set of hard-to-solve optimization functions with different characteristics (e.g., uni-modal, multi-modal), which helps examining the performance of the proposed method on different types of problems.
- The CEC 2011 benchmark, on the other hand, allows to assess the applicability of the proposed method in real-world problems from diverse fields.

Following the indications of the two benchmarks, each of the compared algorithms was executed for 30 independent runs for the CEC 2020 benchmark functions, and for 25 runs for the CEC 2011 real-world problems. For the CEC 2020 benchmark functions, we used 100,000 function evaluations, while for the CEC 2011 real-world problems, we used 150,000 function evaluations.

The numerical experiments were performed on an HP Desktop with 3.6 GHz Intel Core *i*7, 32 GB RAM, running Matlab R2017b on MS Windows 10 Pro.

The detailed statistics of Jaya2, in terms of best, median, mean, standard deviation (std.), and worst objective function values (across the best function values obtained at the end of the allotted budget in all the available runs for each problem) on the CEC 2020 benchmark functions and the CEC 2011 real-world problems are reported in Tables 3 and 4, respectively.

For the comparative analysis presented in the rest of this section, we used the pairwise Wilcoxon signed rank test [58] with a 5% significance level, to verify the existence of statistical differences between the competing approaches. Given that the Wilcoxon test confronts the median values of two samples, in Tables 5–21, reported next, we show the comparative results in terms of median function values (also in this

Table 3: The CEC 2020 ($D = 10$) function values achieved by Jaya2 over 30 runs and 100,000 function evaluations

Function	Best	Median	Mean	Std.	Worst
f_1	1.0078×10^{2}	6.0145×10^{2}	1.0176×10^{3}	9.5179×10^{2}	3.2385×10^{3}
f_2	1.1036×10^{3}	1.1153×10^{3}	1.1646×10^{3}	1.1127×10^{2}	1.5826×10^{3}
f_3	7.1066×10^{2}	7.1387×10^{2}	7.1394×10^{2}	1.5745×10^{0}	7.1721×10^{2}
f_4	1.9005×10^{3}	1.9008×10^{3}	1.9009×10^{3}	2.8970×10^{-1}	1.9015×10^{3}
f_5	1.7740×10^{3}	2.3698×10^{3}	2.5043×10^{3}	5.7657×10^{2}	4.1150×10^{3}
f_6	1.6005×10^{3}	1.6005×10^{3}	1.6006×10^{3}	1.1825×10^{-1}	1.6008×10^{3}
f_7	2.1013×10^{3}	2.1183×10^{3}	2.1348×10^{3}	4.7341×10^{1}	2.2911×10^{3}
f_8	2.2000×10^{3}	2.3004×10^{3}	2.2972×10^{3}	1.8354×10^{1}	2.3014×10^{3}
f_9	2.5000×10^{3}	2.7344×10^{3}	2.7111×10^{3}	7.1632×10^{1}	2.7444×10^{3}
f_{10}	2.8977×10^{3}	2.8996×10^{3}	2.9106×10^{3}	2.0356×10^{1}	2.9459×10^{3}

case across the best function values obtained at the end of the allotted budget in all the available runs of each algorithm for each problem), along with the corresponding p-values obtained with the Wilcoxon test, which represent the probability of two selected samples of algorithm results being statistically equivalent ($\alpha = 0.05$). We should remark that, for each comparison, Jaya2 was separately rerun: this explains the small fluctuations on the median function values shown in Tables 3–21, which anyway do not affect the conclusions drawn on the comparative analysis.

In what follows, first, we present the comparison on both testbeds between the proposed Jaya2 algorithm and both the original Jaya and its recent variant LJA. Then, we present the comparison between Jaya2 and three alternative few-parameter approaches from the literature. This is followed by a comparison with three complex, state-of-the-art algorithms. Finally, we conclude with an analysis of the contribution to the performance of each algorithmic element introduced in Jaya2.

Table 4: The CEC 2011 function values achieved by Jaya2 over 25 runs and 150,000 function evaluations

Problem	Best	Median	Mean	Std.	Worst
T_1	0.0000×10^{0}	0.0000×10^{0}	3.2404×10^{0}	5.3596×10^{0}	1.4779×10^{1}
T_2	-2.7543×10^{1}	-2.7047×10^{1}	-2.6676×10^{1}	8.6573×10^{-1}	-2.4865×10^{1}
T_4	1.3771×10^{1}	1.3771×10^{1}	1.5875×10^{1}	3.2409×10^{0}	2.0957×10^{1}
T_5	-3.5934×10^{1}	-3.1719×10^{1}	-3.0962×10^{1}	3.2419×10^{0}	-2.2626×10^{1}
T_6	-2.9166×10^{1}	-2.3006×10^{1}	-2.2415×10^{1}	2.5966×10^{0}	-1.9512×10^{1}
T_7	8.5119×10^{-1}	1.5571×10^{0}	1.5358×10^{0}	2.8696×10^{-1}	1.8745×10^{0}
T_8	2.2000×10^{2}	2.2000×10^{2}	2.2000×10^{2}	0.0000×10^{0}	2.2000×10^{2}
T_9	8.5532×10^{2}	1.6129×10^{3}	1.7192×10^{3}	6.0105×10^{2}	2.9524×10^{3}
T_{10}	-2.1632×10^{1}	-2.1387×10^{1}	-2.0097×10^{1}	2.6999×10^{0}	-1.2470×10^{1}
$T_{11.1}$	5.1485×10^{4}	5.2100×10^{4}	5.2182×10^{4}	4.5418×10^{2}	5.3117×10^{4}
$T_{11.2}$	1.7463×10^{7}	1.7628×10^{7}	1.7629×10^{7}	1.2689×10^{5}	1.7970×10^{7}
$T_{11.3}$	1.5447×10^{4}	1.5467×10^{4}	1.5470×10^{4}	1.4056×10^{1}	1.5495×10^{4}
$T_{11.4}$	1.8692×10^{4}	1.9180×10^{4}	1.9157×10^{4}	1.7630×10^{2}	1.9380×10^{4}
$T_{11.5}$	3.2731×10^{4}	3.2817×10^{4}	3.2823×10^{4}	5.9609×10^{1}	3.2944×10^{4}
$T_{11.6}$	1.3059×10^{5}	1.3508×10^{5}	1.3542×10^{5}	2.4490×10^{3}	1.4058×10^{5}
$T_{11.7}$	1.9408×10^{6}	2.0420×10^{6}	2.0461×10^{6}	1.0970×10^{5}	2.3876×10^{6}
$T_{11.8}$	9.4323×10^{5}	1.0044×10^{6}	1.0552×10^{6}	1.2831×10^{5}	1.3388×10^{6}
$T_{11.9}$	9.7082×10^{5}	1.3801×10^{6}	1.3800×10^{6}	2.5126×10^{5}	1.9100×10^{6}
$T_{11.10}$	9.4323×10^{5}	1.0044×10^{6}	1.0552×10^{6}	1.2831×10^{5}	1.3388×10^{6}
T_{12}	1.1881×10^{1}	1.4757×10^{1}	1.4782×10^{1}	1.3436×10^{0}	1.7302×10^{1}
T_{13}	1.1106×10^{1}	1.9045×10^{1}	1.8986×10^{1}	2.2571×10^{0}	2.2419×10^{1}

Table 5: The CEC 2020 ($D = 10$) median function values achieved by Jaya2, Jaya, and LJA over 30 runs and 100,000 function evaluations

Function	Jaya2	Jaya	LJA
f_1	$\mathbf{4.63 \times 10^2}$	1.32×10^8	9.86×10^3
f_2	$\mathbf{1.12 \times 10^3}$	2.20×10^3	2.09×10^3
f_3	$\mathbf{7.14 \times 10^2}$	7.52×10^2	7.36×10^2
f_4	$\mathbf{1.90 \times 10^3}$	$\mathbf{1.90 \times 10^3}$	$\mathbf{1.90 \times 10^3}$
f_5	$\mathbf{2.15 \times 10^3}$	1.19×10^4	4.23×10^3
f_6	$\mathbf{1.60 \times 10^3}$	$\mathbf{1.60 \times 10^3}$	$\mathbf{1.60 \times 10^3}$
f_7	$\mathbf{2.11 \times 10^3}$	3.05×10^3	2.43×10^3
f_8	$\mathbf{2.30 \times 10^3}$	2.32×10^3	$\mathbf{2.30 \times 10^3}$
f_9	$\mathbf{2.73 \times 10^3}$	2.77×10^3	2.76×10^3
f_{10}	$\mathbf{2.90 \times 10^3}$	2.95×10^3	$\mathbf{2.90 \times 10^3}$

The boldface indicates the lowest median function value per function.

4.1 Jaya2 vs the original Jaya and LJA

In this section, Jaya2 is compared against the original Jaya and the recently proposed Jaya variant LJA [31]. We have chosen LJA because it is one of the most recently proposed Jaya variant and because, contrary to the other Jaya variants discussed in Section 2, it kept the simplicity of Jaya almost intact. It only introduces one parameter for the Lévy distribution, i.e., β. Moreover, ref. [31] shows that LJA generally outperforms Jaya on the 30 CEC 2014 functions. We have used a population size of 30 for Jaya and LJA, while the initial population size of Jaya2 ($P_{\max}$) was set to 100. For LJA, β was set to 1.8 as recommended in ref. [31].

The median function values of the objective functions of Jaya2, Jaya, and LJA on the 10 functions of CEC 2020 are reported in Table 5. Table 6 shows the results of the statistical analysis based on the Wilcoxon test. The results of the two tables show that Jaya2 outperformed Jaya and LJA on almost all the functions.

Figure 4 shows the convergence behavior on six selected CEC 2020 benchmark functions (i.e., the best function value found at each step of the algorithm, averaged across 30 runs) for the three Jaya-based

Table 6: The CEC 2020 ($D = 10$) statistical analysis showing the Wilcoxon test p-value

Function	Jaya	LJA
f_1	$1.73 \times 10^{-6}(+)$	$1.92 \times 10^{-6}(+)$
f_2	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$
f_3	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$
f_4	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$
f_5	$1.73 \times 10^{-6}(+)$	$2.35 \times 10^{-6}(+)$
f_6	$5.75 \times 10^{-6}(+)$	$8.47 \times 10^{-6}(+)$
f_7	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$
f_8	$3.11 \times 10^{-5}(+)$	$1.73 \times 10^{-6}(+)$
f_9	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$
f_{10}	$1.02 \times 10^{-5}(+)$	$1.63 \times 10^{-1}(=)$
Wins	10	9
Draws	0	1
Loses	0	0

Each p-value is followed by a "+" if Jaya2 is significantly better than the competing algorithm, "−" if significantly worse, or "=" if statistically equivalent.

Table 7: The CEC 2011 median function values achieved by Jaya2, Jaya, and LJA over 25 runs and 150,000 function evaluations

Problem	Jaya2	Jaya	LJA
T_1	$\mathbf{0.00 \times 10^0}$	1.79×10^1	1.72×10^1
T_2	$\mathbf{-2.65 \times 10^1}$	-8.07×10^0	-6.39×10^0
T_4	$\mathbf{1.38 \times 10^1}$	1.39×10^1	1.39×10^1
T_5	$\mathbf{-3.15 \times 10^1}$	-2.01×10^1	-2.16×10^1
T_6	$\mathbf{-2.30 \times 10^1}$	-1.44×10^1	-1.53×10^1
T_7	1.70×10^0	1.72×10^0	$\mathbf{1.67 \times 10^0}$
T_8	$\mathbf{2.20 \times 10^2}$	$\mathbf{2.20 \times 10^2}$	$\mathbf{2.20 \times 10^2}$
T_9	$\mathbf{1.56 \times 10^3}$	1.79×10^4	6.98×10^3
T_{10}	$\mathbf{-2.14 \times 10^1}$	-1.07×10^1	-1.14×10^1
$T_{11.1}$	5.25×10^4	$\mathbf{5.23 \times 10^4}$	6.28×10^4
$T_{11.2}$	1.76×10^7	1.76×10^7	$\mathbf{1.75 \times 10^7}$
$T_{11.3}$	$\mathbf{1.55 \times 10^4}$	$\mathbf{1.55 \times 10^4}$	$\mathbf{1.55 \times 10^4}$
$T_{11.4}$	1.93×10^4	$\mathbf{1.91 \times 10^4}$	$\mathbf{1.91 \times 10^4}$
$T_{11.5}$	$\mathbf{3.28 \times 10^4}$	$\mathbf{3.28 \times 10^4}$	$\mathbf{3.28 \times 10^4}$
$T_{11.6}$	1.35×10^5	$\mathbf{1.33 \times 10^5}$	$\mathbf{1.33 \times 10^5}$
$T_{11.7}$	$\mathbf{1.99 \times 10^6}$	$\mathbf{1.99 \times 10^6}$	2.05×10^6
$T_{11.8}$	$\mathbf{1.01 \times 10^6}$	1.38×10^6	1.53×10^6
$T_{11.9}$	$\mathbf{1.44 \times 10^6}$	1.49×10^6	1.85×10^6
$T_{11.10}$	$\mathbf{1.01 \times 10^6}$	1.38×10^6	1.53×10^6
T_{12}	$\mathbf{1.54 \times 10^1}$	2.93×10^1	3.07×10^1
T_{13}	$\mathbf{1.93 \times 10^1}$	2.88×10^1	2.96×10^1

The boldface indicates the lowest median function value per problem.

