Soft Computing (2021) 25:15163–15181
https://doi.org/10.1007/s00500-021-06445-2

OPTIMIZATION

Self-adaptively commensal learning-based Jaya algorithm with multi-populations and its application

Zuanjia Xie[1] · Chunliang Zhang[1,2] · Haibin Ouyang[1,2] · Steven Li[3] · Liqun Gao[4]

Accepted: 15 October 2021 / Published online: 25 October 2021

Abstract
Jaya algorithm is an advanced optimization algorithm, which has been applied to many real-world optimization problems. Jaya algorithm has better performance in some optimization field. However, Jaya algorithm exploration capability is not better. In order to enhance exploration capability of the Jaya algorithm, a self-adaptively commensal learning-based Jaya algorithm with multi-populations (Jaya-SCLMP) is presented in this paper. In Jaya-SCLMP, a commensal learning strategy is used to increase the probability of finding the global optimum, in which the person history best and worst information is used to explore new solution area. Moreover, a multi-populations strategy based on Gaussian distribution scheme and learning dictionary is utilized to enhance the exploration capability, meanwhile every subpopulation employed three Gaussian distributions at each generation, roulette wheel selection is employed to choose a scheme based on learning dictionary. The performance of Jaya-SCLMP is evaluated based on 28 CEC 2013 unconstrained benchmark problems. In addition, three reliability problems, i.e., complex (bridge) system, series system and series–parallel system, are selected. Compared with several Jaya variants and several state-of-the-art other algorithms, the experimental results reveal that Jaya-SCLMP is effective.

Keywords Jaya algorithm · Multi-populations strategy · Learning strategy · Reliability problem

1 Introduction

To solve complex optimization problems with a limited time in engineering optimization area is a challenge and hot research topic. How to design some simple and effective methods to adapt and overcome more and more real-world engineering optimization problems are value to study and discussing. Many optimization problems come from real life and industrial production, and they through principle logic, mathematical thinking and planning modeling method evolved into mathematical optimization problems. Although the conventional methods can often find a solution, it has become more and more tedious and time consuming. Moreover, the conventional methods cannot guarantee finding the optimal solution effectively in a very short time. Therefore, many advanced metaheuristic optimization algorithms are being developed. These new optimization algorithms are capable of achieving the global or near global optimum solution with less information about the problems. Compared to the conventional method, the metaheuristic optimization algorithms have some advantages and they play an important role in science and engineering field.

During the past several decades, many well-known metaheuristic optimization algorithms have been developed to solve optimization problems, such as genetic algorithm (GA) (Deb et al. 2002), harmony search(HS) (Geem et al. 2001; Papa et al. 2016), particles swarm optimization (PSO) (Eberhart and Kennedy 1995), artificial

Zuanjia Xie and Chunliang Zhang are the common first author.

✉ Haibin Ouyang
oyhb1987@163.com

1 School of Mechanical and Electric Engineering, Guangzhou University, Guangzhou 510006, China

2 Guangzhou Key Laboratory of Condition Monitoring and Control of Mechanical and Electrical Equipment, Guangzhou, China

3 Graduate School of Business and Law, RMIT University, Melbourne 3000, Australia

4 College of Information Science and Engineering, Northeastern University, Shenyang 110004, China

bee colony (ABC) (Akay and Karaboga 2012), differential evolution (DE) (Storn and Price 1997), gravitational search algorithm (GSA) (Rashedi et al. 2009) and teaching–learning-based optimization (TLBO) (Rao et al. 2011). These algorithms attracted much attention and aroused many scholars interesting. Jaya algorithm is a relatively new algorithm (Rao and Waghmare 2016) based on the principle to move the solution closer to the best solution and further away from the worst solution at the same time. This principle ensures that Jaya algorithm has good exploitation ability. However, its exploration capability is not better. Therefore, it is important to find a strategy to enhance the exploration capability of Jaya algorithm. To improve the performance of the Jaya algorithm, researchers have proposed various Jaya variants in the past decades (Azizi et al. 2019; Degertekin et al. 2019; Yu et al. 2017; Zhang et al. 2018; Wang et al. 2018a). Warid et al. (2018) proposed a modified Jaya algorithm based on novel quasi-oppositional strategy, and it is applied to the multi-objective optimal power flow problem. Yu et al. (2019) employed a performance-guided Jaya algorithm to solve the parameter identification problem of photovoltaic cell and module. Rao (2019) summarized the application of Jaya algorithm and its variants on constrained and unconstrained benchmark functions. Ocłoń et al. (2018) presented a modified Jaya algorithm to solve the thermal performance optimization of the underground power cable system. Rao and Saroj (2018a) designed an elitism-based self-adaptive multi-population Jaya algorithm, which was used to solve some engineering optimization problems. Jaya algorithm faces a few problems like other meta-heuristic optimization algorithms such as TLBO algorithm and HS algorithm. For example, it easily gets stuck in local space for some optimization problems and its exploitation and exploration capability need to be balanced and adjusted. Based on this observation, our aim is to improve the performance of Jaya algorithm and to make it more applicable.

