applied sciences

MDPI

Article

Semi-Steady-State Jaya Algorithm for Optimization

Uday K. Chakraborty

Department of Computer Science, University of Missouri, St. Louis, MO 63121, USA; chakrabortyu@umsl.edu; Tel.: +1-314-516-6339

Received: 22 June 2020; Accepted: 27 July 2020; Published: 4 August 2020

Abstract: The Jaya algorithm is arguably one of the fastest-emerging metaheuristics amongst the newest members of the evolutionary computation family. The present paper proposes a new, improved Jaya algorithm by modifying the update strategies of the best and the worst members in the population. Simulation results on a twelve-function benchmark test-suite and a real-world problem show that the proposed strategy produces results that are better and faster in the majority of cases. Statistical tests of significance are used to validate the performance improvement.

Keywords: optimization; jaya algorithm; heuristic; evolutionary computation; machine learning

1. Introduction

For optimization of computationally hard problems and of problems that are mathematically intractable, machine-learning-based strategies such as evolutionary computation (EC) [1] and artificial neural network (ANN) [2] have seen significant success in numerous application areas. The "no-free-lunch theorem" [3] tells us that, theoretically, over all possible optimization functions, all algorithms perform equally well. In practice, however, for specific problems (particularly, hard problems), the need for better and still better algorithms (and heuristics) remains.

The Jaya algorithm [4], one of the newest members of the evolutionary computation family, has seen remarkable success across a wide variety of applications in continuous optimization. Jaya's success can arguably be attributed to the following two features: (a) it requires very few algorithm parameters, and (b) compared to most of its EC-cousins, Jaya is extremely simple to implement. A user of the Jaya algorithm has to decide on suitable values for only two parameters—population size and the number of iterations (generations). Because any population-based algorithm (or heuristic) must have a population size, and because the user of any algorithm/heuristic must have an idea of when to stop the process, it can be argued that the population size and the stopping condition are two fundamental attributes of any population-based heuristic and that the Jaya algorithm is parameterless. In this paper, we present an algorithm that improves over the Jaya algorithm by modifying the search strategy, without compromising on the above two qualities. The improved algorithm uses new update strategies for the best and the worst members in the population. The comparative performance of Jaya and the proposed method is studied empirically on a twelve-function benchmark test-suite as well as on a real-world problem from fuel cell stack design optimization. The improvement in performance afforded by the proposed algorithm is validated with statistical tests of significance. (Technically, Jaya is not an algorithm; it is a heuristic. However, following common practice in the evolutionary computation community, we continue to refer to it as an algorithm in this paper.)

The remainder of this paper is organized as follows. A very brief outline of some of the most interesting previous work on the Jaya algorithm is presented in Section 2. Section 3 presents the proposed algorithm. Simulation results and statistical tests for performance analysis are presented in Section 4. Finally, conclusions are drawn in Section 5.

2. A Brief Overview of Previous Work on Jaya

A variation of the standard Jaya algorithm is presented in the multi-team perturbation-guiding Jaya (MTPG-Jaya) [5] where several "teams" explore the search space, with the same population being used by each team, while the "perturbations" governing the progression of the teams are different. The MTPG-Jaya was applied to the layout optimization problem of a wind farm. The Jaya algorithm was originally designed for continuous (real-valued) optimization, and most of Jaya's applications to date have been in the continuous domain. A binary version of Jaya, however, was proposed in [6], where the authors borrowed (from [7]) the idea of combining particle swarm optimization with angle modulation and adapted that idea for Jaya. The binary Jaya was applied to feature selection in [6]. Modifications to the standard Jaya algorithm include a self-adaptive multi-population-based Jaya algorithm that was applied to entropy generation minimization of a plate-fin heat exchanger [8], a multi-objective Jaya algorithm that was applied to waterjet machining process optimization [9], and a hybrid parallel Jaya algorithm for a multi-core environment [10]. Application areas of the Jaya algorithm have included such diverse fields as pathological brain detection systems [11], flow-shop scheduling [12], maximum power point tracking problems in photovoltaic systems [13], identification and monitoring of electroencephalogram-based brain-computer interface for motor imagery tasks [14], and traffic signal control [15].

3. The Proposed Algorithm

The new algorithm is presented in Algorithm 1 where, without loss of generality, an array representation with conventional indexed access is assumed for the members (individuals) of a population. At each generation, we examine the individuals in the population one by one, in sequence, conditionally replacing each with a newly created individual. A new individual is created from the current individual by using the best individual, the worst individual, and two random numbers—each chosen uniformly randomly in (0, 1]—per problem parameter (variable). The generation of the new individual x^{new}, given the current individual x^{current}, is described by the following equation (x^{new}, x^{current}, x^{best} and x^{worst} are each a d-component vector):

$$x_i^{\text{new}} = x_i^{\text{current}} + r_{t,i,1}(x_i^{\text{best}} - |x_i^{\text{current}}|) - r_{t,i,2}(x_i^{\text{worst}} - |x_i^{\text{current}}|)$$

where x_i, i = 1 to d, represent the d parameters (variables) to be optimized, $r_{t,i,1}$ and $r_{t,i,2}$ are each a random number in (0.0, 1.0], t indicates the iteration (generation) number, x^{best} and x^{worst} represent, respectively, the best and the worst individual in the population at the time of the creation of x^{new} from x^{current}. When x_i^{new} falls outside its problem-specified lower or upper bound, it is clamped at the appropriate bound.

In the original Jaya algorithm, the new individual replaces the current individual only if it (the former) is better than the latter. The present algorithm, however, accepts the new individual if it is at least as good as (that is, has a fitness value that is equal or better—either smaller (for minimization) or larger (for maximization)—than that of) the current individual.

The original Jaya updates the population-best and the population-worst individuals once every generation. Algorithm 1, however, checks to see if x^{best} needs to be updated, and performs the update if needed, after every single replacement of the existing individual. A similar approach is adopted for updating x^{worst}, but in this case, an update is needed only for the case when the existing (current) individual is the worst one; this is because a replacement is guaranteed never to cause the objective (cost) function to be worse.