Table 8: The CEC 2011 statistical analysis showing the Wilcoxon test p-value

Problem	Jaya	LJA
T_1	$3.22 \times 10^{-5}(+)$	$2.26 \times 10^{-5}(+)$
T_2	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
T_4	$1.91 \times 10^{-2}(+)$	$3.97 \times 10^{-1}(=)$
T_5	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
T_6	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
T_7	$3.26 \times 10^{-1}(=)$	$7.78 \times 10^{-1}(=)$
T_8	$3.13 \times 10^{-2}(-)$	$1.00 \times 10^0(=)$
T_9	$2.00 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
T_{10}	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
$T_{11.1}$	$6.77 \times 10^{-1}(=)$	$1.23 \times 10^{-5}(+)$
$T_{11.2}$	$7.36 \times 10^{-2}(=)$	$3.00 \times 10^{-1}(=)$
$T_{11.3}$	$3.22 \times 10^{-5}(+)$	$3.28 \times 10^{-4}(+)$
$T_{11.4}$	$8.08 \times 10^{-4}(-)$	$1.08 \times 10^{-3}(-)$
$T_{11.5}$	$8.27 \times 10^{-2}(=)$	$1.83 \times 10^{-1}(=)$
$T_{11.6}$	$4.43 \times 10^{-1}(=)$	$2.42 \times 10^{-1}(=)$
$T_{11.7}$	$3.82 \times 10^{-1}(=)$	$3.96 \times 10^{-2}(+)$
$T_{11.8}$	$5.13 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
$T_{11.9}$	$1.58 \times 10^{-1}(=)$	$1.26 \times 10^{-4}(+)$
$T_{11.10}$	$5.13 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
T_{12}	$1.39 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
T_{13}	$1.57 \times 10^{-5}(+)$	$1.57 \times 10^{-5}(+)$
Wins	12	14
Draws	7	6
Loses	2	1

Each p-value is followed by a "+" if Jaya2 is significantly better than the competing algorithm, "−" if significantly worse, or "=" if statistically equivalent.

Table 9: The CEC 2020 ($D = 10$) median function values achieved by Jaya2, DFO, BFPA, and LCBO over 30 runs and 100,000 function evaluations

Function	Jaya2	DFO	BFPA	LCBO
f_1	4.82×10^2	5.42×10^9	$\mathbf{1.00 \times 10^2}$	7.06×10^2
f_2	$\mathbf{1.12 \times 10^3}$	2.55×10^3	1.63×10^3	1.52×10^3
f_3	$\mathbf{7.13 \times 10^2}$	1.02×10^3	7.30×10^2	7.18×10^2
f_4	$\mathbf{1.90 \times 10^3}$	2.02×10^3	$\mathbf{1.90 \times 10^3}$	$\mathbf{1.90 \times 10^3}$
f_5	2.33×10^3	1.61×10^5	$\mathbf{1.72 \times 10^3}$	2.83×10^3
f_6	$\mathbf{1.60 \times 10^3}$	$\mathbf{1.60 \times 10^3}$	$\mathbf{1.60 \times 10^3}$	$\mathbf{1.60 \times 10^3}$
f_7	2.12×10^3	3.18×10^4	$\mathbf{2.10 \times 10^3}$	2.14×10^3
f_8	$\mathbf{2.30 \times 10^3}$	2.65×10^3	$\mathbf{2.30 \times 10^3}$	$\mathbf{2.30 \times 10^3}$
f_9	2.73×10^3	2.78×10^3	$\mathbf{2.50 \times 10^3}$	2.73×10^3
f_{10}	$\mathbf{2.90 \times 10^3}$	3.29×10^3	$\mathbf{2.90 \times 10^3}$	2.94×10^3

The boldface indicates the lowest median function value per function.

Table 10: The CEC 2020 ($D = 10$) statistical analysis showing the Wilcoxon test p-value

Function	DFO	BFPA	LCBO
f_1	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(-)$	$4.91 \times 10^{-1}(=)$
f_2	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$	$2.60 \times 10^{-6}(+)$
f_3	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$	$5.71 \times 10^{-4}(+)$
f_4	$1.73 \times 10^{-6}(+)$	$4.07 \times 10^{-5}(+)$	$4.99 \times 10^{-3}(-)$
f_5	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(-)$	$2.43 \times 10^{-2}(+)$
f_6	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(-)$	$7.69 \times 10^{-6}(+)$
f_7	$1.73 \times 10^{-6}(+)$	$2.60 \times 10^{-5}(-)$	$3.16 \times 10^{-3}(+)$
f_8	$1.92 \times 10^{-6}(+)$	$8.59 \times 10^{-2}(=)$	$2.56 \times 10^{-1}(=)$
f_9	$1.73 \times 10^{-6}(+)$	$5.22 \times 10^{-6}(-)$	$5.30 \times 10^{-1}(=)$
f_{10}	$1.73 \times 10^{-6}(+)$	$4.73 \times 10^{-6}(-)$	$2.83 \times 10^{-4}(+)$
Wins	10	3	6
Draws	0	1	3
Loses	0	6	1

Each p-value is followed by a "+" if Jaya2 is significantly better than the competing algorithm, "−" if significantly worse, or "=" if statistically equivalent.

approaches. The figure shows that Jaya2 generally reaches better solutions faster than the two other approaches. This suggests that Jaya2 has a good balance between exploration and exploitation, focusing on exploration at the beginning of the run and then switching to exploitation. Figure 5 shows the average diversity (across 30 runs) of the population during the progress of the three Jaya algorithms on the same six functions shown in Figure 4. The diversity of the population is measured as in ref. [59], namely:

$$\mathrm{DI} = \sqrt{\frac{1}{P}\sum_{i=1}^{P}\sum_{j=1}^{D}(x_{i,j} - \overline{x}_j)^2}, \tag{7}$$

where $\overline{x}_j$ is the jth component of the mean vector of the solutions in the population. The figure shows that Jaya2 tends to start with a high diversity, while toward the end of the run, the diversity drops rapidly, allowing for exploitation of good solutions. This behavior is due to a combination of the effects of the ring topology and the linearly decreasing population size. On the contrary, both Jaya and LJA do not make use of these mechanisms; thus, they keep large distances between solutions over all generations, which prevents them from exploiting good regions in the search space.

Table 11: The CEC 2011 median function values achieved by Jaya2, DFO, BFPA, and LCBO over 25 runs and 150,000 function evaluations

Problem	Jaya2	DFO	BFPA	LCBO
T_1	$\mathbf{0.00 \times 10^0}$	2.56×10^1	8.55×10^0	1.12×10^1
T_2	$\mathbf{-2.65 \times 10^1}$	-5.79×10^0	-8.74×10^0	-2.09×10^1
T_4	1.39×10^1	2.10×10^1	$\mathbf{1.38 \times 10^1}$	1.43×10^1
T_5	$\mathbf{-3.16 \times 10^1}$	-1.59×10^1	-2.98×10^1	-1.97×10^1
T_6	$\mathbf{-2.30 \times 10^1}$	-8.10×10^0	-2.27×10^1	-1.56×10^1
T_7	1.70×10^0	$\mathbf{1.16 \times 10^0}$	1.25×10^0	1.42×10^0
T_8	$\mathbf{2.20 \times 10^2}$	$\mathbf{2.20 \times 10^2}$	$\mathbf{2.20 \times 10^2}$	$\mathbf{2.20 \times 10^2}$
T_9	$\mathbf{1.75 \times 10^3}$	2.65×10^3	1.57×10^6	1.91×10^5
T_{10}	$\mathbf{-2.14 \times 10^1}$	-1.20×10^1	-2.03×10^1	-2.06×10^1
$T_{11.1}$	$\mathbf{5.24 \times 10^4}$	5.30×10^4	1.16×10^7	4.52×10^6
$T_{11.2}$	$\mathbf{1.76 \times 10^7}$	$\mathbf{1.76 \times 10^7}$	4.07×10^7	6.06×10^7
$T_{11.3}$	1.55×10^4	1.55×10^4	$\mathbf{1.54 \times 10^4}$	1.55×10^4
$T_{11.4}$	1.92×10^4	1.91×10^4	$\mathbf{1.83 \times 10^4}$	1.92×10^4
$T_{11.5}$	$\mathbf{3.28 \times 10^4}$	3.29×10^4	3.29×10^4	3.30×10^4
$T_{11.6}$	1.36×10^5	1.41×10^5	$\mathbf{1.29 \times 10^5}$	1.48×10^5
$T_{11.7}$	1.98×10^6	$\mathbf{1.94 \times 10^6}$	2.10×10^6	$1.76 \times \mathbf{10^{10}}$
$T_{11.8}$	9.73×10^5	$\mathbf{9.56 \times 10^5}$	2.77×10^7	1.61×10^8
$T_{11.9}$	1.42×10^6	$\mathbf{1.33 \times 10^6}$	2.89×10^7	1.78×10^8
$T_{11.10}$	9.73×10^5	$\mathbf{9.56 \times 10^5}$	2.77×10^7	1.61×10^8
T_{12}	$\mathbf{1.47 \times 10^1}$	2.27×10^1	2.32×10^1	2.07×10^1
T_{13}	$\mathbf{2.01 \times 10^1}$	2.90×10^1	2.61×10^1	2.50×10^1

The boldface indicates the lowest median function value per problem.

Table 12: The CEC 2011 statistical analysis showing the Wilcoxon test p-value

Problem	DFO	BFPA	LCBO
T_1	$1.23 \times 10^{-5}(+)$	$6.02 \times 10^{-4}(+)$	$5.46 \times 10^{-4}(+)$
T_2	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$	$1.77 \times 10^{-5}(+)$
T_4	$1.32 \times 10^{-3}(+)$	$6.09 \times 10^{-4}(-)$	$2.21 \times 10^{-1}(=)$
T_5	$1.23 \times 10^{-5}(+)$	$1.04 \times 10^{-1}(=)$	$1.57 \times 10^{-5}(+)$
T_6	$1.23 \times 10^{-5}(+)$	$3.82 \times 10^{-1}(=)$	$2.26 \times 10^{-5}(+)$
T_7	$3.62 \times 10^{-5}(-)$	$7.22 \times 10^{-5}(-)$	$3.03 \times 10^{-2}(-)$
T_8	$2.50 \times 10^{-1}(=)$	$1.00 \times 10^0(=)$	$1.00 \times 10^0(=)$
T_9	$1.49 \times 10^{-2}(+)$	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
T_{10}	$1.23 \times 10^{-5}(+)$	$8.19 \times 10^{-1}(=)$	$5.45 \times 10^{-1}(=)$
$T_{11.1}$	$2.66 \times 10^{-4}(+)$	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
$T_{11.2}$	$1.28 \times 10^{-2}(-)$	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
$T_{11.3}$	$4.57 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(-)$	$3.70 \times 10^{-2}(-)$
$T_{11.4}$	$3.22 \times 10^{-3}(-)$	$1.23 \times 10^{-5}(-)$	$1.28 \times 10^{-2}(+)$
$T_{11.5}$	$1.08 \times 10^{-3}(+)$	$6.02 \times 10^{-4}(+)$	$1.39 \times 10^{-5}(+)$
$T_{11.6}$	$5.35 \times 10^{-3}(+)$	$1.23 \times 10^{-5}(-)$	$1.57 \times 10^{-5}(+)$
$T_{11.7}$	$2.76 \times 10^{-1}(=)$	$5.44 \times 10^{-2}(=)$	$1.23 \times 10^{-5}(+)$
$T_{11.8}$	$5.63 \times 10^{-1}(=)$	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
$T_{11.9}$	$8.19 \times 10^{-1}(=)$	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
$T_{11.10}$	$5.63 \times 10^{-1}(=)$	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
T_{12}	$2.86 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$	$1.26 \times 10^{-4}(+)$
T_{13}	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$	$1.39 \times 10^{-5}(+)$
Wins	13	11	16
Draws	5	5	3
Loses	3	5	2

Each p-value is followed by a "+" if Jaya2 is significantly better than the competing algorithm, "−" if significantly worse, or "=" if statistically equivalent.