To improve the performance of Jaya algorithm, some main contributions are summarized as follows:

(1) A commensal learning strategy is used to increase the probability of finding the global optimum, in which the person history best and worst information is used to explore new solution area.

(2) A multi-populations strategy based on Gaussian distribution scheme and learning dictionary is utilized to enhance the exploration capability, meanwhile every subpopulation employed three Gaussian distributions at each generation, roulette wheel selection is employed to choose a scheme based on learning dictionary.

(3) The performance of Jaya-SCLMP is evaluated based on 28 CEC 2013 unconstrained benchmark problems and reliability problems, i.e., complex (bridge) system, series system and series–parallel system. Compared with several Jaya variants and several state-of-the-art other algorithms, the experimental results reveal that Jaya-SCLMP can obtain some competitive results.

The remainder of this paper is organized as follows. The related work on the Jaya algorithm is reviewed in Sect. 2. Section 3 describes the original Jaya algorithm. The proposed Jaya-SCLMP is proposed in Sect. 4. In Sect. 5, Jaya-SCLMP is compared with several EAs based on 28 CEC 2013 unconstrained benchmark problems and three example reliability problems. The experimental results and discussions are also reported. Finally, Sect. 6 draws the conclusions.

2 Related work

To improve the performance of Jaya algorithm, researchers have proposed many Jaya algorithm variants in recent years. The improvements in Jaya algorithm have been active and rapid with many successful applications to various real-world optimization problems.

To improve the Jaya algorithm, researchers are focusing on parameter adjustment, operator design, hybrid algorithm, etc. To increase the probability of finding the global optimum, Ocłon et al. (2018) proposed a modified Jaya algorithm (MJaya) with a novel candidate update scheme. Farah and Belazi (2018) proposed a novel chaotic Jaya algorithm for unconstrained numerical optimization, in which chaotic theory and strategy are integrated into search operation. Rao et al. (2017) introduced a quasi-oppositional based Jaya algorithm (QO-Jaya). In QO-Jaya, a quasi-opposite population is generated at each generation to achieve a better performance (Rao and Rai 2017). Rao et al. (2017) presented a self-adaptive multi-population-based Jaya algorithm for engineering optimization problems, called SAMP-Jaya. SAMP-Jaya divides the population into a number of groups based on the quality of the solution (Rao and Saroj 2017a). One year later, Rao and Saroj (2018b) incorporated an elitism strategy into SAMP-Jaya to improve the performance of SAMP-Jaya. Rao and More (2017) proposed a self-adaptive Jaya algorithm to optimize and analyze the selected thermal devices. Some improved Jaya algorithms are summarized in Table 1.

Application study is another research aspect of Jaya algorithm. Rao et al. (2016) used the Jaya algorithm to solve micro-channel heat sink dimensional optimization, compared to other related algorithm, Jaya algorithm has

Table 1 Several Jaya variants and their improvements

Jaya variants	Improvements
MJaya (Ocłoń et al. 2018)	Multi-elitism strategy
CJaya (Farah and Belazi 2018)	Chaotic search strategy
QO-Jaya (Rao and Rai 2017)	Quasi-opposite population
SAMP-Jaya (Rao and Saroj 2017a)	Self-adaptive multi-population strategy
S-Jaya (Rao and More 2017)	Self-adaptive population sizes
EO-Jaya (Wang and Huang 2018)	Novel Elite opposition-based
SAMPE-Jaya (Rao and Saroj 2018a)	Elitism-based self-adaptive multi-population
S-Jaya (Huang et al. 2018)	Natural cubic-spline-based prediction model
DJaya (Gao et al. 2019)	Five objective-oriented local search operators and four ensembles of them

some merits in optimization performance. Moreover, (2017b) further considered the constrained economic optimization of shell-and-tube heat exchangers and provided a modified Jaya algorithm based on differential strategies such as elitist mechanism, and the simulation shows that the modified Jaya algorithms perform better. Wang et al. (2018b) combined wavelet Renyi entropy with three-segment encoded Jaya algorithm to solve alcoholism identification. Azizi et al. (2019) used hybrid ant lion optimizer and Jaya algorithm to solve fuzzy controller optimum design. In 2018, Rao et al. employed elitist-Jaya algorithm to solve heat exchangers multi-objective optimization problem (Rao and Saroj 2018c) and proposed a multi-team perturbation guiding Jaya algorithm for wind farm layout optimization problem (Rao and Keesari 2018). Grzywinski employed Jaya algorithm with frequency constraints for the optimization of the braced dome structures (Grzywinski et al. 2019). Degertekin proposed a Jaya algorithm to solve sizing, layout and topology design optimization of truss structures (Degertekin et al. 2018). Huang proposed a novel model-free solution algorithm, the natural cubic-spline-guided Jaya algorithm (S-Jaya), for efficiently solving the maximum power point tracking (MPPT) problem of PV systems under partial shading conditions (Huang et al. 2018). Wang designed a novel elite opposition-based Jaya algorithm for parameter estimation of photovoltaic cell models (Wang and Huang 2018). In 2019, Gao et al. (2019) proposed a discrete Jaya algorithm for solving a flexible job-shop rescheduling problem (FJRP), in which five objective-oriented local search operators and four ensembles of them are proposed to improve the performance of DJaya algorithm.