Algorithm 1: Pseudocode of the improved algorithm.

```
initialize the population;
find the best and the worst individuals in the population, and initialize bestIndex to the index of
    the best individual and worstIndex to the index of the worst individual;
while a pre-determined stopping condition is not satisfied do
    set the parameters (the r's), independently of one another, to random values between 0.0
        and 1.0;
    for each individual in the population starting from the first index do
        create a new individual using the current individual, the individual at bestIndex,
            the individual at worstIndex, and the random parameters;
        if the new individual is at least as good as the current individual then
            replace the current individual with the new individual;
            if the current individual is better than the individual at bestIndex then
                update bestIndex to set it to the current index;
            end
            if the current individual's index is the same as worstIndex then
                find the worst individual in the population and set worstIndex to the index of the
                    worst individual;
            end
        end
    end
end
```

The simultaneous presence in the population of more than one best (or worst) individual (clones of the same individual and/or different genotypes with the same phenotype) presents no problem for the new algorithm, because the computation of the best (or worst) is always over the entire population, that is, it is never done incrementally.

We improve upon Jaya by changing the policies of updating the best and the worst members and also by changing the criterion used to accept a new member as a replacement of an existing member. The motivation for the first pair of changes comes from the argument that an early availability and use of the best and worst individuals should lead to an earlier creation of better individuals; this is similar to the idea behind the "steady-state" operation of genetic algorithms [16,17]. The logic behind the second change is to try to avoid the "plateau problem".

We call the proposed algorithm semi-steady-state Jaya or SJaya.

4. Simulation Results

For studying the comparative performance of Jaya and SJaya, we use a benchmark test-suite comprising a dozen well-known test functions from the literature and a real-world problem of fuel cell stack design optimization. All of the thirteen problems involve minimization of the objective function value (fitness). The following metrics [18] are used for performance comparison:

- Best-of-run fitness: the best (lowest), mean, and standard deviation of the best-of-run fitness values from 30 (or 100 [Section 4.3]) runs;
- The number of fitness evaluations (FirstHitEvals) needed to reach a specified fitness value for the first time in a run: the best (fewest), mean, and standard deviation of these numbers from 30 (or 100 [Section 4.3]) runs;
- Success count: the number of runs (out of the thirty or the hundred) in which the specified fitness level is reached (it is possible that the specified level is never reached with the given population size and the given number of generations).

The best-of-run fitness provides a measure of the quality of the solution, while the FirstHitEvals metric expresses how fast the algorithm is able to find a solution of a given quality. The two metrics are thus complementary to each other.

4.1. Results on the Benchmark Test-Suite

The benchmark suite (Table 1) [4,19] includes functions of a wide variety of features and levels of problem difficulty, including unimodal/multimodal, separable/non-separable, continuous/discontinuous, differentiable/non-differentiable, and convex/non-convex functions.

For each test function, the population size and the number of generations were chosen based loosely on the problem size (number of variables) and the problem difficulty. No systematic tuning of the population size (PopSize) or the number of generations (Gens) was attempted; the values used in this study were found to be reasonably good across a majority of the problems after a few initial trials. Two PopSize-Gens combinations were used for each function (see Table 2). For d = 30, population sizes of 100 and 150 were used, with the corresponding number of generations being 3000 and 5000. For d = 2, the population sizes were 15 and 20, with 5000 generations used for both. Thirty independent runs of each of the two algorithms were executed for each PopSize-Gens combination on each of the test functions. A run is considered a success if it manages to produce at least one solution with a fitness within a distance of $\pm 1.0 \times 10^{-6}$ from the true (known) global optimum, and the number of fitness evaluations corresponding to the first appearance of such a solution is recorded as the FirstHitEvals of that run.

Tables 2 and 3 show the results of SJaya and Jaya, respectively, on the 12-function test-suite. In all the tables in this paper, results are rounded at the fourth decimal place.

From Tables 2 and 3 we see that SJaya produces superior results than Jaya on all the metrics. Specifically,

- On the best of best-of-runs metric, out of 24 cases, SJaya outperforms Jaya in 12 cases and is outperformed by Jaya in 2 cases, with 10 cases resulting in ties. In a few cases (such as the values of 3.0000 of the best of best-of-run fitnesses and of the mean of best-of-run fitnesses corresponding to the Goldstein-Price function for both SJaya and Jaya), differences exist at the fifth or a later decimal position but do not show in Tables 2 and 3.
- On the mean of best-of-runs metric, SJaya is the winner with win-loss-tie figures of 18-1-5.
- The success counts are higher (5-1-18) for SJaya.
- SJaya outperforms Jaya 19-1-4 on the best FirstHitEvals metric.
- On the mean FirstHitEvals metric, SJaya outperforms Jaya 19-1-4.

Table 1. Benchmark functions.