Table 13: Parameter settings for the state-of-the-art approaches

Algorithm	Parameters
GA-MPC	Population size = 90 and $p = 0.1$
L-SHADE	NP = $18 \times D$, $p = 0.11$, and number of historical circle memories is 5
jSO	NP = $25 \times \sqrt{D} \times \log(D)$, p is linearly decreasing from 0.25 to 0.125, and number of historical circle memories is 5
SS	$N_{\text{init}} = 100$, $p = 0.1$, rank = $0.5 \times D$, and $c = 0.5$

The three Jaya-based approaches were then tested on the CEC 2011 real-world problems. The results, summarized in Tables 7 and 8, confirm the superiority of Jaya2 compared to Jaya and LJA. The only two problems where Jaya managed to statistically outperform Jaya2 were T_8 and $T_{11.4}$. LJA performed better than Jaya2 on only $T_{11.4}$. Figures 6 and 7 depict the convergence behavior and the diversity of the three approaches. The figures are generally similar to Figures 4 and 5; thus, similar conclusions can be drawn.

Table 14: The CEC 2020 ($D = 10$) median function values achieved by Jaya2, GA-MPC, L-SHADE, jSO, and SS over 30 runs and 100,000 function evaluations

Function	Jaya2	GA-MPC	L-SHADE	jSO	SS
f_1	5.94×10^2	$\mathbf{1.00 \times 10^2}$	$\mathbf{1.00 \times 10^2}$	$\mathbf{1.00 \times 10^2}$	$\mathbf{1.00 \times 10^2}$
f_2	$\mathbf{1.11 \times 10^3}$	1.35×10^3	$\mathbf{1.11 \times 10^3}$	$\mathbf{1.11 \times 10^3}$	2.25×10^3
f_3	7.14×10^2	7.19×10^2	$\mathbf{7.12 \times 10^2}$	$\mathbf{7.12 \times 10^2}$	7.33×10^2
f_4	$\mathbf{1.90 \times 10^3}$	$\mathbf{1.90 \times 10^3}$	$\mathbf{1.90 \times 10^3}$	$\mathbf{1.90 \times 10^3}$	$\mathbf{1.90 \times 10^3}$
f_5	2.08×10^3	1.71×10^3	$\mathbf{1.70 \times 10^3}$	$\mathbf{1.70 \times 10^3}$	1.72×10^3
f_6	$\mathbf{1.60 \times 10^3}$	$\mathbf{1.60 \times 10^3}$	$\mathbf{1.60 \times 10^3}$	$\mathbf{1.60 \times 10^3}$	$\mathbf{1.60 \times 10^3}$
f_7	2.11×10^3	$\mathbf{2.10 \times 10^3}$	$\mathbf{2.10 \times 10^3}$	$\mathbf{2.10 \times 10^3}$	$\mathbf{2.10 \times 10^3}$
f_8	$\mathbf{2.30 \times 10^3}$	$\mathbf{2.30 \times 10^3}$	$\mathbf{2.30 \times 10^3}$	$\mathbf{2.30 \times 10^3}$	$\mathbf{2.30 \times 10^3}$
f_9	2.73×10^3	2.73×10^3	2.73×10^3	2.73×10^3	$\mathbf{2.72 \times 10^3}$
f_{10}	$\mathbf{2.90 \times 10^3}$	2.94×10^3	$\mathbf{2.90 \times 10^3}$	$\mathbf{2.90 \times 10^3}$	2.92×10^3

The boldface indicates the lowest median function value per function.

Table 15: The CEC 2020 ($D = 10$) statistical analysis showing the Wilcoxon test p-value

Function	GA-MPC	L-SHADE	jSO	SS
f_1	$4.07 \times 10^{-5}(-)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(-)$
f_2	$1.97 \times 10^{-5}(+)$	$1.40 \times 10^{-2}(-)$	$2.70 \times 10^{-2}(-)$	$1.73 \times 10^{-6}(+)$
f_3	$1.25 \times 10^{-4}(+)$	$1.59 \times 10^{-3}(-)$	$2.83 \times 10^{-4}(-)$	$1.73 \times 10^{-6}(+)$
f_4	$1.04 \times 10^{-3}(-)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(+)$
f_5	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(-)$
f_6	$1.06 \times 10^{-1}(=)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(-)$	$4.73 \times 10^{-6}(-)$
f_7	$2.77 \times 10^{-3}(-)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(-)$	$2.13 \times 10^{-6}(-)$
f_8	$4.68 \times 10^{-3}(+)$	$8.26 \times 10^{-6}(-)$	$8.25 \times 10^{-6}(-)$	$7.14 \times 10^{-4}(-)$
f_9	$7.04 \times 10^{-1}(=)$	$1.04 \times 10^{-3}(-)$	$5.45 \times 10^{-2}(=)$	$1.59 \times 10^{-1}(=)$
f_{10}	$1.11 \times 10^{-2}(+)$	$3.04 \times 10^{-1}(=)$	$2.40 \times 10^{-2}(-)$	$5.80 \times 10^{-1}(=)$
Wins	4	0	0	3
Draws	2	1	1	2
Loses	4	9	9	5

Each p-value is followed by a "+" if Jaya2 is significantly better than the competing algorithm, "−" if significantly worse, or "=" if statistically equivalent.

Table 16: The CEC 2011 median function values achieved by Jaya2, GA-MPC, L-SHADE, jSO, and SS over 25 runs and 150,000 function evaluations

Function	Jaya2	GA-MPC	L-SHADE	jSO	SS
T_1	0.00×10^{0}	0.00×10^{0}	$\mathbf{1.10 \times 10^{-20}}$	0.00×10^{0}	1.27×10^{1}
T_2	-2.65×10^{1}	$\mathbf{-2.72 \times 10^{1}}$	-2.63×10^{1}	-2.69×10^{1}	-1.47×10^{1}
T_4	$\mathbf{1.39 \times 10^{1}}$	$\mathbf{1.39 \times 10^{1}}$	2.10×10^{1}	2.10×10^{1}	2.18×10^{1}
T_5	-3.18×10^{1}	-3.44×10^{1}	$\mathbf{-3.65 \times 10^{1}}$	-3.01×10^{1}	-2.16×10^{1}
T_6	-2.30×10^{1}	-2.30×10^{1}	$\mathbf{-2.92 \times 10^{1}}$	-1.88×10^{1}	-1.56×10^{1}
T_7	1.62×10^{0}	$\mathbf{9.06 \times 10^{-1}}$	1.16×10^{0}	1.03×10^{0}	2.54×10^{0}
T_8	$\mathbf{2.20 \times 10^{2}}$	$\mathbf{2.20 \times 10^{2}}$	$\mathbf{2.20 \times 10^{2}}$	$\mathbf{2.20 \times 10^{2}}$	3.33×10^{2}
T_9	$\mathbf{1.74 \times 10^{3}}$	5.02×10^{4}	2.66×10^{3}	2.58×10^{4}	3.37×10^{6}
T_{10}	-2.14×10^{1}	-2.13×10^{1}	$\mathbf{-2.16 \times 10^{1}}$	-1.04×10^{1}	-1.49×10^{1}
$T_{11.1}$	5.23×10^{4}	5.27×10^{4}	$\mathbf{5.19 \times 10^{4}}$	5.69×10^{6}	6.33×10^{6}
$T_{11.2}$	$\mathbf{1.76 \times 10^{7}}$	2.17×10^{7}	1.78×10^{7}	2.44×10^{7}	6.13×10^{7}
$T_{11.3}$	1.55×10^{4}	$\mathbf{1.54 \times 10^{4}}$	$\mathbf{1.54 \times 10^{4}}$	$\mathbf{1.54 \times 10^{4}}$	1.56×10^{4}
$T_{11.4}$	1.92×10^{4}	1.85×10^{4}	$\mathbf{1.81 \times 10^{4}}$	$\mathbf{1.81 \times 10^{4}}$	5.86×10^{5}
$T_{11.5}$	3.28×10^{4}	3.29×10^{4}	$\mathbf{3.27 \times 10^{4}}$	1.41×10^{6}	4.11×10^{6}
$T_{11.6}$	1.35×10^{5}	1.36×10^{5}	$\mathbf{1.24 \times 10^{5}}$	5.24×10^{7}	2.56×10^{6}
$T_{11.7}$	2.01×10^{6}	2.00×10^{6}	$\mathbf{1.85 \times 10^{6}}$	$1.27 \times \mathbf{10^{10}}$	$2.15 \times \mathbf{10^{10}}$
$T_{11.8}$	1.01×10^{6}	9.92×10^{5}	$\mathbf{9.32 \times 10^{5}}$	2.59×10^{7}	1.75×10^{8}
$T_{11.9}$	1.38×10^{6}	1.21×10^{6}	$\mathbf{9.40 \times 10^{5}}$	2.59×10^{7}	1.78×10^{8}
$T_{11.10}$	9.79×10^{5}	9.88×10^{5}	$\mathbf{9.31 \times 10^{5}}$	2.59×10^{7}	1.72×10^{8}
T_{12}	$\mathbf{1.52 \times 10^{1}}$	1.61×10^{1}	1.61×10^{1}	7.46×10^{1}	4.17×10^{1}
T_{13}	1.92×10^{1}	2.07×10^{1}	$\mathbf{1.42 \times 10^{1}}$	N/A	4.10×10^{1}

The boldface indicates the lowest median function value per problem. The "N/A" means that the data are not available.

Table 17: The CEC 2011 statistical analysis showing the Wilcoxon test p-value

Problem	GA-MPC	L-SHADE	jSO	SS
T_1	$6.38 \times 10^{-1}(=)$	$4.26 \times 10^{-1}(=)$	$7.81 \times 10^{-3}(-)$	$1.82 \times 10^{-4}(+)$
T_2	$6.38 \times 10^{-1}(=)$	$2.42 \times 10^{-1}(=)$	$1.35 \times 10^{-1}(=)$	$1.23 \times 10^{-5}(+)$
T_4	$6.64 \times 10^{-1}(=)$	$1.54 \times 10^{-3}(+)$	$2.67 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
T_5	$4.46 \times 10^{-4}(-)$	$1.23 \times 10^{-5}(-)$	$8.40 \times 10^{-1}(=)$	$2.00 \times 10^{-5}(+)$
T_6	$6.38 \times 10^{-1}(=)$	$1.23 \times 10^{-5}(-)$	$4.07 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
T_7	$1.57 \times 10^{-5}(-)$	$4.57 \times 10^{-5}(-)$	$2.86 \times 10^{-5}(-)$	$1.23 \times 10^{-5}(+)$
T_8	$1.00 \times 10^{0}(=)$	$1.00 \times 10^{0}(=)$	$1.00 \times 10^{0}(=)$	$8.69 \times 10^{-6}(+)$
T_9	$1.23 \times 10^{-5}(+)$	$1.57 \times 10^{-3}(+)$	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
T_{10}	$6.53 \times 10^{-2}(=)$	$1.23 \times 10^{-5}(-)$	$1.23 \times 10^{-5}(+)$	$3.62 \times 10^{-5}(+)$
$T_{11.1}$	$2.70 \times 10^{-3}(+)$	$6.93 \times 10^{-2}(=)$	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
$T_{11.2}$	$1.23 \times 10^{-5}(+)$	$1.39 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
$T_{11.3}$	$1.77 \times 10^{-5}(-)$	$1.23 \times 10^{-5}(-)$	$1.23 \times 10^{-5}(-)$	$1.23 \times 10^{-5}(+)$
$T_{11.4}$	$1.23 \times 10^{-5}(-)$	$1.23 \times 10^{-5}(-)$	$1.23 \times 10^{-5}(-)$	$1.23 \times 10^{-5}(+)$
$T_{11.5}$	$5.13 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(-)$	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
$T_{11.6}$	$4.43 \times 10^{-1}(=)$	$1.23 \times 10^{-5}(-)$	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
$T_{11.7}$	$6.19 \times 10^{-1}(=)$	$1.23 \times 10^{-5}(-)$	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
$T_{11.8}$	$2.11 \times 10^{-1}(=)$	$1.23 \times 10^{-5}(-)$	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
$T_{11.9}$	$2.06 \times 10^{-3}(-)$	$1.23 \times 10^{-5}(-)$	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
$T_{11.10}$	$8.19 \times 10^{-1}(=)$	$1.23 \times 10^{-5}(-)$	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
T_{12}	$1.04 \times 10^{-1}(=)$	$3.51 \times 10^{-3}(+)$	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$
T_{13}	$7.80 \times 10^{-2}(=)$	$5.13 \times 10^{-5}(-)$	N/A	$1.23 \times 10^{-5}(+)$
Wins	4	4	13	21
Draws	12	4	3	0
Loses	5	13	4	0

Each p-value is followed by a "+" if Jaya2 is significantly better than the competing algorithm, "−" if significantly worse, or "=" if statistically equivalent. The "N/A" means that the data are not available.