In sum, Jaya algorithm has many advantages. However, like other algorithms, it suffers from some weaknesses while it is used to solve real-world complex and large-scale optimization problems. It is valuable and important to enhance the exploration capability of Jaya algorithm. Our paper focuses on the improvement of the Jaya algorithm and some applications with some new ideas.

3 Jaya algorithm

Jaya algorithm is a relatively new algorithm. The core of Jaya algorithm lies in the principle of moving the solution closer to the best solution and further away from the worst solution at the same time. The details of Jaya algorithm can be described as below.

Let *f(x)* be the objective function to be optimized. Assume that at any iteration *i*, there are D design variables and NP candidate solutions (i.e., population size, *i* = 1, 2, …, NP). If $x^t_{i,j}$ is the value of the *j*th variable for the *i*th candidate during the *t*th iteration, then this value is modified as follows:

$$X^{t+1}_{i,j} = X^{t}_{i,j} + r_{1,i,j} \times \left(X^{t}_{best,j} - \left| X^{t}_{i,j} \right| \right) - r_{2,i,j} \times \left(X^{t}_{worst,j} - \left| X^{t}_{i,j} \right| \right) \quad (1)$$

where $X^t_{best,j}$ is the value of the variable j for the best candidate and $X^t_{worst,j}$ is the value of the variable j for the worst candidate in the population. $X^{t+1}_{i,j}$ is the updated value of $X^t_{i,j}$ and $r_{1,i,j}$ and $r_{2,i,j}$ are the two random numbers for the *j*th variable during the *t*th iteration in the range [0,1]. The term $r_{1,i,j} \times (X^t_{best,j} - |X^t_{i,j}|)$ indicates the tendency of the solution to move closer to the best solution, and the term $r_{2,i,j} \times (X^t_{worst,j} - |X^t_{i,j}|)$ indicates the tendency of the solution to avoid the worst solution (Rao and Waghmare 2016). $X^{t+1}_{i,j}$ is accepted if it gives a better fitness value. Figure 1 shows the flowchart of the Jaya algorithm.