Name	Definition	Dim.	Global Minimum	Bounds
Ackley	$f(x_1,\cdots,x_n) = -20exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - exp\left(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)\right) + 20 + e$	30	$f(x^*) = 0$ $x^* = (0,\cdots,0)$	$-10 \leq x_i \leq 10$
Rosenbrock	$f(x_1,\cdots,x_n) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$	30	$f(x^*) = 0$ $x^* = (1,\cdots,1)$	$-10 \leq x_i \leq 10$
Chung-Reynolds	$f(x_1,\cdots,x_n) = \left(\sum_{i=1}^{n} x_i^2\right)^2$	30	$f(x^*) = 0$ $x^* = (0,\cdots,0)$	$-10 \leq x_i \leq 10$
Step	$f(x_1,\cdots,x_n) = \sum_{i=1}^{n} \lfloor \lvert x_i \rvert \rfloor$	30	$f(x^*) = 0$ $x_i^* \in (-1,1)$	$-100 \leq x_i \leq 100$
Alpine-1	$f(x_1,\cdots,x_n) = \sum_{i=1}^{n} \lvert x_i \sin(x_i) + 0.1x_i \rvert$	30	$f(x^*) = 0$ $x^* = (0,\cdots,0)$	$-10 \leq x_i \leq 10$
SumSquares	$f(x_1,\cdots,x_n) = \sum_{i=1}^{n} ix_i^2$	30	$f(x^*) = 0$ $x^* = (0,\cdots,0)$	$-10 \leq x_i \leq 10$
Sphere	$f(x_1,\cdots,x_n) = \sum_{i=1}^{n} x_i^2$	30	$f(x^*) = 0$ $x^* = (0,\cdots,0)$	$-100 \leq x_i \leq 100$
Bohachevsky-3	$f(x_1,x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1 + 4\pi x_2) + 0.3$	2	$f(x^*) = 0$ $x^* = (0,0)$	$-100 \leq x_1, x_2 \leq 100$
Bohachevsky-2	$f(x_1,x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1)\cos(4\pi x_2) + 0.3$	2	$f(x^*) = 0$ $x^* = (0,0)$	$-100 \leq x_1, x_2 \leq 100$
Bartels Conn	$f(x_1,x_2) = \lvert x_1^2 + x_2^2 + x_1x_2 \rvert + \lvert \sin(x_1) \rvert + \lvert \cos(x_2) \rvert$	2	$f(x^*) = 1$ $x^* = (0,0)$	$-500 \leq x_1, x_2 \leq 500$
Goldstein-Price	$f(x_1,x_2) = \left[1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\right] \times \left[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\right]$	2	$f(x^*) = 3$ $x^* = (0,-1)$	$-2 \leq x_1, x_2 \leq 2$
Matyas	$f(x_1,x_2) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2$	2	$f(x^*) = 0$ $x^* = (0,0)$	$-10 \leq x_1, x_2 \leq 10$

Table 2. Results of SJaya on the 12-function test-suite (each row corresponds to 30 independent runs). Most numbers are shown with rounding at the fourth place after the decimal.

Function	PopSize	Gens	Best-of-Run Fitness			FirstHitEvals			
			Best	Mean	Std Dev	Success	Best	Mean	Std Dev
Ackley	100	3000	7.4347×10^{-10}	1.8090×10^{-9}	9.1920×10^{-10}	30	209,499	217,209.4333	4885.3830
	150	5000	1.0938×10^{-12}	2.7097×10^{-12}	7.9283×10^{-13}	30	407,146	426,516.7667	7522.8400
Rosenbrock	100	3000	0.0015	25.4532	28.8764	0	—	—	—
	150	5000	0.0001	17.0565	26.9145	0	—	—	—
Chu-Rey	100	3000	5.0261×10^{-37}	1.1798×10^{-35}	3.0313×10^{-35}	30	77,035	84,420.6	3325.8495
	150	5000	1.2691×10^{-48}	4.9288×10^{-47}	6.4529×10^{-47}	30	153,594	162,497.0667	3651.8492
Step	100	3000	0.0	0.0667	0.2494	28	39004	43,895.0357	5319.6538
	150	5000	0.0	0.0	0.0	30	68,099	73,154.9333	3639.5311
Alpine-1	100	3000	0.0247	6.8245	6.4345	0	—	—	—
	150	5000	0.0137	4.5976	5.7499	0	—	—	—
SumSquares	100	3000	6.1724×10^{-18}	3.8440×10^{-17}	4.0234×10^{-17}	30	138,646	144,029.4333	3771.9204
	150	5000	1.3309×10^{-23}	7.2599×10^{-23}	5.5164×10^{-23}	30	266,653	280,539.4333	6344.5950
Sphere	100	3000	5.6616×10^{-17}	2.9297×10^{-16}	2.6115×10^{-16}	30	152,133	157,149.2333	2954.1983
	150	5000	1.3981×10^{-22}	6.1597×10^{-22}	4.1632×10^{-22}	30	298,554	306,880.0667	4927.0814
Boha-3	15	5000	0.0	0.0	0.0	30	882	1322.4667	308.4498
	20	5000	0.0	0.0	0.0	30	1182	1838.7	333.6645
Boha-2	15	5000	0.0	0.0	0.0	30	718	1005.3333	268.2153
	20	5000	0.0	0.0	0.0	30	890	1443.3667	222.3957
Bartels	15	5000	1.0	1.0	0.0	30	893	1061.0	90.1706
	20	5000	1.0	1.0	0.0	30	1128	1523.4333	124.4451
Gold-Pr	15	5000	3.0000	3.0000	1.0820×10^{-5}	6	28,320	55,587.5	14,917.3860
	20	5000	3.0000	3.0000	1.8986×10^{-5}	5	58,442	82,977.0	14,243.2530
Matyas	15	5000	0.0	3.0482×10^{-35}	1.6415×10^{-34}	30	471	856.1	169.1497
	20	5000	0.0	5.6005×10^{-123}	3.0160×10^{-122}	30	692	1152.7333	264.9280

Table 3. Results of Jaya on the 12-function test-suite (each row corresponds to 30 independent runs). Most numbers are shown with rounding at the fourth place after the decimal.