Table 18: The CEC 2020 ($D = 10$) median function values achieved by Jaya2, Jaya2 (no ring), Jaya2 (fix pop), and Jaya2 (equation (2)) over 30 runs and 100,000 function evaluations

Function	Jaya2	Jaya2 (no ring)	Jaya2 (fix pop)	Jaya2 (equation (2))
f_1	$\mathbf{5.89 \times 10^2}$	9.81×10^3	1.13×10^3	6.89×10^7
f_2	$\mathbf{1.12 \times 10^3}$	1.48×10^3	1.73×10^3	1.66×10^3
f_3	$\mathbf{7.14 \times 10^2}$	7.29×10^2	7.29×10^2	7.42×10^2
f_4	$\mathbf{1.90 \times 10^3}$	$\mathbf{1.90 \times 10^3}$	$\mathbf{1.90 \times 10^3}$	$\mathbf{1.90 \times 10^3}$
f_5	$\mathbf{2.18 \times 10^3}$	3.13×10^3	2.75×10^3	5.53×10^3
f_6	$\mathbf{1.60 \times 10^3}$	$\mathbf{1.60 \times 10^3}$	$\mathbf{1.60 \times 10^3}$	$\mathbf{1.60 \times 10^3}$
f_7	$\mathbf{2.12 \times 10^3}$	2.22×10^3	2.21×10^3	2.81×10^3
f_8	$\mathbf{2.30 \times 10^3}$	$\mathbf{2.30 \times 10^3}$	$\mathbf{2.30 \times 10^3}$	$\mathbf{2.31 \times 10^3}$
f_9	$\mathbf{2.73 \times 10^3}$	2.75×10^3	2.74×10^3	2.76×10^3
f_{10}	$\mathbf{2.90 \times 10^3}$	$\mathbf{2.90 \times 10^3}$	$\mathbf{2.90 \times 10^3}$	$\mathbf{2.92 \times 10^3}$

The boldface indicates the lowest median function value per function.

Table 19: The CEC 2020 ($D = 10$) statistical analysis showing the Wilcoxon test p-value

Function	Jaya2 (no ring)	Jaya2 (fix pop)	Jaya2 (equation (2))
f_1	$2.60 \times 10^{-6}(+)$	$1.57 \times 10^{-2}(+)$	$1.73 \times 10^{-6}(+)$
f_2	$1.49 \times 10^{-5}(+)$	$2.35 \times 10^{-6}(+)$	$1.02 \times 10^{-5}(+)$
f_3	$1.92 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$
f_4	$1.24 \times 10^{-5}(+)$	$2.35 \times 10^{-6}(+)$	$3.93 \times 10^{-1}(=)$
f_5	$2.05 \times 10^{-4}(+)$	$2.83 \times 10^{-4}(+)$	$1.73 \times 10^{-6}(+)$
f_6	$1.11 \times 10^{-1}(=)$	$5.58 \times 10^{-1}(=)$	$2.11 \times 10^{-3}(+)$
f_7	$4.07 \times 10^{-5}(+)$	$8.47 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$
f_8	$7.51 \times 10^{-5}(+)$	$2.35 \times 10^{-6}(+)$	$3.11 \times 10^{-5}(+)$
f_9	$1.73 \times 10^{-6}(+)$	$2.84 \times 10^{-5}(+)$	$1.17 \times 10^{-2}(+)$
f_{10}	$2.30 \times 10^{-2}(+)$	$5.81 \times 10^{-1}(=)$	$2.41 \times 10^{-3}(+)$
Wins	9	8	9
Draws	1	2	1
Loses	0	0	0

Each p-value is followed by a "+" if Jaya2 is significantly better than the competing algorithm, "−" if significantly worse, or "=" if statistically equivalent.

4.2 Jaya2 vs few-parameter approaches

In this section, Jaya2 is compared with other metaheuristics that have at most one algorithm-specific parameter; thus, they share the same fundamental design principle of Jaya. The three chosen approaches are as follows:

- Dispersive flies optimisation (DFO) [60], which has one parameter, the disturbance threshold (dt), which is set to 0.001 as suggested in ref. [60];
- Biotic flower pollination algorithm (BFPA) [61], which has one parameter, C, which is set to 1 as recommended in ref. [61];
- Life choice-based optimizer (LCBO) [62], which is a recent parameter-less metaheuristic.

For all the compared algorithms, the population size was set to 30.

Tables 9 and 10 summarize the results obtained when applying Jaya2, DFO, BFPA, and LCBO on the CEC 2020 benchmark functions. The results of the Wilcoxon test show that Jaya2 outperformed DFO on all

Table 20: The CEC 2011 median function values achieved by Jaya2, Jaya2 (no ring), Jaya2 (fix pop) and Jaya2 (equation (2)) over 25 runs and 150,000 function evaluations

Problem	Jaya2	Jaya2 (no ring)	Jaya2 (fix pop)	Jaya2 (equation (2))
T_1	$\mathbf{0.00 \times 10^{0}}$	1.18×10^{1}	3.70×10^{-1}	1.56×10^{-3}
T_2	$\mathbf{-2.65 \times 10^{1}}$	-1.53×10^{1}	-1.24×10^{1}	-2.54×10^{1}
T_4	1.40×10^{1}	$\mathbf{1.38 \times 10^{1}}$	2.10×10^{1}	1.40×10^{1}
T_5	$\mathbf{-3.19 \times 10^{1}}$	-3.15×10^{1}	-2.98×10^{1}	-2.19×10^{1}
T_6	$\mathbf{-2.30 \times 10^{1}}$	$\mathbf{-2.30 \times 10^{1}}$	-2.10×10^{1}	-1.59×10^{1}
T_7	1.67×10^{0}	1.68×10^{0}	$\mathbf{1.38 \times 10^{0}}$	1.74×10^{0}
T_8	$\mathbf{2.20 \times 10^{2}}$	$\mathbf{2.20 \times 10^{2}}$	$\mathbf{2.20 \times 10^{2}}$	$\mathbf{2.20 \times 10^{2}}$
T_9	1.61×10^{3}	3.86×10^{3}	3.50×10^{3}	$\mathbf{1.36 \times 10^{3}}$
T_{10}	$\mathbf{-2.14 \times 10^{1}}$	-1.23×10^{1}	-1.54×10^{1}	-1.69×10^{1}
$T_{11.1}$	$\mathbf{5.22 \times 10^{4}}$	5.23×10^{4}	5.32×10^{4}	5.24×10^{4}
$T_{11.2}$	$\mathbf{1.75 \times 10^{7}}$	1.77×10^{7}	$\mathbf{1.75 \times 10^{7}}$	1.76×10^{7}
$T_{11.3}$	$\mathbf{1.55 \times 10^{4}}$	$\mathbf{1.55 \times 10^{4}}$	$\mathbf{1.55 \times 10^{4}}$	$\mathbf{1.55 \times 10^{4}}$
$T_{11.4}$	1.92×10^{4}	1.91×10^{4}	$\mathbf{1.90 \times 10^{4}}$	1.92×10^{4}
$T_{11.5}$	3.29×10^{4}	$\mathbf{3.28 \times 10^{4}}$	3.29×10^{4}	$\mathbf{3.28 \times 10^{4}}$
$T_{11.6}$	1.34×10^{5}	$\mathbf{1.32 \times 10^{5}}$	1.39×10^{5}	1.34×10^{5}
$T_{11.7}$	1.99×10^{6}	2.03×10^{6}	$\mathbf{1.96 \times 10^{6}}$	1.99×10^{6}
$T_{11.8}$	$\mathbf{1.02 \times 10^{6}}$	1.13×10^{6}	1.03×10^{6}	1.03×10^{6}
$T_{11.9}$	1.47×10^{6}	$\mathbf{1.15 \times 10^{6}}$	1.27×10^{6}	1.45×10^{6}
$T_{11.10}$	$\mathbf{1.02 \times 10^{6}}$	1.13×10^{6}	1.03×10^{6}	1.03×10^{6}
T_{12}	$\mathbf{1.54 \times 10^{1}}$	1.76×10^{1}	2.05×10^{1}	1.68×10^{1}
T_{13}	2.05×10^{1}	$\mathbf{1.82 \times 10^{1}}$	2.22×10^{1}	2.28×10^{1}

The boldface indicates the lowest median function value per problem.

functions. In addition, Jaya2 outperformed LCBO on most functions. On the other hand, compared to BFPA, Jaya2 performed better on the three basic functions while performing worse on six functions.

The four approaches were then tested on the CEC 2011 real-world problems, and the results are reported in Tables 11 and 12. The results on these problems clearly show the superiority of Jaya2 compared to DFO, BFPA, and LCBO. In fact, contrary to what we observed on the CEC 2020 benchmark functions, Jaya2 outperformed BFPA on 11 problems, while BFPA performed better on only five problems.

Hence, from the results of this section, we can say that Jaya2 is superior to DFO, BFPA, and LCBO, with few exceptions (in particular, the CEC 2020 hybrid and composition functions where BFPA appears to perform slightly better).

4.3 Jaya2 vs state-of-the-art approaches

In this section, Jaya2 is compared with three state-of-the-art and more complex methods, namely:

- the Genetic Algorithm with Multi-Parent Crossover (GA-MPC) [63], which is the winner of the IEEE CEC 2011 competition.
- L-SHADE [54], one of the best DE variants to date, that ranked first in the IEEE CEC 2014 competition.
- jSO [64], an enhanced variant of an improved version of L-SHADE, iL-SHADE [65]. It was the second-best optimizer at the CEC 2017 competition, and according to ref. [66], jSO was generally the best approach on two benchmark suites (CEC 2013 and CEC 2017) compared to nine other state-of-the-art approaches. The main differences between jSO and L-SHADE are the weighted mutation strategy and a modified adaptation of DE control parameters. The adaptation process of jSO control parameters is based on the current stage of the search process.

Table 21: The CEC 2011 statistical analysis showing the Wilcoxon test p-value

Problem	Jaya2 (no ring)	Jaya2 (fix pop)	Jaya2 (equation (2))
T_1	$1.29 \times 10^{-3}(+)$	$3.51 \times 10^{-3}(+)$	$1.02 \times 10^{-2}(+)$
T_2	$1.23 \times 10^{-5}(+)$	$1.23 \times 10^{-5}(+)$	$8.71 \times 10^{-3}(+)$
T_4	$2.33 \times 10^{-2}(-)$	$3.13 \times 10^{-3}(+)$	$8.27 \times 10^{-1}(=)$
T_5	$7.36 \times 10^{-2}(=)$	$1.86 \times 10^{-2}(+)$	$4.57 \times 10^{-5}(+)$
T_6	$1.50 \times 10^{-1}(=)$	$2.30 \times 10^{-2}(+)$	$9.04 \times 10^{-5}(+)$
T_7	$4.76 \times 10^{-1}(=)$	$1.30 \times 10^{-3}(-)$	$9.89 \times 10^{-1}(=)$
T_8	$1.00 \times 10^{0}(=)$	$1.00 \times 10^{0}(=)$	$1.00 \times 10^{0}(=)$
T_9	$4.57 \times 10^{-5}(+)$	$2.40 \times 10^{-4}(+)$	$7.80 \times 10^{-2}(=)$
T_{10}	$2.26 \times 10^{-5}(+)$	$1.74 \times 10^{-4}(+)$	$1.74 \times 10^{-4}(+)$
$T_{11.1}$	$7.78 \times 10^{-1}(=)$	$2.86 \times 10^{-5}(+)$	$5.10 \times 10^{-1}(=)$
$T_{11.2}$	$4.53 \times 10^{-3}(+)$	$4.43 \times 10^{-1}(=)$	$2.11 \times 10^{-1}(=)$
$T_{11.3}$	$2.01 \times 10^{-1}(=)$	$2.21 \times 10^{-1}(=)$	$8.71 \times 10^{-3}(-)$
$T_{11.4}$	$9.80 \times 10^{-2}(=)$	$6.65 \times 10^{-4}(-)$	$7.16 \times 10^{-1}(=)$
$T_{11.5}$	$8.08 \times 10^{-4}(-)$	$2.40 \times 10^{-4}(+)$	$2.76 \times 10^{-1}(=)$
$T_{11.6}$	$1.15 \times 10^{-1}(=)$	$1.89 \times 10^{-3}(+)$	$7.78 \times 10^{-1}(=)$
$T_{11.7}$	$7.80 \times 10^{-2}(=)$	$2.14 \times 10^{-2}(-)$	$5.45 \times 10^{-1}(=)$
$T_{11.8}$	$1.99 \times 10^{-2}(+)$	$5.63 \times 10^{-1}(=)$	$5.27 \times 10^{-1}(=)$
$T_{11.9}$	$3.82 \times 10^{-3}(-)$	$2.64 \times 10^{-2}(-)$	$5.81 \times 10^{-1}(=)$
$T_{11.10}$	$1.99 \times 10^{-2}(+)$	$5.63 \times 10^{-1}(=)$	$5.27 \times 10^{-1}(=)$
T_{12}	$3.82 \times 10^{-3}(+)$	$1.77 \times 10^{-5}(+)$	$2.70 \times 10^{-3}(+)$
T_{13}	$7.36 \times 10^{-2}(=)$	$1.60 \times 10^{-2}(+)$	$1.01 \times 10^{-4}(+)$
Wins	8	12	7
Draws	10	5	13
Loses	3	4	1

Each p-value is followed by a "+" if Jaya2 is significantly better than the competing algorithm, "−" if significantly worse, or "=" if statistically equivalent.

– The spherical search (SS) algorithm [67], which is a recent metaheuristic where the main operations are the creation of a spherical boundary and the generation of new trial solutions on the surface of the spherical boundary.