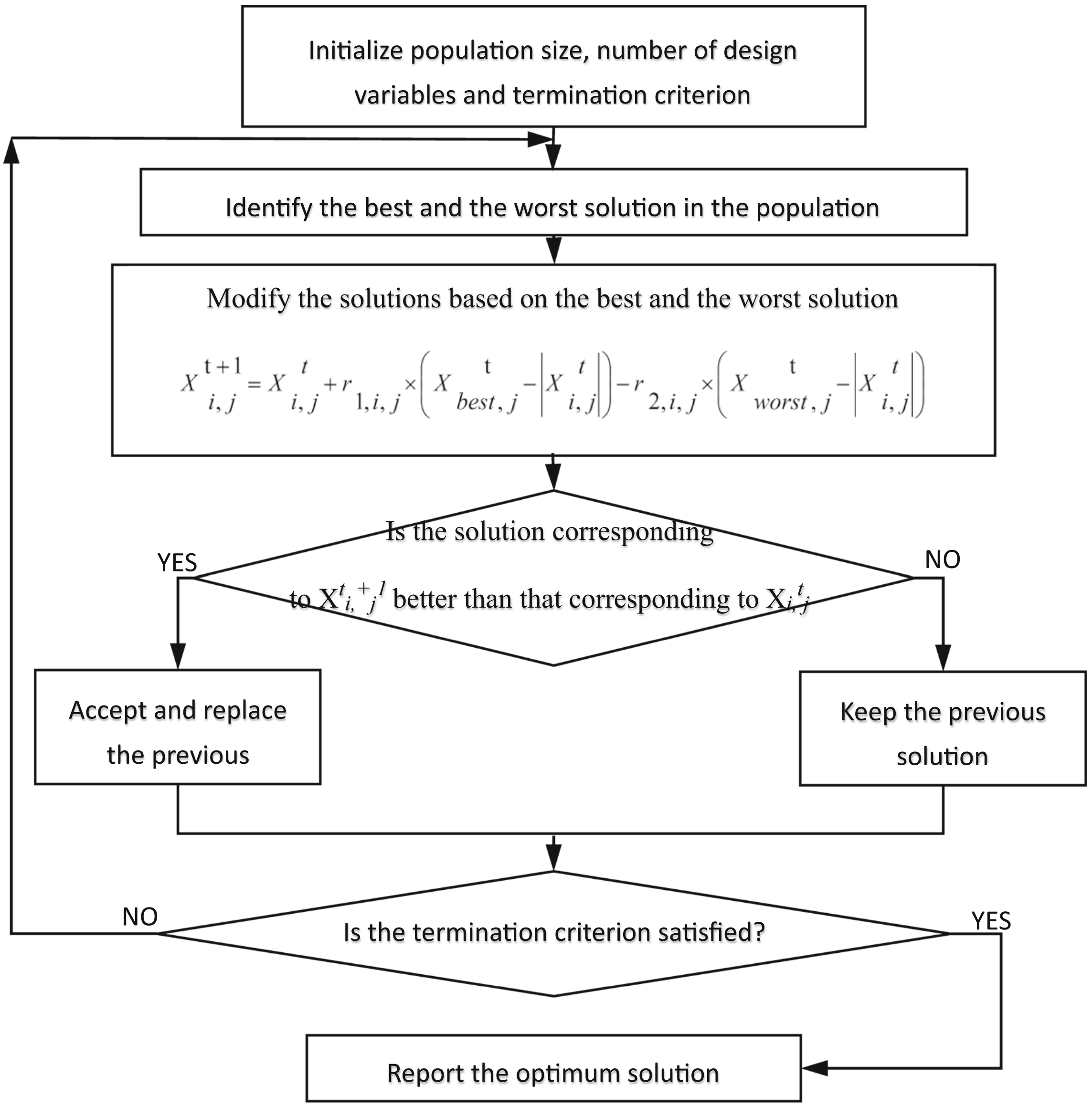
Initialize population size, number of design variables and termination criterion
Identify the best and the worst solution in the population
Modify the solutions based on the best and the worst solution
$X^{t+1}_{i,j} = X^{t}_{i,j} + r_{1,i,j} \times \left(X^{t}_{best,j} - \left| X^{t}_{i,j} \right| \right) - r_{2,i,j} \times \left(X^{t}_{worst,j} - \left| X^{t}_{i,j} \right| \right)$
Is the solution corresponding to $X^{t+1}_{i,j}$ better than that corresponding to $X^{t}_{i,j}$
YES
NO
Accept and replace the previous
Keep the previous solution
Is the termination criterion satisfied?
NO
YES
Report the optimum solution

Fig. 1 The flowchart of the Jaya algorithm

4 Jaya-SCLMP algorithm

In this section, we propose a self-adaptively commensal learning-based multi-population Jaya algorithm, namely Jaya-SCLMP. In Jaya-SCLMP, we modify the candidate update scheme of Jaya algorithm. Moreover, the commensal learning strategy and multi-population strategy are incorporated into Jaya-SCLMP to increase the probability of finding the global optimum.

4.1 Commensal learning strategy

In 2017, Peng et al. (2017) proposed the conception of commensal learning, and the primary idea is the mutation strategies and parameter settings adaptively adjust together under the same criteria and multiple combinations of the two parts commensal evolution for each individual. In Peng et al. (2017), the results show that the commensal learning can enhance the performance of differential evolution algorithm. We analyze the characteristic of Jaya algorithm and consider to use the commensal learning to amend the performance of Jaya algorithm. Jaya algorithm was proposed as an algorithm with less algorithm-specific parameters, but it can easily be trapped in local optima. To the best of our knowledge, using a number of subpopulations distributes the solutions over the search space rather than concentrating on a particular region. This multi-population strategy can enhance the population diversity of the algorithm. Therefore, the candidate solutions can escape from the local optima. How to use the commensal learning to improve the performance of Jaya algorithm is a challenge? We realized the random uniform distribution random number maybe affect the optimization process of Jaya algorithm. To adjust the random number under the same condition for balancing the search space, so the idea of commensal learning is integrated. In Jaya algorithm, new candidate solutions are generated through Eq. (1), but in Jaya-SCLMP, new candidates are produced by following Eq. (2):

$$X_{i,\mathrm{j}}^{\mathrm{t}+1} = X_{i,j}^{t} + N\left(\mu,\sigma^2\right) \times \left(X_{best(\mathrm{p}),j}^{\mathrm{t}} - \left|X_{i,j}^{t}\right|\right) - N\left(\mu,\sigma^2\right) \times \left(X_{worst(p),j}^{\mathrm{t}} - \left|X_{i,j}^{t}\right|\right) \quad (2)$$