Function	PopSize	Gens	Best-of-Run Fitness			FirstHitEvals			
			Best	**Mean**	**Std Dev**	**Success**	**Best**	**Mean**	**Std Dev**
Ackley	100	3000	4.2232×10^{-6}	7.6506×10^{-6}	1.9595×10^{-6}	0	—	—	—
	150	5000	3.9148×10^{-8}	8.2624×10^{-8}	2.5913×10^{-8}	30	620,422	651,813.4333	11,801.5819
Rosenbrock	100	3000	0.0310	26.8113	27.5200	0	—	—	—
	150	5000	0.0521	37.0939	32.6063	0	—	—	—
Chu-Rey	100	3000	6.1251×10^{-23}	2.2695×10^{-21}	2.7432×10^{-21}	30	122,216	130,083.4667	3283.9261
	150	5000	1.1798×10^{-30}	1.1626×10^{-29}	1.2429×10^{-29}	30	230,733	245,191.6	6139.8179
Step	100	3000	0.0	0.0	0.0	30	82115	88940.6	4467.5422
	150	5000	0.0	0.0	0.0	30	154,374	166,652.7667	6105.3915
Alpine-1	100	3000	0.0240	9.7502	5.6913	0	—	—	—
	150	5000	0.0381	6.2610	5.6690	0	—	—	—
SumSquares	100	3000	1.5297×10^{-10}	4.5700×10^{-10}	2.2292×10^{-10}	30	213,427	222,775.1667	3910.2976
	150	5000	4.9038×10^{-15}	3.7103×10^{-14}	1.7512×10^{-14}	30	406,918	421,195.1	7055.2581
Sphere	100	3000	1.2410×10^{-9}	4.6650×10^{-9}	2.4779×10^{-9}	30	231,137	245,599.1667	4874.0277
	150	5000	8.6939×10^{-14}	3.6152×10^{-13}	2.3875×10^{-13}	30	441,574	464,684.3667	10,701.7923
Boha-3	15	5000	0.0	0.0301	0.1624	29	947	1368.5517	257.8614
	20	5000	0.0	0.0	0.0	30	1461	1877.5333	275.5259
Boha-2	15	5000	0.0	0.0347	0.1866	29	809	1102.7931	158.0520
	20	5000	0.0	0.0	0.0	30	1160	1590.8667	243.5768
Bartels	15	5000	1.0	1.0	0.0	30	995	1238.7667	91.8632
	20	5000	1.0	1.0	0.0	30	1375	1684.0667	152.4998
Gold-Pr	15	5000	3.0000	3.0000	1.4203×10^{-5}	5	36,981	57,683.4	12,921.8507
	20	5000	3.0000	3.0000	1.7344×10^{-5}	3	37,550	52,030.0	15,017.5414
Matyas	15	5000	0.0	1.6173×10^{-11}	8.7092×10^{-11}	30	572	906.9667	261.1821
	20	5000	0.0	1.9566×10^{-55}	1.0537×10^{-54}	30	761	1286.0	264.6156

Table 4 presents the t-scores and one-tailed p-values from Smith–Satterthwaite tests (Welch's tests) [20] (corresponding to unequal population variances) run on the data in Tables 2 and 3 for examining whether or not the difference between the means of Jaya and SJaya (for the best-of-run fitnesses metric and, separately, for the FirstHitEvals metric) is significant. Using the subscripts 1 and 2 for Jaya and SJaya respectively, we obtain the test statistic as a t-score given by

$$t = \frac{\bar{x}_1 - \bar{x}_2 - 0}{\sqrt{\dfrac{s_1^2}{n_1} + \dfrac{s_2^2}{n_2}}},$$

and the degrees of freedom of the t-distribution (this t-distribution is used to approximate the sampling distribution of the difference between the two means) as

$$\frac{\left(\dfrac{s_1^2}{n_1} + \dfrac{s_2^2}{n_2}\right)^2}{\dfrac{(s_1^2/n_1)^2}{n_1 - 1} + \dfrac{(s_2^2/n_2)^2}{n_2 - 1}},$$

where the symbols $\bar{x}$, s and n represent mean, standard deviation and sample size, respectively. Note that even though 30 runs were executed in each case, the sample sizes are not always 30 (because not all runs were successful in all cases); for instance, for the Goldstein-Price function (executed with parameters PopSize = 15 and Gens = 5000), $n_1 = n_2 = 30$ for the mean best-of-run fitness calculation, whereas $n_1 = 5$ and $n_2 = 6$ for the mean FirstHitEvals computation. (To avoid division by zero, we cannot use the above formulas when both s_1 and s_2 are zeros or when any one of n_1 and n_2 is unity.)

Using $\alpha = 0.05$ as the level of significance, we see from the results in Table 4 that on the best-of-run metric, out of a total of 19 cases, ten cases produce a positive t statistic that corresponds to a one-tailed p-value less than α (the p-values were obtained with t-tests from scipy.stats). Thus the null hypothesis $\bar{x}_1 = \bar{x}_2$ must be rejected in favor of $\bar{x}_1 > \bar{x}_2$ for those ten cases. The 19 cases include a lone negative t score, but the corresponding p-value is greater than 0.05. On the FirstHitEvals metric, we have a total of 19 cases (the two occurrences of 19 between best-of-run and FirstHitEvals is a coincidence), of which fourteen have a positive t with a p-value less than 0.05, and a single case has a negative t-score with a less-than-0.05 p-value.

The statistical tests in Table 4 provide performance comparison separately on each of the twelve functions (using two different algorithm parameter settings for each function). A measure of the combined performance on the 12 functions taken together can be obtained using a paired-sample Wilcoxon signed rank test on the 12-function suite. The results of this test for each of the two metrics are presented in Table 5 where the null hypothesis is that the Jaya mean and the SJaya mean are identical and the alternate hypothesis is that the former is larger than the latter. The second column in Table 5 shows the number of zero differences between SJaya and Jaya; n represents the effective number of samples obtained by ignoring the samples, if any, corresponding to zero differences (e.g., n is $24 - 5 = 19$ for the mean of best-of-run fitness metric); W is the test statistic obtained as the minimum of $W+$ and $W-$; α represents the level of significance (a value of 0.05 is used here); and the critical W for a given n and for $\alpha = 0.05$ is obtained from standard statistical tables. The W statistic is seen to be less than the critical W. The mean of W is

$$\text{mean} = \frac{n(n+1)}{4},$$

and its standard deviation is given by

$$\text{std dev} = \sqrt{\frac{n(n+1)(2n+1)}{24}},$$

and, arguing that the sample size n is large enough for the discrete distribution of the W statistic to be approximated by a normal distribution, we obtain the z-statistic as

$$z = \frac{W - \text{mean}}{\text{std dev}}.$$

The one-tailed p-value corresponding to the above z-statistic is obtained from standard tables of the normal distribution.

From the results in Tables 4 and 5 we conclude that at the 5% significance level, SJaya is better than Jaya on the benchmark test-set.