The population size and all other algorithm-specific parameters of the aforementioned approaches were set based on the recommendations of their original papers. Table 13 summarizes the values used for these parameters.

Given the simplicity of Jaya2, it is not reasonable to expect it to outperform these complex approaches. However, the results of this section are useful for understanding how far Jaya2 is from the state-of-the-art approaches.

Concerning the CEC 2020 benchmark functions, Tables 14 and 15 show that compared to GA-MPC and SS, the performance of Jaya2 is generally better on the basic functions and comparable on the composition functions. However, L-SHADE and jSO outperformed Jaya2 on most functions. L-SHADE and jSO have an almost identical performance on this set of problems.

On the CEC 2011 real-world problems, an important result is that Jaya2 outperformed SS on all the problems and performed better than jSO on 13 problems. The performance of jSO on this benchmark suite is interesting. It seems that the jSO suffers from "overfitting" in the sense that it has been tailored to work on the IEEE CEC benchmark problems used on the competitions on single-objective optimization (e.g., CEC 2013, CEC 2017, etc.), but it does not work well on a different set of problems like CEC 2011.[3] This is similar to

3 We have used the Matlab implementation of jSO provided by Dr. Radka Polakova given that the original jSO was implemented in C++.

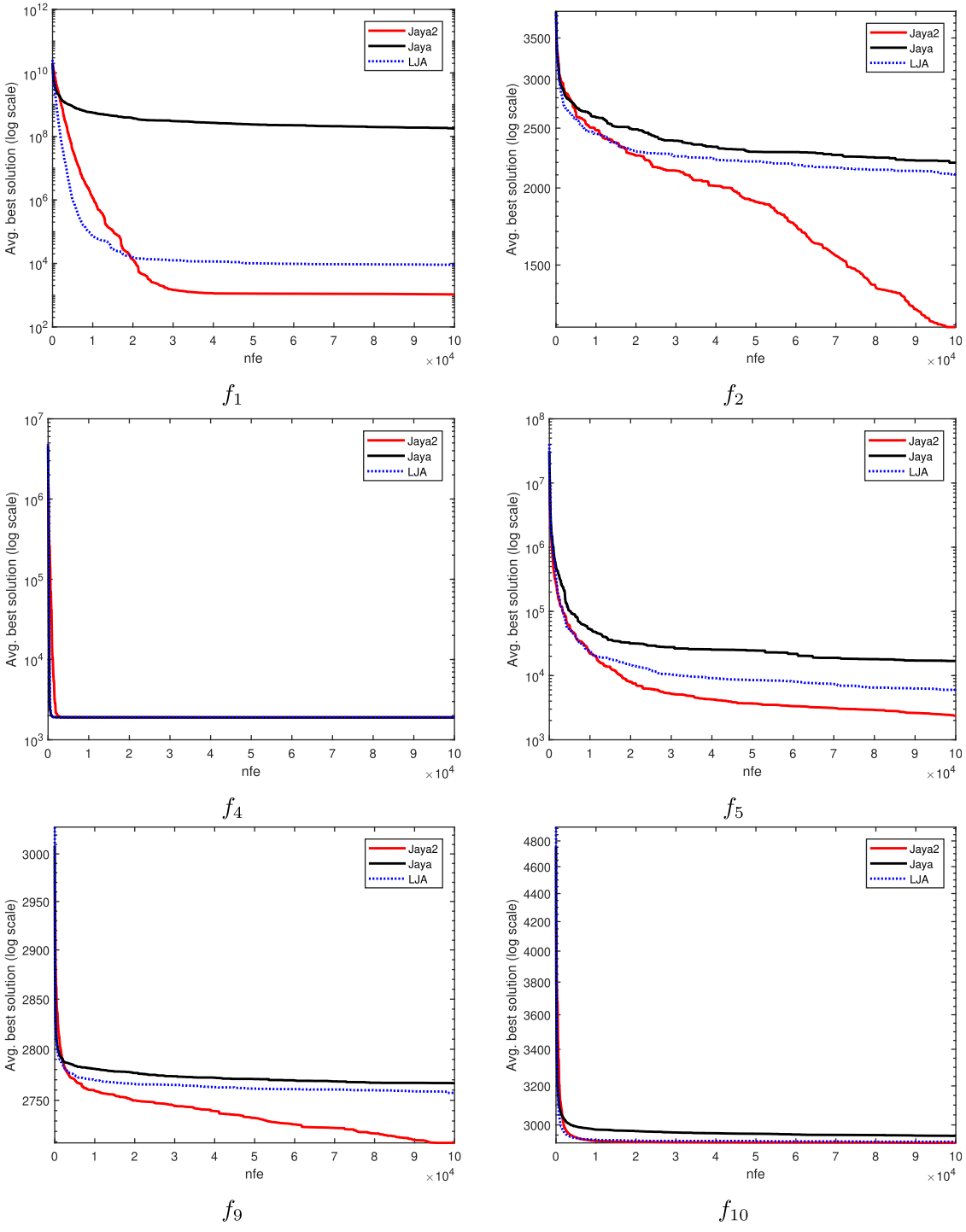

Figure 4: Best function value curves (averaged across 30 runs) of Jaya2, Jaya, and LJA for selected CEC 2020 benchmark functions ($D = 10$).

overfitting a model to work on a training set in machine learning. The overfit model often fails to generalize to a test set. Another finding is that although Jaya is algorithmically simpler than GA-MPC and L-SHADE, it outperformed them on four problems each, while performing comparably on 12 and 4 problems, respectively, being outperformed in the other cases.

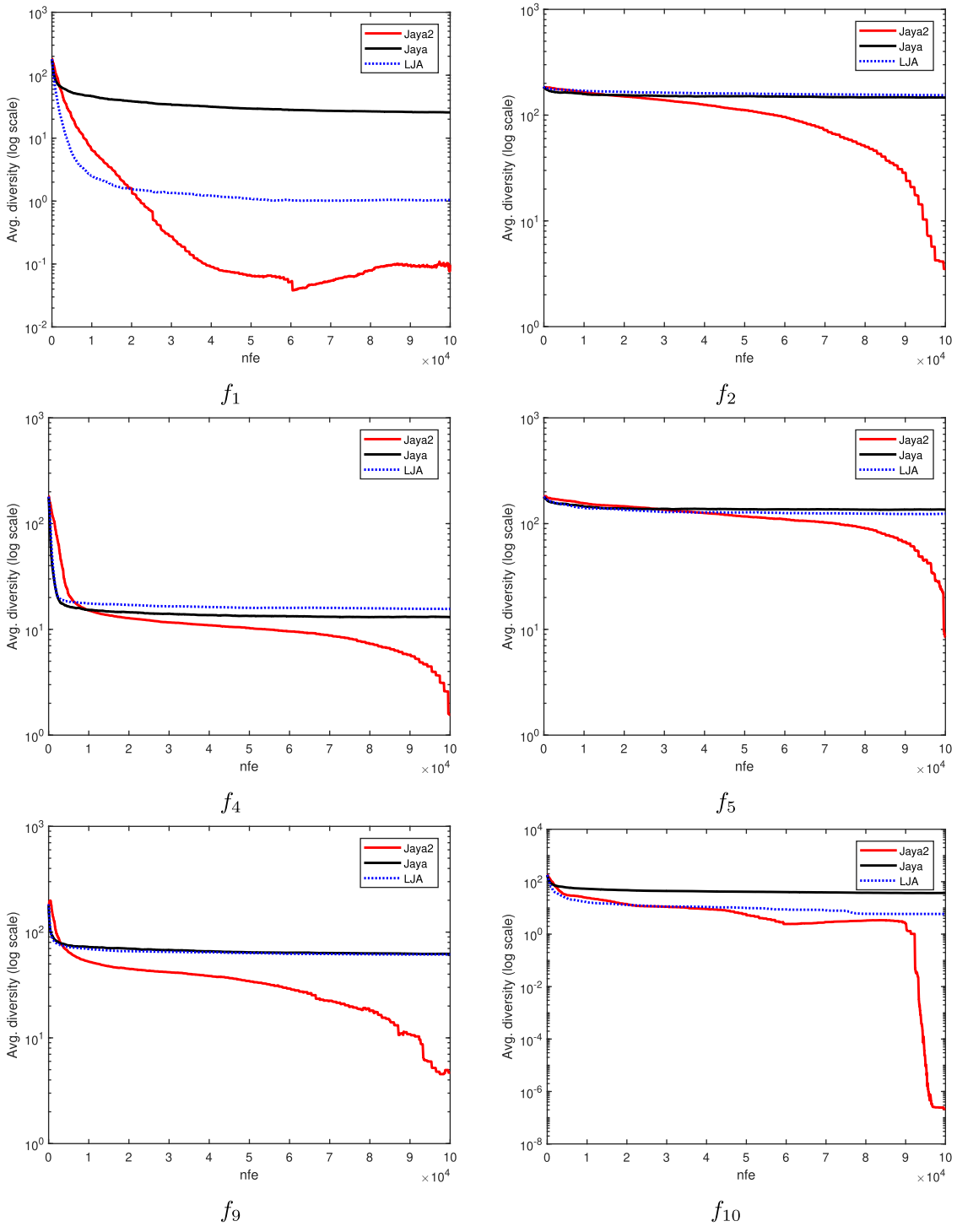

Figure 5: Diversity curves (averaged across 30 runs) of Jaya2, Jaya, and LJA for selected CEC 2020 benchmark functions ($D = 10$).

To summarize, Jaya2 was able to find on some specific problems better solutions than more complex algorithms, including top performers like GA-MPC and L-SHADE. However, we noted that these problems have different characteristics; hence, it is difficult to identify a clear trend of cases where Jaya2 performs better than its competitors.

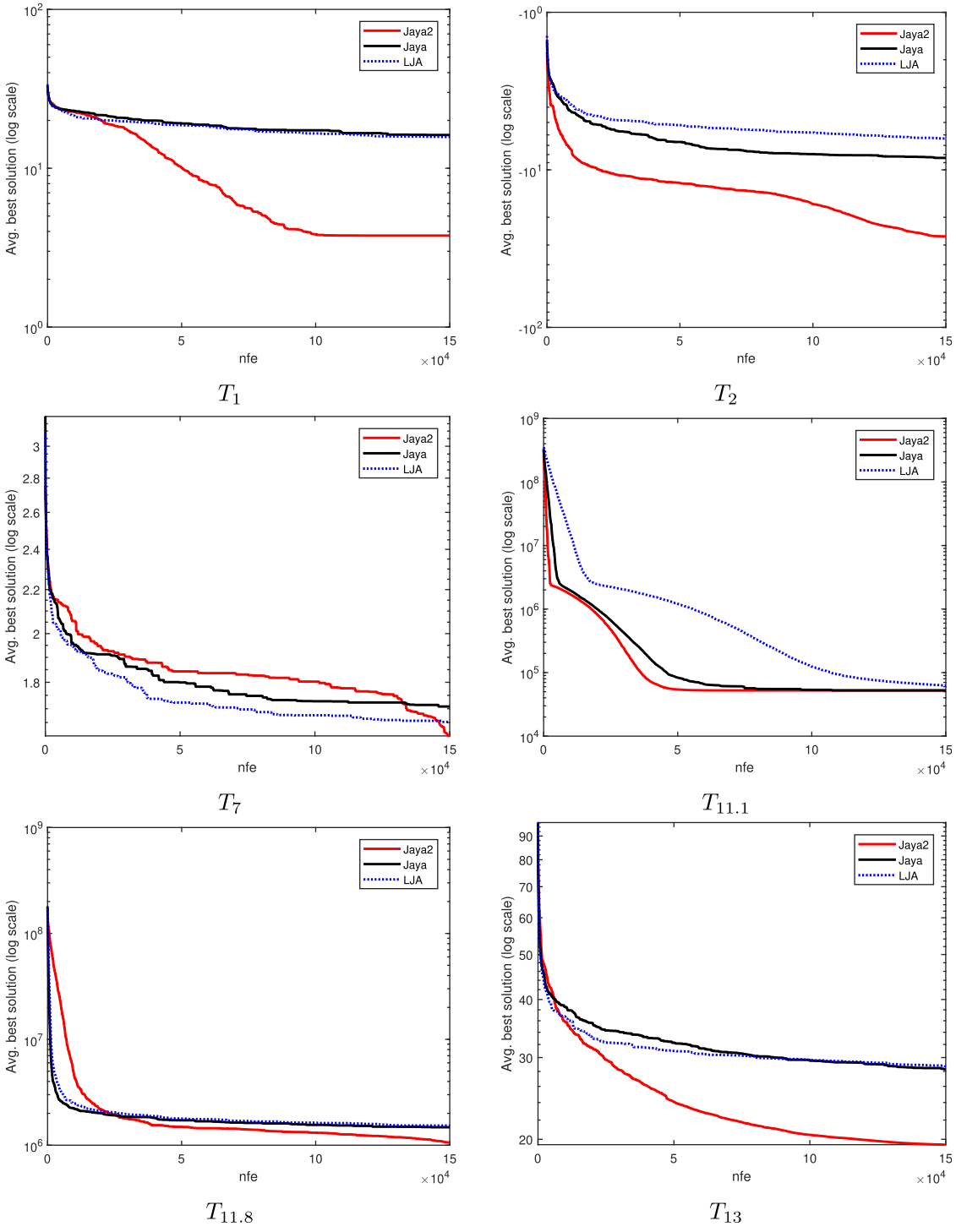

Figure 6: Best function value curves (averaged across 25 runs) of Jaya2, Jaya, and LJA for selected CEC 2011 real-world problems.