where $X^{t}_{best(p),j}$ is the value of the jth variable for the person best candidate and $X^{t}_{worst(p),j}$ is the value of the jth variable for the person worst candidate in pth subpopulation at tth generation. $N(\mu,\sigma^2)$ is a random real number Gaussian distribution $\mu = 0.3, 06, 0.9$; $\sigma^2 = 0.025$. We use Gaussian distribution with different average value take the place of uniform distribution which is used in Jaya algorithm. The idea is to maximize the likelihood of generating new solutions along appropriate directions and accelerate the convergence speed. To enhance the population diversity of the algorithm, the commensal learning is proposed in this paper. The primary idea is to balance the search space based on the best solution and the worst solution under the same criteria. Further, it uses multiple combinations of the two parts of the commensal evolution for each individual.

4.2 Multi-population strategy

In order to improve the diversity of the Jaya algorithm, we consider to design multi-population strategy based on different Gaussian distribution. At first, combine with the commensal learning to analyze the setting of Gaussian distribution parameter values and the number of group. The number of groups is set 2, 3, 4, 5, 6, the value of μ set in (0, 1), the value of σ^2 set in stochastic value, a great many of simulation test shown that if we use four groups to this algorithm, it has better results. Meanwhile, we find the value of μ and σ^2 have effect for the performance, so we fixed four groups and discuss the value of μ and σ^2. We all know if each groups contain different schemes, maybe the diversity is better, but too many will reduce the search accuracy, so we try to use three schemes, in fact, too many times tests also show the three schemes are effect. Therefore, we should use 12 schemes because each group has three schemes. Although the simulation results may not show the only one conditions, we obtain a relative better situation. Due to space limitation, the data and charts of specific parameter simulation will not be added. We elaborate how to choose the best solution and the worst solution of four subpopulations combined with three Gaussian distribution parameter settings to form twelve subpopulation update schemes. The twelve subpopulation update schemes are shown as follows:

Scheme 1: $X^{t}_{\mathrm{best}(1)}$, $X^{t}_{\mathrm{worst}(1)}$, $N(0.3, 0.025)$.
Scheme 2: $X^{t}_{\mathrm{best}(1)}$, $X^{t}_{\mathrm{worst}(1)}$, $N(0.6, 0.025)$.
Scheme 3: $X^{t}_{\mathrm{best}(1)}$, $X^{t}_{\mathrm{worst}(1)}$, $N(0.9, 0.025)$.
Scheme 4: $X^{t}_{\mathrm{best}(2)}$, $X^{t}_{\mathrm{worst}(2)}$, $N(0.3, 0.025)$.
Scheme 5: $X^{t}_{\mathrm{best}(2)}$, $X^{t}_{\mathrm{worst}(2)}$, $N(0.6, 0.025)$.
Scheme 6: $X^{t}_{\mathrm{best}(2)}$, $X^{t}_{\mathrm{worst}(2)}$, $N(0.9, 0.025)$.
Scheme 7: $X^{t}_{\mathrm{best}(3)}$, $X^{t}_{\mathrm{worst}(3)}$, $N(0.3, 0.025)$.
Scheme 8: $X^{t}_{\mathrm{best}(3)}$, $X^{t}_{\mathrm{worst}(3)}$, $N(0.6, 0.025)$.
Scheme 9: $X^{t}_{\mathrm{best}(3)}$, $X^{t}_{\mathrm{worst}(3)}$, $N(0.9, 0.025)$.
Scheme 10: $X^{t}_{\mathrm{best}(4)}$, $X^{t}_{\mathrm{worst}(4)}$, $N(0.3, 0.025)$.
Scheme 11: $X^{t}_{\mathrm{best}(4)}$, $X^{t}_{\mathrm{worst}(4)}$, $N(0.6, 0.025)$.
Scheme 12: $X^{t}_{\mathrm{best}(4)}$, $X^{t}_{\mathrm{worst}(4)}$, $N(0.9, 0.025)$.

For every subpopulation at each generation, roulette wheel selection is employed to choose a scheme based on learning dictionary to evolve. Learning dictionary is an assessment table of evolution effectiveness. After selection operation in pth subpopulation, the times of successful update ($st_{p,s}$) and the times of tried update ($\mathrm{tt}_{\mathrm{p,s}}$) on the sth scheme are recorded. As shown in Table 2, the row and column of the learning dictionary represent four subpopulations and the twelve update schemes, respectively. In the learning dictionary, the cell(s,p) records the twelve update schemes and the success rate ($sr_{p,s}$) for pth subpopulation on sth scheme, and $sr_{p,s}$ is obtained by dividing $st_{p,s}$ by $tt_{p,s}$. Subpopulations will select an update scheme according to the success rate ($sr_{p,s}$) to update its individuals.