Table 4. Smith–Satterthwaite tests: Jaya vs. SJaya on the benchmark functions.

Function	PopSize	Gens	Best-of-Run Fitness		FirstHitEvals	
			t-Statistic	p-Value	t-Statistic	p-Value
Ackley	100	3000	21.3800	1.3355×10^{-19}	—	—
	150	5000	17.4636	3.1280×10^{-17}	88.1720	3.7508×10^{-56}
Rosenbrock	100	3000	0.1865	0.4264	—	—
	150	5000	2.5958	0.0060	—	—
Chu-Rey	100	3000	4.5314	4.6542×10^{-5}	53.5110	2.3156×10^{-51}
	150	5000	5.1236	8.9954×10^{-6}	63.4031	1.1548×10^{-47}
Step	100	3000	−1.4639	0.0770	34.7952	1.9600×10^{-38}
	150	5000	—	—	72.0480	2.6003×10^{-50}
Alpine-1	100	3000	1.8655	0.0336	—	—
	150	5000	1.1283	0.1319	—	—
SumSquares	100	3000	11.2285	2.2360×10^{-12}	79.3863	4.1571×10^{-61}
	150	5000	11.6045	1.0180×10^{-12}	81.1938	3.3244×10^{-61}
Sphere	100	3000	10.3116	1.6374×10^{-11}	85.0016	4.2333×10^{-54}
	150	5000	8.2938	1.9158×10^{-9}	73.3631	3.1842×10^{-45}
Boha-3	15	5000	1.0171	0.1588	0.6234	0.2678
	20	5000	—	—	0.4915	0.3125
Boha-2	15	5000	1.0171	0.1588	1.7071	0.0472
	20	5000	—	—	2.4494	0.0087
Bartels	15	5000	—	—	7.5641	1.6549×10^{-10}
	20	5000	—	—	4.4699	1.9412×10^{-5}
Gold-Pr	15	5000	1.0676	0.1452	0.2496	0.4042
	20	5000	0.7407	0.2309	−2.8765	0.0217
Matyas	15	5000	1.0171	0.1588	0.8954	0.1875
	20	5000	1.0171	0.1588	1.9494	0.0280

Table 5. Wilcoxon signed rank tests: Jaya vs. SJaya on the 12-function benchmark suite.

Metric	#zero Diff.	n	$W+$	$W-$	W	α	Critical W	Mean of W	Std. Dev. of W	z-Statistic	Left Tail p
Mean of Best-of-Run Fitnesses	5	19	175	15	15	0.05	53	95	24.8495	−3.2194	0.0006
Mean of FirstHitEvals	0	19	180	10	10	0.05	53	95	24.8495	−3.4206	0.0003

4.2. Results on Fuel Cell Stack Design Optimization

A proton exchange membrane fuel cell (PEMFC) [21,22] stack design optimization problem [23–25] is considered here. This problem has been investigated in the fuel cell literature as a problem of practical importance for which the global minimum is believed to be mathematically intractable [24]. This is a constrained optimization problem where the task is to minimize the cost of building a PEMFC stack that meets specific requirements. The objective (cost) function is a function of three variables N_p, N_s, A_{cell}:

$$\text{cost} = K_n \times N_p \times N_s + K_{diff} \times \left| V_{load,rated} - V_{load,mpp} \right| + K_a \times A_{cell} + \mathcal{P},$$

where N_s is the number of cells connected in series in each group; N_p is the number of groups connected in parallel; A_{cell} is the cell area; $V_{load,r}$ is the rated (given) terminal voltage of the stack; $V_{load,mpp}$ represents the output voltage at the maximum power point of the stack; $P_{load,r}$ is the rated (given) output power of the stack; $P_{load,max}$ is the maximum output power of the stack; K_n, K_{diff}, K_a are pre-determined constants [24] used to adjust the relative importance of the different components of the cost function; and $\mathcal{P}$ represents a penalty term given by

$$\mathcal{P} = \begin{cases} 0 & \text{if } P_{load,max} \geq P_{load,r}; \\ c(P_{load,r} - P_{load,max}) & \text{otherwise.} \end{cases}$$

$P_{load,max}$ and $V_{load,mpp}$ are obtained numerically from the following equation by iterating over the load current $i_{load,d}$ (recall that power is voltage times current), using a step size of i_{load} = 1 mA:

$$V_{st} = N_s \left\{ E_{Nernst} - A \ln\left(\frac{i_{load,d}/N_p + i_{n,d}}{i_{0,d}}\right) + B \ln\left(1 - \frac{i_{load,d}/N_p + i_{n,d}}{i_{limit,d}}\right) - (i_{load,d}/N_p + i_{n,d}) r_a \right\},$$

where V_{st} is the stack voltage, E_{Nernst} is the Nernst e.m.f., A and B are constants known from electrochemistry, r_a is the area-specific resistance, and the i's represent different types of current densities (the subscript d is used to indicate density) in the cell [21,26]. The lower and upper bounds of N_s, N_p and A_{cell} are provided in Table 6 and the numerical values of the parameters in Table 7.

Table 6. Bounds of the design variables [23].

Variable	Lower Bound	Upper Bound
N_s	1	50
N_p	1	50
A_{cell} (cm^2)	10	400

Table 7. PEMFC parameters and coefficients.