4.4 Analysis of scalability

In this section, the effect of increasing the dimensionality of the problems on the performance of the competing approaches is investigated. To do that, the dimension, D, of the 10 CEC 2020 functions is doubled, i.e., D is set to 20, which is the maximum allowed dimension for the CEC 2020 functions [56].

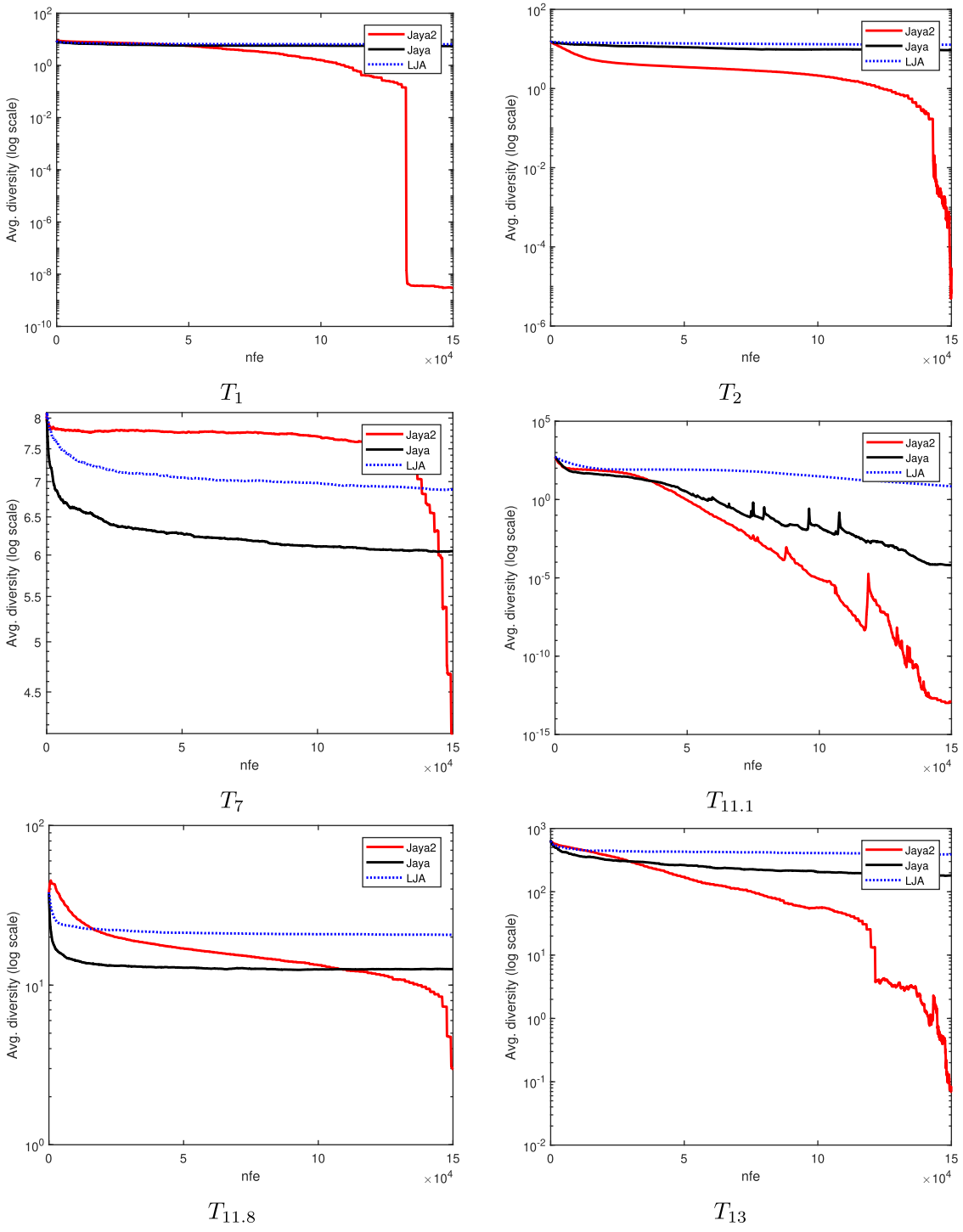

Figure 7: Diversity curves (averaged across 25 runs) of Jaya2, Jaya, and LJA for selected CEC 2011 real-world problems.

Figure 8 depicts the percentage of wins for Jaya2 relative to each of the competing algorithms for both $D = 10$ and $D = 20$. The figure shows that increasing the dimensionality of the problems generally does not degrade excessively the performance of Jaya2 (in two comparisons, namely, vs BFPA and SS, the number of wins even increases). It should be noted that once again the Jaya2 is able to outperform in several cases, most of the compared algorithms, except L-SHADE and jSO. The detailed results of these experiments are presented in the Appendix.

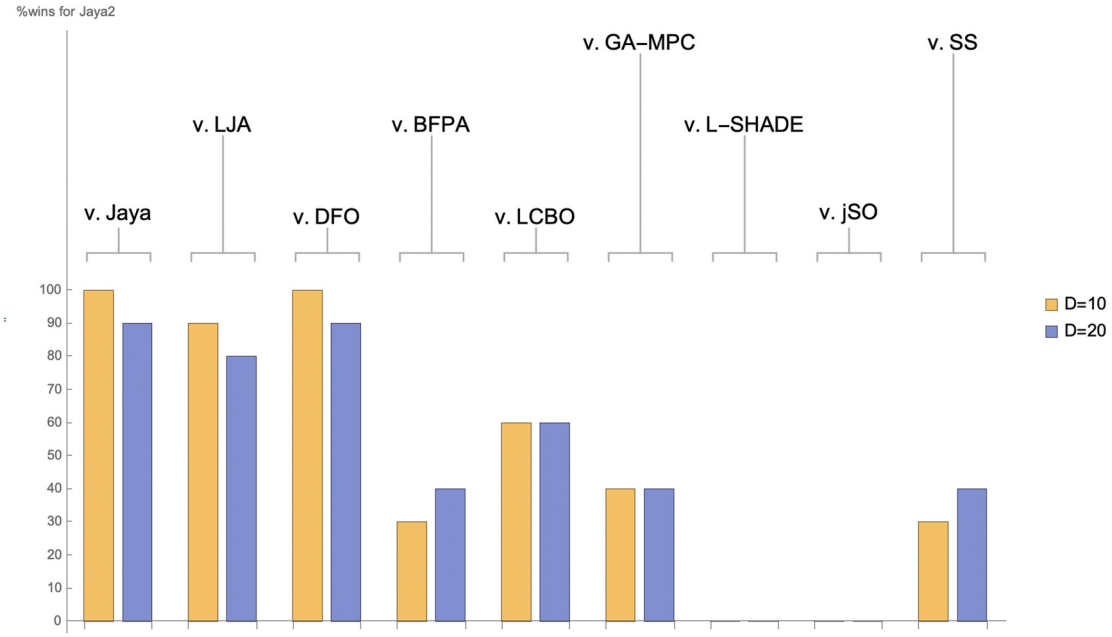

Figure 8: The percentage of wins for Jaya2 vs the other competing approaches.

4.5 Analysis of effect of the algorithmic modifications

In this section, the contribution of each algorithmic component introduced in Jaya2 (starting from the original Jaya scheme) is investigated. In particular, three Jaya2 variants are investigated:

- Jaya2 (no ring): Jaya2 that does not use the ring topology but rather uses the best and worst solutions in the population (as in the original Jaya algorithm).
- Jaya2 (fix pop): Jaya2 that uses a fixed size population (as in the original Jaya algorithm), rather than a linearly decreasing population size. In this case, the population size is fixed to 30.
- Jaya2 (equation (2)): Jaya2 that uses equation (2) (as in the original Jaya algorithm) rather than the newly proposed equation (4).

Tables 18 and 19 summarize the results of Jaya2 and its three aforementioned variants on the CEC 2020 benchmark functions. The results show that Jaya2 (in the full version described in Section 3) outperformed the three considered variants on almost all functions, with four occasional draws. Concerning the CEC 2011 real-world problems, Tables 20 and 21 confirm the importance of the three proposed modifications. More specifically, using the ring topology improved the performance on eight problems, while degrading it on only three problems. Using the population size reduction had even a more positive impact, enhancing the results on 12 problems while degrading it on only 4 problems. Finally, equation (4) yielded better results on seven problems, while losing on only one problem. Thus, the three modifications we proposed to Jaya contributed to the improved performance of Jaya2. Removing any of them degrades the performance of Jaya2.

4.6 Analysis of the algorithmic overhead

We conclude our empirical analysis with some considerations on the time efficiency of Jaya2. The algorithmic overhead characterizing Jaya2, in comparison with other algorithms that we tested in the previous sections. The algorithmic overhead is defined as the total time of an optimization run (until the end of the allotted budget of $nfe_{\max}$ function evaluations), minus the total time needed to perform the $nfe_{\max}$ function

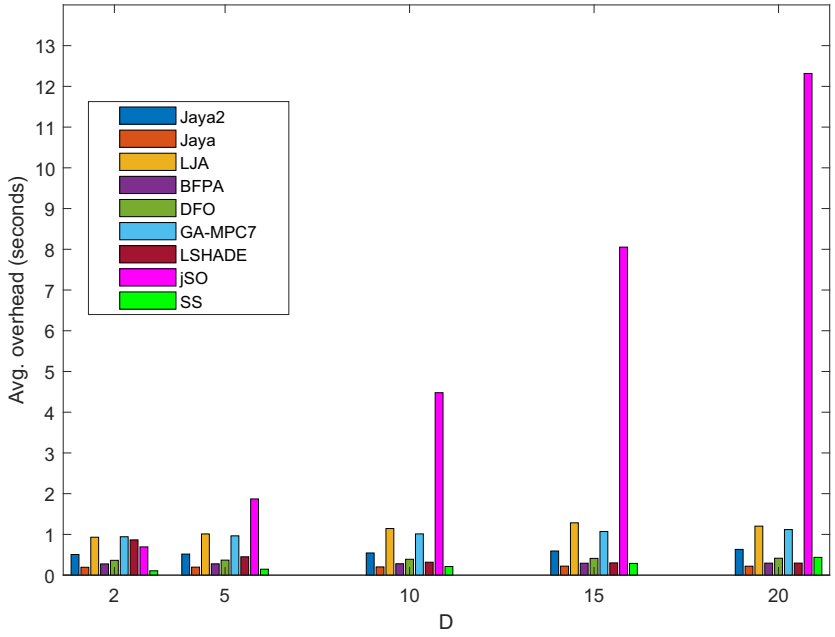

Figure 9: Average algorithmic overhead (in seconds) for the competing algorithms, over increasing dimensionality values.

evaluations. Thus, this metric measures the actual time needed by the algorithm to perform its inner operations and thus conduct the search.

The time needed to perform $nfe_{\max}$ function evaluations of a given fitness function (indicated with T_{elapsed}) is computed first. Herein, we set $nfe_{\max} = 100{,}000$, and we considered f_1 from the CEC 2020 benchmark set. This measure is repeated 30 times to have a more robust estimation.

The time needed to run an optimization process consisting of $nfe_{\max}$ function evaluations (referred to as *elapsed time* T_{elapsed}) is recorded. This measure is also repeated 30 times. Finally, the difference $T_{\text{elapsed}} - T_{\text{evals}}$, averaged over the 30 performed repetitions to return the average algorithmic overhead values, expressed in seconds, for the competing algorithms is computed. To show the effect of the dimensionality on the average algorithmic overhead, the aforementioned process was repeated by scaling up the f_1 problem. The considered numbers of design variables were 2, 5, 10, 15, and 20. The resulting average $T_{\text{elapsed}} - T_{\text{evals}}$ values, one for each algorithm and problem size, are depicted in Figure 9. It is clear that jSO is the slowest approach, which is significantly affected by the problem size. The computational overhead of Jaya2 is always less than 1 s and the increase in the dimensionality does not increase the overhead significantly (the overhead increase is less than 0.3 second). Compared to Jaya, Jaya2 has a slightly higher overhead mainly because of the use of the ring topology and the sorting process.

5 Conclusion and future work

An improved variant of the Jaya optimization algorithm was proposed in this article. The proposed variant, called Jaya2, modified Jaya by:

- Using a ring topology (similar to the one used by *lbest* PSO) to address premature convergence.
- Using a linearly decreasing population size to further balance the exploration and exploitation.
- Replacing the original update equation of Jaya (equation (2)) with one that is translation independent (equation (4)).