4.3 Framework of Jaya-SCLMP

Step 1: Initialization four subpopulations.

1.1 Randomly initialize the individuals of four subpopulations within the upper and lower limits;

1.2 Evaluate fitness of each individual of four subpopulations;

Step 2: Evolutionary phase.

Table 2 Illustration of the learning dictionary

Scheme population	Scheme 1	Scheme 2	……	Scheme 12
Subpopulation1	$sr_{1,1}$	$sr_{1,2}$	……	$sr_{1,12}$
Subpopulation2	$sr_{2,1}$	$sr_{2,2}$	……	$sr_{2,12}$
Subpopulation3	$sr_{3,1}$	$sr_{3,2}$	……	$sr_{3,12}$
Subpopulation4	$sr_{4,1}$	$sr_{4,2}$	……	$sr_{4,12}$

2.1 Find the best and worst individual of each subpopulation.

2.2 For each subpopulation, select a scheme according to learning dictionary for each subpopulation to update its individuals;

2.3 If $X^{t+1}_{i,j}$ is better than $X^{t}_{i,j}$, accept $X^{t+1}_{i,j}$.

Step 3: If the termination criteria is satisfied, stop; otherwise go to Step 2.

Like the traditional Jaya algorithm, Jaya-SCLMP also consists of a very simple framework. At the first, Jaya-SCLMP initializes four subpopulations. In the evolutionary process, Jaya-SCLMP first finds the best solutions and the worst solutions of the subpopulations. Then, for each subpopulation, an update scheme according to the success rate is chosen and then its individuals are updated. The steps of the Jaya-SCLMP algorithm are described in Fig. 2.

4.4 Computational complexity

For simplicity we compute the running time of an algorithm purely as a function of the length of the string representing the input. In the worst-case analysis, the form we consider the longest running time of all inputs of a particular length. The time complexity of an algorithm is commonly expressed using the big O notation, which excludes coefficients and lower-order terms. In this subsection, the computational complexity of the proposed algorithm was briefly analyzed based on the computation procedure of the proposed algorithm. Based on the

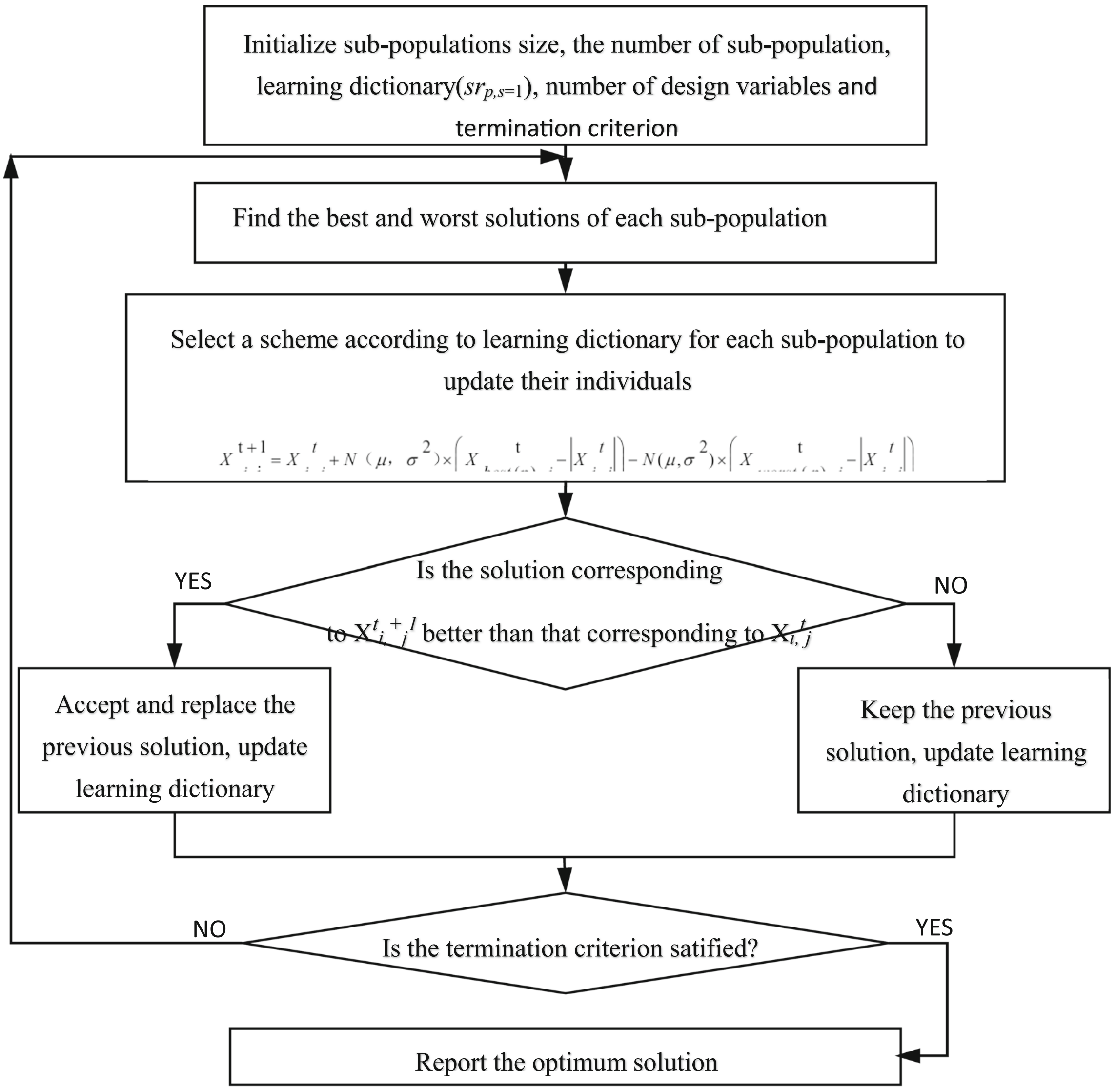
Initialize sub-populations size, the number of sub-population, learning dictionary($sr_{p,s=1}$), number of design variables and termination criterion
Find the best and worst solutions of each sub-population
Select a scheme according to learning dictionary for each sub-population to update their individuals
$X^{t+1}_{i,j} = X^{t}_{i,j} + N(\mu, \sigma^2) \times \left(X^{t}_{best(p),j} - \left| X^{t}_{i,j} \right| \right) - N(\mu, \sigma^2) \times \left(X^{t}_{worst(p),j} - \left| X^{t}_{i,j} \right| \right)$
Is the solution corresponding to $X^{t+1}_{i,j}$ better than that corresponding to $X^{t}_{i,j}$
YES
NO
Accept and replace the previous solution, update learning dictionary
Keep the previous solution, update learning dictionary
Is the termination criterion satified?
NO
YES
Report the optimum solution