Parameter	Value
$V_{load,r}$	12 V
$P_{load,r}$	200 W
K_n	0.5
K_{diff}	10
K_a	0.001
c	200
r_a	98.0×10^{-6} KΩ cm^2
$i_{limit,d}$	129 mA/cm^2
$i_{0,d}$	0.21 mA/cm^2
$i_{n,d}$	1.26 mA/cm^2
A	0.05 V
B	0.08 V
E_{Nernst}	1.04 V

Tables 8 and 9 present results of the two algorithms on the fuel cell problem; 30 independent runs are executed for each of 13 PopSize-Gens combinations for either algorithm. For this problem, the success of a run is defined as the production of at least one solution with a fitness of 13.62 or lower [24]. For 12 of the 13 cases in Table 8, the mean of the best-of-run costs is better for SJaya than for Jaya. Furthermore, on the mean FirstHitEvals metric, SJaya outperforms Jaya 10 out of the 13 times. Again, SJaya beats Jaya 9-3-1 on the success count metric. Results of Smith–Satterthwaite tests (Table 10) show that for the best-of-run cost metric, the t-statistic is positive in all cases but one, but the one-tailed p-values are not less than 0.05. Thus we do not have a strong reason at the 5% significance level to reject the null hypothesis that the two means of the best-of-run costs are equal. For the best-of-run metric, the single negative t-score in Table 10 corresponds to a p-value that is close to 0.5, indicating no reason to consider Jaya to be significantly better than SJaya on that case. The FirstHitEvals metric shows SJaya to be significantly better (at the 5% level) in two of the 12 cases, the other cases being ties at that level of significance.

Table 8. Results of SJaya on the fuel cell problem (each row corresponds to 30 independent runs). Most numbers are shown with rounding at the fourth place after the decimal.

PopSize	Gens	Best-of-Run Fitness			FirstHitEvals			
		Best	**Mean**	**Std Dev**	**Success**	**Best**	**Mean**	**Std Dev**
20	10	13.6162	13.6885	0.0759	3	127	172.0	31.9479
15	20	13.6161	13.6255	0.0190	21	128	254.8095	47.8187
20	20	13.6159	13.6376	0.0523	21	127	310.0	71.8338
20	25	13.6159	13.6302	0.0484	25	127	335.8	89.0222
25	40	13.6157	13.6164	0.0023	29	89	510.6897	166.4118
40	25	13.6158	13.6184	0.0044	25	291	654.24	213.3435
20	100	13.6157	13.6158	8.7813×10^{-5}	30	127	436.1333	304.5035
100	20	13.6159	13.6195	0.0029	20	463	1491.5	437.1448
30	100	13.6157	13.6158	0.0002	30	370	585.4667	230.4605
100	30	13.6158	13.6179	0.0022	25	463	1675.08	550.3110
40	100	13.6157	13.6160	0.0006	30	291	778.9333	385.4906
100	40	13.6157	13.6174	0.0022	26	463	1737.1154	622.4179
100	100	13.6157	13.6162	0.0010	29	463	2155.3103	1395.2800

Table 9. Results of Jaya on the fuel cell problem (each row corresponds to 30 independent runs). Most numbers are shown with rounding at the fourth place after the decimal.

PopSize	Gens	Best-of-Run Fitness			FirstHitEvals			
		Best	**Mean**	**Std Dev**	**Success**	**Best**	**Mean**	**Std Dev**
20	10	13.6213	13.7026	0.0713	0	—	—	
15	20	13.6160	13.6374	0.0342	13	124	241.8462	45.9378
20	20	13.6163	13.6367	0.0483	20	298	363.65	31.7636
20	25	13.6160	13.6312	0.0463	25	298	382.36	48.3867
25	40	13.6158	13.6298	0.0520	28	144	540.6071	144.4191
40	25	13.6158	13.6229	0.0236	26	250	739.5	170.0993
20	100	13.6157	13.6182	0.0126	29	298	454.6897	236.2226
100	20	13.6160	13.7947	0.9338	14	907	1595.2857	360.2738
30	100	13.6157	15.1444	8.2308	29	368	740.6207	546.2285
100	30	13.6159	13.7910	0.9344	25	907	1922.44	492.2595
40	100	13.6157	13.6202	0.0237	29	250	787.5517	222.2544
100	40	13.6157	13.7907	0.9345	26	907	1972.7308	544.2687
100	100	13.6157	13.7900	0.9346	27	907	2118.4074	914.8884

Table 10. Smith–Satterthwaite tests: Jaya vs. Sjaya on the fuel cell problem.

PopSize	Gens	Best-of-Run Fitness		FirstHitEvals	
		t-Statistic	p-Value	t-Statistic	p-Value
20	10	0.7429	0.2303	—	—
15	20	1.6673	0.0512	−0.7872	0.2191
20	20	−0.0627	0.4751	3.1175	0.0021
20	25	0.0865	0.4657	2.2976	0.0137
25	40	1.4068	0.0850	0.7256	0.2356
40	25	1.0202	0.1578	1.5742	0.0612
20	100	1.0461	0.1521	0.2620	0.3971
100	20	1.0279	0.1562	0.7564	0.2276
30	100	1.0172	0.1587	1.4129	0.0830
100	30	1.0147	0.1593	1.6751	0.0503
40	100	0.9838	0.1667	0.1056	0.4582
100	40	1.0158	0.1591	1.4530	0.0763
100	100	1.0190	0.1583	−0.1178	0.4534

Table 11 shows results of Wilcoxon signed-rank tests for the PEMFC problem. For each of the two metrics, the W-statistic is less than the critical W. Moreover, the one-tailed p-value computed from the z-score is less than 0.05 for both the metrics, thereby establishing a significant (at the 5% level) superiority of SJaya over Jaya on the fuel cell problem.

4.3. Performance Comparison with the Algorithm of Chakraborty (Energies, 2019)

For a head-to-head comparison of SJaya with the Jaya variant developed in [24], 100 independent runs of SJaya are executed and the results summarized in Table 12. A comparison of the mean of 100 best-of-run costs (Table 12 in this paper and Table 14 in [24]) shows that the present approach's mean value is lower in five of the 13 cases and higher in the remaining eight. The difference, however, is not statistically significant, as seen from the results (Table 13) of the Wilcoxon signed rank test which shows no clear advantage for either algorithm on this metric (the one-tailed p-value is much closer to 0.5 than to zero). On the success count metric (Table 15 in [24]), SJaya outperforms the method of [24] in six cases and is outperformed in four cases, with three cases being ties. On the mean FirstHitEvals metric (Table 15 in [24]), SJaya wins in 11 out of the 13 cases, with the difference seen to be statistically significant at the 5% level (the p-value in Table 13 is 0.0044). Thus we conclude that SJaya is quite competitive with the method of [24].