The proposed algorithmic modifications did not substantially increase the complexity of Jaya. Thus, Jaya2 is still easy to implement and requires no algorithm-specific parameters, besides the initial

population size $P_{\max}$ and the maximum number of function evaluations (which replaces the maximum number of generations G used in Jaya, since the population size is not fixed anymore). These are two very desirable features for a metaheuristic, since, on the one hand, the parameter tuning can be a tedious and time-consuming task, while, on the other hand, having a simple algorithmic structure facilitates its understanding and broad application. Moreover, the translation dependency problem of the original Jaya has been demonstrated and fixed with the alternative update equation.

In addition, we showed that Jaya2 is able to perform well on many hard-to-solve optimization problems. In particular, we tested Jaya2 on the CEC 2020 benchmark functions and the CEC 2011 real-world problems and compared it with nine other optimization approaches (ranging from very simple approaches, like Jaya and DFO, to complex approaches like L-SHADE, jSO, and SS). Our numerical results show that Jaya2 outperforms Jaya and its recently proposed variant LJA on almost all tested problems. Moreover, it performed better than DFO, BFPA, and LCBO, which are comparably simple approaches. As expected, however, the proposed approach could not outperform the winners of the IEEE CEC 2011 and IEEE CEC 2014 competitions, respectively, GA-MPC and L-SHADE, although it outperformed them on some problems. Moreover, we found that Jaya2 performed worse than jSO on most CEC 2020 functions but outperformed it on most CEC 2011 problems. This seems to suggest that jSO suffers from "overfitting," i.e., it works very well on certain types of problems, but it may fail to generalize. Overall, apart from GA-MPC and L-SHADE, Jaya2 performed on average better on all the tested problems than all the other compared optimizers, including way more complex recent approaches like jSO and SS.

To summarize, one of the main strengths of Jaya2 is its simplicity, in terms of the implementation and parameter tuning. It can be implemented with few lines of code, and at the same time, it does not require any parameter tuning. Moreover, its performance is good on many problems. In this respect, it inherits the fundamental design principles of the original Jaya algorithm, while consistently improving its performance. Thus, Jaya2 can be very useful for practitioners with limited programming skills or limited time and/or knowledge to tune the parameters of an optimizer.

However, the simplicity of Jaya2 may not allow it to outperform more complex and fine-tuned metaheuristics on certain types of problems like those from the IEEE CEC 2020 benchmark. Moreover, Jaya2 is slightly slower than Jaya because of the need for sorting the population and arranging it as a ring.

Future research may investigate other kinds of neighborhood topology (e.g., the von Neumann topology [48]), as well as different mechanisms for adjusting the population size and different out-of-bounds correction mechanisms, such as the nonlinear mechanism used in ref. [68]. Furthermore, future research will investigate how the modifications introduced here in the context of Jaya (in particular, the ring topology and the population size reduction mechanism) can be used within (and affect) other SI algorithms.

Another interesting area for investigation would be using Jaya2 for embedded systems (with limited resources). In Section 4.6, we have shown that Jaya2 has a small computational overhead. In addition, Jaya2 requires a relatively small memory of $O(P_{\max} \cdot D + P_{\max})$ (unlike L-SHADE and jSO, that require additional memory for the archive). Along with an implementation of few lines of code, these features may make it suitable for embedded systems. One possible way to reduce the memory requirements of Jaya2 by almost a factor of 2, is to use an *asynchronous* version of it. In the original Jaya2 (as in Jaya), the solutions are updated after the whole population performance is evaluated. Hence, there is a need to store both the original and updated solutions (along with their fitnesses). This is the *synchronous* version of Jaya2. In the asynchronous alternative of Jaya2, each solution is updated as soon as its own performance has been evaluated. Thus, there is no need to maintain both the original and updated solutions. According to our preliminary experiments, both versions performs comparably on most problems used in this study. Thus, for systems with limited memory, asynchronous Jaya2 can be used.

Funding information: Not applicable.

Author contributions: Both authors contributed equally to the manuscript. Both authors read and approved the final manuscript.

Conflict of interest: Not applicable.

Code availability: The code is available upon request.

Data availability statement: The raw data are available upon request.

References

[1] Del Ser J, Osaba E, Molina D, Yang XS, Salcedo-Sanz S, Camacho D, et al. Bio-inspired computation: Where we stand and what's next. Swarm Evol Comput. 2019;48:220–50.
[2] Heidari AA, Mirjalili S, Faris H, Aljarah I, Mafarja M, Chen H. Harris hawks optimization: Algorithm and applications. Future Gener Comput Syst. 2019;97:849–72.
[3] Li S, Chen H, Wang M, Heidari AA, Mirjalili S. Slime mould algorithm: a new method for stochastic optimization. Future Gener Comput Syst. 2020;111:300–23.
[4] Yang Y, Chen H, Heidari AA, Gandomi AH. Hunger games search: visions, conception, implementation, deep analysis, perspectives, and towards performance shifts. Expert Syst Appl. 2021;177:114864.
[5] Wang GG, Deb S, Cui Z. Monarch butterfly optimization. Neural Comput Appl. 2019;31(7):1995–2014.
[6] Wang GG. Moth search algorithm: a bio-inspired metaheuristic algorithm for global optimization problems. Memetic Computing. 2018;10(2):151–64.
[7] Ferreira MP, Rocha ML, SilvaNeto AJ, Sacco WF. A constrained ITGO heuristic applied to engineering optimization. Expert Syst Appl. 2018;110:106–24.
[8] Nabil E. A modified flower pollination algorithm for global optimization. Expert Syst Appl. 2016;57:192–203.
[9] Ayala HVH, dos Santos FM, Mariani VC, dos Santos Coelho L. Image thresholding segmentation based on a novel beta differential evolution approach. Expert Syst Appl. 2015;42(4):2136–42.
[10] Wang L, Xiong Y, Li S, Zeng YR. New fruit fly optimization algorithm with joint search strategies for function optimization problems. Knowledge-Based Syst. 2019;176:77–96. https://www.sciencedirect.com/science/article/pii/S0950705119301534.
[11] Wang Z, Zeng YR, Wang S, Wang L. Optimizing echo state network with backtracking search optimization algorithm for time series forecasting. Eng Appl Artif Intell. 2019;81:117–32. https://www.sciencedirect.com/science/article/pii/S0952197619300326.
[12] Zhang Y. Backtracking search algorithm with specular reflection learning for global optimization. Knowledge-Based Syst. 2021;212:106546. https://www.sciencedirect.com/science/article/pii/S0950705120306754.
[13] dos Santos Coelho L, Mariani VC. Use of chaotic sequences in a biologically inspired algorithm for engineering design optimization. Expert Syst Appl. 2008;34(3):1905–13.
[14] Kayhan AH, Ceylan H, Ayvaz MT, Gurarslan G. PSOLVER: A new hybrid particle swarm optimization algorithm for solving continuous optimization problems. Expert Syst Appl. 2010;37(10):6798–808.
[15] Chen MR, Huang YY, Zeng GQ, Lu KD, Yang LQ. An improved bat algorithm hybridized with extremal optimization and Boltzmann selection. Expert Syst Appl. 2021;175:114812.
[16] Sörensen K. Metaheuristics the metaphor exposed. Int Trans Oper Res. 2015;22(1):3–18.
[17] Piotrowski AP, Napiorkowski JJ. Some metaheuristics should be simplified. Inform Sci. 2018;427:32–62.
[18] Iacca G, Neri F, Mininno E, Ong YS, Lim MH. Ockham's Razor in memetic computing: three stage optimal memetic exploration. Inform Sci. 2012;188:17–43.
[19] Iacca G, Caraffini F, Neri F. Compact differential evolution light: high performance despite limited memory requirement and modest computational overhead. J Comput Sci Tech. 2012;27(5):1056–76.
[20] Iacca G, Caraffini F, Neri F. Memory-saving memetic computing for path-following mobile robots. Appl Soft Comput. 2013;13(4):2003–16.
[21] Iacca G. Distributed optimization in wireless sensor networks: an island-model framework. Soft Comput. 2013;17(12):2257–77.
[22] Price K, Storn RM, Lampinen JA. Differential evolution: a practical approach to global optimization. Springer Berlin, Heidelberg: Springer Science & Business Media; 2006.
[23] Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by simulated annealing. Science. 1983;220(4598):671–80.
[24] Xinchao Z. Simulated annealing algorithm with adaptive neighborhood. Appl Soft Comput. 2011;11(2):1827–36.
[25] Zhou J, Ji Z, Shen L. Simplified intelligence single particle optimization based neural network for digit recognition. In: Chinese Conference on Pattern Recognition. IEEE; 2008. p. 1–5.
[26] Iacca G, Caraffini F, Neri F, Mininno E. Single particle algorithms for continuous optimization. In: Congress on evolutionary computation (CEC). IEEE; 2013. p. 1610–7.

[27] Iacca G, Bakker FL, Wörtche H. Real-time magnetic dipole detection with single particle optimization. Appl Soft Comput. 2014;23:460–73.
[28] Squillero G, Tonda A. Divergence of character and premature convergence: a survey of methodologies for promoting diversity in evolutionary optimization. Inform Sci. 2016;329:782–99.
[29] Rao RV. Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. Int J Ind Eng Comput. 2016;7(1):19–34.
[30] Farah A, Belazi A. A novel chaotic Jaya algorithm for unconstrained numerical optimization. Nonlinear Dyn. 2018;93(3):1451–80.
[31] Iacca G, dos Santos Junior V, de Melo V. An improved Jaya optimization algorithm with Lévy flight. Expert Syst Appl. 2021;165:113902.
[32] Chakraborty S, Saha AK, Sharma S, Sahoo SK, Pal G. Comparative performance analysis of differential evolution variants on engineering design problems. J Bionic Eng. 2022;19:1140–60.
[33] Kennedy J, Eberhart R. Particle swarm optimization. In: International Joint Conference on Neural Networks (IJCNN). IEEE; 1995. p. 1942–8.
[34] Nordmoen J, Nygaard TF, Samuelsen E, Glette K. On restricting real-valued genotypes in evolutionary algorithms. 2020. arXiv: http://arXiv.org/abs/arXiv:200509380.
[35] Kononova AV, Caraffini F, Bäck T, Differential evolution outside the box. 2020. arXiv: http://arXiv.org/abs/arXiv:200410489.
[36] Rao RV, Saroj A. A self-adaptive multi-population based Jaya algorithm for engineering optimization. Swarm Evol Comput. 2017;37:1–26.
[37] Bekdaş G. Optimum design of post-tensioned axially symmetric cylindrical walls using novel hybrid metaheuristic methods. Struct Design Tall Spec Build. 2019;28(1):e1550.
[38] Gao K, Zhang Y, Sadollah A, Lentzakis A, Su R. Jaya, harmony search and water cycle algorithms for solving large-scale real-life urban traffic light scheduling problem. Swarm Evol Comput. 2017;37:58–72.
[39] Wang S, Rao RV, Chen P, Zhang Y, Liu A, Wei L. Abnormal breast detection in mammogram images by feed-forward neural network trained by Jaya algorithm. Fundam Inform. 2017;151(1–4):191–211.
[40] Zhang Y, Yang X, Cattani C, Rao RV, Wang S, Phillips P. Tea category identification using a novel fractional Fourier entropy and Jaya algorithm. Entropy. 2016;18(3):77.
[41] Aslan M, Gunduz M, Kiran MS. JayaX: Jaya algorithm with xor operator for binary optimization. Appl Soft Comput. 2019;82:105576.
[42] Rao RV, Saroj A. A self-adaptive multi-population based Jaya algorithm for engineering optimization. Swarm Evol Comput. 2017;37:1–26.
[43] Rao RV, Saroj A. An elitism-based self-adaptive multi-population Jaya algorithm and its applications. Soft Comput. 2019;23(12):4383–406.
[44] Rao RV, Keesari HS, Oclon P, Taler J. An adaptive multi-team perturbation-guiding Jaya algorithm for optimization and its applications. Eng Comput. 2020;36(1):391–419.
[45] Awadallah M, Al-Betar M, Doush I. cJAYA: cellular JAYA algorithm. In: The 2020 International Conference on Promising Electronic Technology (ICPET). IEEE; 2020. p. 155–60.
[46] Von neumann J, Burks A. Theory of self-reproducing automata. IEEE Trans Neural Networks. 1966;5(1):3–14.
[47] Rao RV. Jaya optimization algorithm and its variants. In: Jaya: An advanced optimization algorithm and its engineering applications. Springer; 2019. p. 9–58.
[48] Kennedy J, Mendes R. Population structure and particle swarm performance. In: Congress on evolutionary computation (CEC). vol. 2. IEEE; 2002. p. 1671–6.
[49] Lynn N, Ali MZ, Suganthan PN. Population topologies for particle swarm optimization and differential evolution. Swarm Evolut Comput. 2018;39:24–35.
[50] Omran MG, Engelbrecht AP, Salman A. Using the ring neighborhood topology with self-adaptive differential evolution. In: International Conference on Natural Computation. Springer; 2006. p. 976–9.
[51] Salman A, Engelbrecht AP, Omran MG. Empirical analysis of self-adaptive differential evolution. European J Operat Res. 2007;183(2):785–804.
[52] Omran MG, Engelbrecht AP, Salman A. Bare bones differential evolution. European J Operat Res. 2009;196(1):128–39.
[53] Iacca G, Mallipeddi R, Mininno E, Neri F, Suganthan PN. Super-fit and population size reduction in compact differential evolution. In: Workshop on Memetic Computing (MC). IEEE; 2011. p. 1–8.
[54] Tanabe R, Fukunaga A. Improving the search performance of SHADE using linear population size reduction. In: Congress on Evolutionary Computation (CEC). IEEE; 2014. p. 1658–65.
[55] Clerc M.. (Multi-agents multi-strategies optimiser); 2021. Working paper or preprint.
[56] Yue D, Price K, P S, Liang J, Ali M, Qu B, Problem definitions and evaluation criteria for CEC 2020 competition on single objective bound constrained numerical optimization. Zhengzhou University (China), Nanyang Technological University (Singapore); 2019.
[57] Das S, Suganthan P. Problem definitions and evaluation criteria for CEC 2011 competition on testing evolutionary algorithms on real world optimization problems. Kolkata: Jadavpur University, Nanyang Technological University; 2010.