Fig. 2 The flowchart of the Jaya-SCLMP algorithm

flowchart of the Jaya algorithm (Fig. 1) and the flowchart of the Jaya-SCLMP algorithm (Fig. 2), we can know that the difference is Jaya-SCLMP algorithm need select a scheme according to learning dictionary for each subpopulation to update their individuals. Assume the population size is NP, the dimension is D, to find the best and worst solution need time is TB, F_t is the objective function computational time, initialize each decision variables need time is TI, the implement iteration is K, £ denotes the updating time, then the computation time of the Jaya as follows:

$$\mathrm{TQ} = \mathrm{TI} + \mathrm{K} \times (\mathrm{TB} + \mathrm{NP} \times (\mathrm{D} + \mathrm{F}_t + \text{£})) \tag{3}$$

$$\mathrm{T1} = \lim_{K\to\infty} \frac{TQ}{K} = \mathrm{TB} + \mathrm{NP} \times (\mathrm{D} + \ \mathrm{Ft} + \text{£}) \tag{4}$$

Obviously, assume the same condition, the selection operation needs W time, then the computation time of the Jaya-SCLMP algorithm as follows:

$$\mathrm{TP} = \mathrm{TI} + K \times (\mathrm{TB} + 4 \times (W + \mathrm{NP}/4 \times (D + F_t + \text{£}))) \tag{5}$$

$$T2 = \lim_{K\to\infty} \frac{\mathrm{TP}}{K} = \mathrm{TB} + 4 \times (W + \mathrm{NP}/4 \times (D + Ft + \text{£})) \tag{6}$$

$$T2 - T1 = 4W \tag{7}$$

Baes on the above analysis, we realized that the proposed algorithm needs more time compared to the original algorithm, but it not relative to the iteration times, if the search mechanism is better, it can obtain a good solution in a few time. The experiment results are shown in Table 10. Under the same optimization accuracy, the computation effort of SCLMP-Jaya is better than the other compared algorithms.