Table 11. Wilcoxon signed rank tests: Jaya vs. Sjaya on the fuel cell problem.

Metric	#zero Diff.	n	$W+$	$W-$	W	α	Critical W	Mean of W	Std. Dev. of W	z-Statistic	Left Tail p
Mean of Best-of-Run Fitnesses	0	13	90	1	1	0.05	21	45.5	14.3091	−3.1099	0.0009
Mean of FirstHitEvals	0	12	71	7	7	0.05	17	39	12.7475	−2.5103	0.0060

Table 12. Results of SJaya on the fuel cell problem (each row corresponds to 100 independent runs). Most numbers are shown with rounding at the fourth place after the decimal.

PopSize	Gens	Best-of-Run Fitness			FirstHitEvals			
		Best	Mean	Std Dev	Success	Best	Mean	Std Dev
20	10	13.6162	13.6847	0.0677	6	127	184.0	29.2461
15	20	13.6159	14.6855	10.5138	64	116	249.6094	48.8686
20	20	13.6159	13.6276	0.0336	69	127	322.4203	62.6424
20	25	13.6158	13.6225	0.0278	86	127	348.9767	78.3951
25	40	13.6157	13.6176	0.0136	97	89	486.6598	149.3530
40	25	13.6157	13.6203	0.0143	81	267	693.6543	189.3033
20	100	13.6157	13.6159	0.0005	100	127	422.63	236.1126
100	20	13.6159	13.6203	0.0049	61	365	1422.6230	434.6534
30	100	13.6157	13.6159	0.0006	99	175	581.5859	200.6175
100	30	13.6157	13.6177	0.0022	86	365	1723.3721	616.9152
40	100	13.6157	13.6159	0.0004	100	267	829.13	383.5640
100	40	13.6157	13.6170	0.0019	92	365	1839.6522	743.6137
100	100	13.6157	13.6162	0.0010	99	365	2136.2727	1335.2097

Table 13. Wilcoxon signed rank tests: Algorithm of [24] vs. Sjaya on the fuel cell problem.

Metric	#zero Diff.	n	$W+$	$W-$	W	α	Critical W	Mean of W	Std. Dev. of W	z-Statistic	Left Tail p
Mean of Best-of-Run Fitnesses	1	12	42	36	36	0.05	17	39	12.7476	−0.2353	0.4070
Mean of FirstHitEvals	0	13	83	8	8	0.05	21	45.5	14.3091	−2.6207	0.0044

4.4. Performance Comparison with Other Heuristics

Because the performance of EC heuristics depends—often dramatically—on parameter settings, empirical performance comparison of SJaya with other heuristics may not mean much unless either those competing heuristics require no other parameters except the population size and the number of generations, or the comparative study is based on runs with a very large number of parameter setting combinations. Most stochastic heuristics in the EC family, however, employ additional parameters (probability of crossover [27], probability of mutation [28], strategy parameters [1,29], to name a few). A proper head-to-head comparison of SJaya with non-Jaya methods is therefore difficult. Table 14 presents the results of a brief comparative study of SJaya with four well-known EC algorithms, namely genetic algorithm (GA) [1], particle swarm optimization (PSO) [30], differential evolution (DE) [29] and artificial bee colony algorithm (ABC) [31]. The metrics used for comparison are the mean and the standard deviation of the best-of-run solutions of 30 independent runs for each problem, with each run executed for 500,000 evaluations (population size = 50 and number of generations = 10,000). (The population size and the number of generations used to produce the data in Table 14 are different from those used earlier in this paper; and, for Ackley and Rosenbrock, the bounds are not the same as the corresponding ones in Table 1.) The non-SJaya results in this table are taken from [31]. These results show that SJaya is competitive with the other methods.

Table 14. Comparative performance: mean and standard deviation of 30 best-of-run fitnesses. For each function, the top row shows the means and the bottom row the standard deviations (results are rounded at the fourth decimal place).

Function	Dim.	Bounds	Glob. min.	GA	PSO	DE	ABC	SJaya
Ackley	30	$(-32, 32)$	0	14.6718	0.1646	0	0	0.2723
				0.178141	0.493867	0	0	0.5022
Rosenbrock	30	$(-30, 30)$	0	1.96×10^5	15.0886	18.2039	0.0888	0.0009
				3.85×10^4	24.1702	5.0362	0.0774	0.0046
Step	30	$(-100, 100)$	0	1.17×10^3	0	0	0	0.2333
				76.5615	0	0	0	0.4955
SumSquares	30	$(-10, 10)$	0	1.48×10^2	0	0	0	0
				12.4093	0	0	0	0
Sphere	30	$(-100, 100)$	0	1.11×10^3	0	0	0	0
				74.2145	0	0	0	0
Boha-3	2	$(-100, 100)$	0	0	0	0	0	0
				0	0	0	0	0
Boha-2	2	$(-100, 100)$	0	0.0683	0	0	0	0
				0.0782	0	0	0	0
Goldstein-Price	2	$(-2, 2)$	3	5.2506	3	3	3	3
				5.8701	0	0	0	0
Matyas	2	$(-10, 10)$	0	0	0	0	0	0
				0	0	0	0	0

5. Conclusions

This paper presented an improvement to the Jaya algorithm by introducing new update policies in the search process. The usefulness of the present approach is that, unlike most other improvements to Jaya reported in the literature, our strategy does not require the introduction of any additional parameter. It retains both the features that the original Jaya is famous for, namely "parameterlessness" and simplicity, while providing performance that is statistically significantly better (in terms of the solution quality) and/or faster (in terms of the speed of finding a near-optimal solution) than that produced by Jaya.

Funding: This research was funded in part by United States National Science Foundation Grant IIS-1115352.