[58] Wilcoxon F. Individual comparisons by ranking methods. Biometrics Bulletin. 1945;1(6):80–83.
[59] Poláková R, Tvrdik J, Bujok P. Adaptation of population size according to current population diversity in differential evolution. In: Proceedings of the IEEE 2017 Symposium Series on Computational Intelligence (SSCI). IEEE; 2017. p. 2627–34.
[60] al Rifaie M. Dispersive flies optimisation. In: Federated Conference on Computer Science and Information Systems (FedCSIS). Vol. 2. IEEE; 2014. p. 529–38.
[61] Kopciewicz P, Lukasik S. Exploiting flower constancy in flower pollination algorithm: improved biotic flower pollination algorithm and its experimental evaluation. Neural Comput Appl. 2020;32:11999–2010.
[62] Khatri A, Gaba A, Rana K, Kumar V. A novel life choice-based optimizer. Soft Comput. 2020;24:9121–41.
[63] Elsayed SM, Sarker RA, Essam DL. GA with a new multi-parent crossover for solving IEEE-CEC2011 competition problems. In: Congress on Evolutionary Computation (CEC). IEEE; 2011. p. 1034–40.
[64] Brest J, Maucec M, Boskovic B. Single objective real- parameter optimization: algorithm jSO. In: Congress on Evolutionary Computation (CEC). IEEE; 2017. p. 1311–8.
[65] Brest J, Maucec M, Boskovic B. iL-SHADE:Improved L-SHADE algorithm for single objective real-parameter optimization. In: Congress on Evolutionary Computation (CEC). IEEE; 2016. p. 1188–95.
[66] Fister I, Brest J, Iglesias A, Galvez A, Deb S, Fister I. On selection of a benchmark by determining the algorithms' qualities. vol. 9. IEEE Access; 2021. p. 51166–78. doi: 10.1109/ACCESS.2021.3058285.
[67] Kumar A, Misra R, Singh D, Mishra S, Das S. The spherical search algorithm for bound-constrained global optimization problems. Appl Soft Comput. 2019;85:105734.
[68] Tzanetos A, Dounias G. A new metaheuristic method for optimization: sonar inspired optimization. In: International Conference on Engineering Applications of Neural Networks. Springer; 2017. p. 417–28.

Appendix

Detailed results on the CEC 2020 benchmark

Tables A1, A2, A3, A4, A5, A6.

Table A1: The CEC 2020 ($D = 20$) median function values achieved by Jaya2, Jaya, and LJA over 30 runs and 200,000 function evaluations

Function	Jaya2	Jaya	LJA
f_1	$\mathbf{3.86 \times 10^2}$	1.63×10^9	8.53×10^3
f_2	$\mathbf{1.47 \times 10^3}$	4.47×10^3	4.49×10^3
f_3	$\mathbf{7.39 \times 10^2}$	8.64×10^2	8.23×10^2
f_4	$\mathbf{1.90 \times 10^3}$	1.91×10^3	1.91×10^3
f_5	$\mathbf{3.20 \times 10^4}$	4.45×10^5	8.87×10^4
f_6	$\mathbf{1.72 \times 10^3}$	$\mathbf{1.72 \times 10^3}$	$\mathbf{1.72 \times 10^3}$
f_7	$\mathbf{5.37 \times 10^3}$	2.68×10^5	2.27×10^4
f_8	$\mathbf{2.30 \times 10^3}$	2.47×10^3	4.63×10^3
f_9	$\mathbf{2.81 \times 10^3}$	2.92×10^3	2.90×10^3
f_{10}	$\mathbf{2.91 \times 10^3}$	2.96×10^3	$\mathbf{2.91 \times 10^3}$

The boldface indicates the lowest median function value per function.

Table A2: The CEC 2020 ($D = 20$) statistical analysis showing the Wilcoxon test p-value

Function	Jaya	LJA
f_1	$1.73 \times 10^{-6}(+)$	$1.92 \times 10^{-6}(+)$
f_2	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$
f_3	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$
f_4	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$
f_5	$1.73 \times 10^{-6}(+)$	$5.79 \times 10^{-5}(+)$
f_6	$1.00 \times 10^{0}(=)$	$1.00 \times 10^{0}(=)$
f_7	$1.73 \times 10^{-6}(+)$	$2.35 \times 10^{-6}(+)$
f_8	$1.73 \times 10^{-6}(+)$	$2.56 \times 10^{-6}(+)$
f_9	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$
f_{10}	$1.15 \times 10^{-4}(+)$	$1.25 \times 10^{-1}(=)$
Wins	9	8
Draws	1	2
Loses	0	0

Each p-value is followed by a "+" if Jaya2 is significantly better than the competing algorithm, "−" if significantly worse, or "=" if statistically equivalent.

Table A3: The CEC 2020 ($D = 20$) median function values achieved by Jaya2, DFO, BFPA, and LCBO over 30 runs and 200,000 function evaluations

Function	Jaya2	DFO	BFPA	LCBO
f_1	3.41×10^{2}	$1.96 \times 10^{\mathbf{10}}$	$\mathbf{1.00 \times 10^{2}}$	5.41×10^{2}
f_2	$\mathbf{1.55 \times 10^{3}}$	5.82×10^{3}	3.07×10^{3}	2.40×10^{3}
f_3	$\mathbf{7.38 \times 10^{2}}$	2.03×10^{3}	8.59×10^{2}	7.47×10^{2}
f_4	$\mathbf{1.90 \times 10^{3}}$	4.79×10^{6}	1.91×10^{3}	1.90×10^{3}
f_5	3.74×10^{4}	1.60×10^{7}	$\mathbf{2.19 \times 10^{3}}$	1.19×10^{5}
f_6	$\mathbf{2.05 \times 10^{3}}$	$\mathbf{2.05 \times 10^{3}}$	$\mathbf{2.05 \times 10^{3}}$	$\mathbf{2.05 \times 10^{3}}$
f_7	5.40×10^{3}	1.50×10^{6}	$\mathbf{2.43 \times 10^{3}}$	4.35×10^{4}
f_8	$\mathbf{2.30 \times 10^{3}}$	6.60×10^{3}	2.30×10^{3}	$\mathbf{2.30 \times 10^{3}}$
f_9	$\mathbf{2.81 \times 10^{3}}$	3.03×10^{3}	2.87×10^{3}	$\mathbf{2.81 \times 10^{3}}$
f_{10}	2.91×10^{3}	6.95×10^{3}	$\mathbf{2.91 \times 10^{3}}$	2.97×10^{3}

The boldface indicates the lowest median function value per function.

Table A4: The CEC 2020 ($D = 20$) statistical analysis showing the Wilcoxon test p-value

Function	DFO	BFPA	LCBO
f_1	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(-)$	$6.29 \times 10^{-1}(=)$
f_2	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$
f_3	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$	$4.90 \times 10^{-4}(+)$
f_4	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$	$9.84 \times 10^{-3}(+)$
f_5	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(-)$	$2.35 \times 10^{-6}(+)$
f_6	$1.00 \times 10^{0}(=)$	$1.00 \times 10^{0}(=)$	$1.00 \times 10^{0}(=)$
f_7	$1.73 \times 10^{-6}(+)$	$1.92 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(+)$
f_8	$1.73 \times 10^{-6}(+)$	$1.73 \times 10^{-6}(+)$	$1.27 \times 10^{-1}(=)$
f_9	$1.73 \times 10^{-6}(+)$	$2.06 \times 10^{-1}(=)$	$4.28 \times 10^{-1}(=)$
f_{10}	$1.73 \times 10^{-6}(+)$	$4.86 \times 10^{-5}(-)$	$2.58 \times 10^{-3}(+)$
Wins	9	4	6
Draws	1	2	4
Loses	0	4	0

Each p-value is followed by a "+" if Jaya2 is significantly better than the competing algorithm, "−" if significantly worse, or "=" if statistically equivalent.

Table A5: The CEC 2020 ($D = 20$) median function values achieved by Jaya2, GA-MPC, L-SHADE, jSO, and SS over 30 runs and 200,000 function evaluations

Function	Jaya2	GA-MPC	L-SHADE	jSO	SS
f_1	2.50×10^2	1.10×10^3	$\mathbf{1.00 \times 10^2}$	$\mathbf{1.00 \times 10^2}$	$\mathbf{1.00 \times 10^2}$
f_2	1.36×10^3	1.58×10^3	$\mathbf{1.11 \times 10^3}$	$\mathbf{1.11 \times 10^3}$	4.68×10^3
f_3	7.37×10^2	7.40×10^2	$\mathbf{7.22 \times 10^2}$	$\mathbf{7.22 \times 10^2}$	8.08×10^2
f_4	1.90×10^3	1.90×10^3	$\mathbf{1.90 \times 10^3}$	$\mathbf{1.90 \times 10^3}$	1.91×10^3
f_5	2.93×10^4	4.61×10^3	$\mathbf{1.72 \times 10^3}$	$\mathbf{1.72 \times 10^3}$	2.04×10^3
f_6	$\mathbf{1.62 \times 10^3}$	$\mathbf{1.62 \times 10^3}$	$\mathbf{1.62 \times 10^3}$	$\mathbf{1.62 \times 10^3}$	$\mathbf{1.62 \times 10^3}$
f_7	5.32×10^3	2.30×10^3	$\mathbf{2.10 \times 10^3}$	$\mathbf{2.10 \times 10^3}$	2.23×10^3
f_8	2.30×10^3	$\mathbf{2.30 \times 10^3}$	$\mathbf{2.30 \times 10^3}$	$\mathbf{2.30 \times 10^3}$	$\mathbf{2.30 \times 10^3}$
f_9	2.81×10^3	2.82×10^3	$\mathbf{2.80 \times 10^3}$	$\mathbf{2.80 \times 10^3}$	2.88×10^3
f_{10}	2.95×10^3	2.96×10^3	$\mathbf{2.91 \times 10^3}$	$\mathbf{2.91 \times 10^3}$	$\mathbf{2.91 \times 10^3}$

The boldface indicates the lowest median function value per function.

Table A6: The CEC 2020 ($D = 20$) statistical analysis showing the Wilcoxon test p-value

Function	GA-MPC	L-SHADE	jSO	SS
f_1	$4.39 \times 10^{-3}(+)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(-)$
f_2	$9.78 \times 10^{-2}(=)$	$3.88 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(+)$
f_3	$1.53 \times 10^{-1}(=)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(+)$
f_4	$3.33 \times 10^{-2}(+)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(+)$
f_5	$1.49 \times 10^{-5}(-)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(-)$
f_6	$1.00 \times 10^{0}(=)$	$1.00 \times 10^{0}(=)$	$1.00 \times 10^{0}(=)$	$1.00 \times 10^{0}(=)$
f_7	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(-)$
f_8	$4.75 \times 10^{-2}(+)$	$2.44 \times 10^{-4}(-)$	$4.88 \times 10^{-4}(-)$	$4.88 \times 10^{-4}(-)$
f_9	$6.89 \times 10^{-5}(+)$	$1.73 \times 10^{-6}(-)$	$2.35 \times 10^{-6}(-)$	$1.73 \times 10^{-6}(+)$
f_{10}	$2.54 \times 10^{-1}(=)$	$3.88 \times 10^{-4}(-)$	$3.88 \times 10^{-4}(-)$	$3.88 \times 10^{-4}(-)$
Wins	4	0	0	4
Draws	4	1	1	1
Loses	2	9	9	5

Each p-value is followed by a "+" if Jaya2 is significantly better than the competing algorithm, "−" if significantly worse, or "=" if statistically equivalent.