Acknowledgments: The author thanks the anonymous reviewers for their comments.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Michalewicz, Z. *Genetic Algorithms+ Data Structures = Evolution Programs*; Springer Science & Business Media: Berlin, Germany, 2013.
2. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
3. Wolpert, D.H.; Macready, W.G. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.* **1997**, *1*, 67–82. [CrossRef]
4. Rao, R. Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *Int. J. Ind. Eng. Comput.* **2016**, *7*, 19–34.
5. Rao, R.V.; Keesari, H.S. Multi-team perturbation guiding Jaya algorithm for optimization of wind farm layout. *Appl. Soft Comput.* **2018**, *71*, 800–815. [CrossRef]
6. Li, Y.; Yang, Z. Application of EOS-ELM with binary Jaya-based feature selection to real-time transient stability assessment using PMU data. *IEEE Access* **2017**, *5*, 23092–23101. [CrossRef]
7. Pampara, G.; Franken, N.; Engelbrecht, A.P. Combining particle swarm optimisation with angle modulation to solve binary problems. In Proceedings of the 2005 IEEE Congress on Evolutionary Computation, Scotland, UK, 2–5 September 2005; Volume 1, pp. 89–96.
8. Rao, R.V.; Saroj, A. A self-adaptive multi-population based Jaya algorithm for engineering optimization. *Swarm Evol. Comput.* **2017**, *37*, 1–26.
9. Rao, R.V.; Rai, D.P.; Balic, J. Multi-objective optimization of abrasive waterjet machining process using Jaya algorithm and PROMETHEE Method. *J. Intell. Manuf.* **2019**, *30*, 2101–2127. [CrossRef]
10. Michailidis, P.D. An efficient multi-core implementation of the Jaya optimisation algorithm. *Int. J. Parallel Emergent Distrib. Syst.* **2019**, *34*, 288–320. [CrossRef]
11. Nayak, D.R.; Dash, R.; Majhi, B. Development of pathological brain detection system using Jaya optimized improved extreme learning machine and orthogonal ripplet-II transform. *Multimed. Tools Appl.* **2018**, *77*, 22705–22733. [CrossRef]
12. Buddala, R.; Mahapatra, S.S. Improved teaching–learning-based and JAYA optimization algorithms for solving flexible flow shop scheduling problems. *J. Ind. Eng. Int.* **2018**, *14*, 555–570. [CrossRef]
13. Huang, C.; Wang, L.; Yeung, R.S.C.; Zhang, Z.; Chung, H.S.H.; Bensoussan, A. A prediction model-guided Jaya algorithm for the PV system maximum power point tracking. *IEEE Trans. Sustain. Energy* **2017**, *9*, 45–55. [CrossRef]
14. Sinha, R.K.; Ghosh, S. Jaya based ANFIS for monitoring of two class motor imagery task. *IEEE Access* **2016**, *4*, 9273–9282.
15. Gao, K.; Zhang, Y.; Sadollah, A.; Su, R. Jaya algorithm for solving urban traffic signal control problem. In Proceedings of the 2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV), Phuket, Thailand, 13–15 November 2016; pp. 1–6.
16. Syswerda, G. A study of reproduction in generational and steady-state genetic algorithms. In *Foundations of Genetic Algorithms*; Elsevier: Amsterdam, The Netherlands, 1991; Volume 1, pp. 94–101.
17. Chakraborty, U.K.; Deb, K.; Chakraborty, M. Analysis of selection algorithms: A Markov chain approach. *Evol. Comput.* **1996**, *4*, 133–167. [CrossRef]
18. Chakraborty, U.K.; Abbott, T.E.; Das, S.K. PEM fuel cell modeling using differential evolution. *Energy* **2012**, *40*, 387–399. [CrossRef]
19. Jamil, M.; Yang, X.S. A literature survey of benchmark functions for global optimisation problems. *Int. J. Math. Model. Numer. Optim.* **2013**, *4*, 150–194. [CrossRef]
20. Johnson, R.A.; Miller, I.; Freund, J.E. *Probability and Statistics for Engineers*; Pearson Education: London, UK, 2000; Volume 2000,
21. Larminie, J.; Dicks, A.; McDonald, M.S. *Fuel Cell Systems Explained*; J. Wiley: Chichester, UK, 2003; Volume 2.
22. O'hayre, R.; Cha, S.W.; Colella, W.; Prinz, F.B. *Fuel Cell Fundamentals*; John Wiley & Sons: Chichester, UK, 2016.

23. Mohamed, I.; Jenkins, N. Proton exchange membrane (PEM) fuel cell stack configuration using genetic algorithms. *J. Power Sources* **2004**, *131*, 142–146. [CrossRef]
24. Chakraborty, U.K. Proton exchange membrane fuel cell stack design optimization using an improved Jaya algorithm. *Energies* **2019**, *12*, 3176. [CrossRef]
25. Besseris, G.J. Using qualimetric engineering and extremal analysis to optimize a proton exchange membrane fuel cell stack. *Appl. Energy* **2014**, *128*, 15–26. [CrossRef]
26. Chakraborty, U.K. A new model for constant fuel utilization and constant fuel flow in fuel cells. *Appl. Sci.* **2019**, *9*, 1066. [CrossRef]
27. Yi, J.H.; Xing, L.N.; Wang, G.G.; Dong, J.; Vasilakos, A.V.; Alavi, A.H.; Wang, L. Behavior of crossover operators in NSGA-III for large-scale optimization problems. *Inf. Sci.* **2020**, *509*, 470–487. [CrossRef]
28. Michalewicz, Z.; Fogel, D.B. *How to Solve It: Modern Heuristics*; Springer Science & Business Media: Berlin, Germany, 2013.
29. Chakraborty, U.K. *Advances in Differential Evolution*; Springer: Berlin, Germany, 2008; Volume 143.
30. Bonyadi, M.R.; Michalewicz, Z. Particle swarm optimization for single objective continuous space problems: A review. *Evol. Comput.* **2017**, *25*, 1–54. [CrossRef]
31. Karaboga, D.; Akay, B. A comparative study of artificial bee colony algorithm. *Appl. Math. Comput.* **2009**, *214*, 108–132. [CrossRef]