5 Experiment results and analysis

5.1 Experimental settings

To test the performance of the proposed Jaya-SCLMP algorithm, the CEC 2013 and CEC 2015 unconstrained benchmark problems are considered in the experiment. Twenty-eight unconstrained benchmark problems with different characteristics including unimodal, multimodal and composition are selected from CEC 2013 test suite. These benchmark problems are briefly described in Table 3. The detailed description of these unconstrained benchmark problems can be found in Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session on Real-Parameter Optimization (Liang et al. 2013). The size of dimension D of these unconstrained benchmark problems is set to 30. In our experiments, Jaya-SCLMP is compared with Jaya algorithm (Rao and Waghmare 2016), MJaya (Ocłoń et al. 2018), QO-Jaya (Rao and Rai 2017), SAMP-Jaya (Rao and Saroj 2017a), GOTLBO (Chen et al. 2016) and GOPSO (Wang et al. 2011) in the experiments. GOTLBO and GOPSO use generalized oppositional teaching learning-based optimization to enhance the performance of basic algorithms. The subpopulation size of SCLMP-Jaya is 25. The other parameters of Jaya, MJaya, QO-Jaya, SAMP-Jaya, GOPSO and GOTLBO are set as the same as in their original papers. Due to the stochastic characteristics of EAs, we conduct 30 independent runs for each algorithm and each benchmark problem with 300,000 function evaluations (FEs) as the termination criterion. Moreover, we record the mean and standard deviation of the optimization error values ($f(X') - f(X^*)$) for evaluating the efficiency of the comparison algorithms, where X' is the best individual gained by the algorithm in a run and X^* is the global optimum of the benchmark problem.

In addition to 28 unconstrained benchmark problems, three example problems are selected to test the performance of the Jaya-SCLMP algorithm for reliability problems. Reliability problem is a kind of constrained optimization problems. According to the definition of the Advisory Group on the Reliability of Electronic Equipment, reliability indicates the probability implementing specific performance or function of products and achieving successfully the objectives within a time schedule under a certain environment. The reliability problem is usually formulated as a nonlinear programming problem, which is subject to several resource constraints such as cost, weight and volume. Various complex systems come out with the development of industrial engineering, and the reliability designing of these systems is very important. Thus, more accurate and efficient methods are needed in finding the optimal system reliability. Otherwise, the safety and efficiency of a system cannot be guaranteed. The three problems are a series system, series–parallel system and complex (bridge) system. Three reliability problems are described as in Wu et al. (2011), Ouyang et al. (2015).

P1. Complex (bridge) system (Fig. 3).

P1 is a nonlinear mixed integer programming problem for a complex (bridge) system with five subsystems, and this example is used to demonstrate the efficiency of Jaya-SCLMP algorithm. The complex (bridge) system optimization problem (Ouyang et al. 2015) is as follows:

Table 3 Summary of the 28 CEC 2013 Test Functions (search range [− 100,100])

	No	Functions	$f^*_i = f_i(x^*)$
Unimodal functions	1	Sphere function	− 1400
	2	Rotated high conditioned elliptic function	− 1300
	3	Rotated bent cigar function	− 1200
	4	Rotated discus function	− 1100
	5	Different powers function	− 1000
Basic multimodal functions	6	Rotated Rosenbrock's function	− 900
	7	Rotated Schaffer's F7 function	− 800
	8	Rotated Ackley's function	− 700
	9	Rotated Weierstrass function	− 600
	10	Rotated Griewank's Function	− 500
	11	Rastrigin's function	− 400
	12	Rotated Rastrigin's function	− 300
	13	Non-continuous Rotated Rastrigin's function	− 200
	14	Schwefel's function	− 100
	15	Rotated Schwefel's function	100
	16	Rotated Katsuura function	200
	17	Lunacek Bi_Rastrigin function	300
	18	Rotated Lunacek Bi_Rastrigin function	400
	19	Expanded Griewank's plus Rosenbrock's function	500
	20	Expanded Schaffer's F6 function	600
Composition functions	21	Composition function 1 (n = 5, rotated)	700
	22	Composition function 2 (n = 3, not rotated)	800
	23	Composition function 3 (n = 3, rotated)	900
	24	Composition function 4 (n = 3, rotated)	1000
	25	Composition function 5 (n = 3, rotated)	1100
	26	Composition function 6 (n = 5, rotated)	1200
	27	Composition function 7 (n = 5, rotated)	1300
	28	Composition function 8 (n = 5, rotated)	1400

$$\begin{aligned}
\max f(\mathbf{r},\mathbf{n}) &= R_1R_2 + R_3R_4 + R_1R_4R_5 + R_2R_3R_5 \\
&\quad - R_1R_2R_3R_4 - R_1R_2R_3R_5 \\
&\quad - R_1R_2R_4R_5 - R_1R_3R_4R_5 - R_2R_3R_4R_5 \\
&\quad + 2R_1R_2R_3R_4R_5 \\
s.t. g_1(\mathbf{r},\mathbf{n}) &= \sum_{i=1}^{m} w_i v_i^2 n_i^2 - V \le 0, \\
g_2(\mathbf{r},\mathbf{n}) &= \sum_{i=1}^{m} \alpha_i \left(-\frac{1000}{\ln(r_i)}\right)^{\beta_i} [n_i + \exp(0.25 n_i)] - C \le 0, \\
g_3(\mathbf{r},\mathbf{n}) &= \sum_{i=1}^{m} w_i n_i \exp(0.5 n_i) - W \le 0, \\
&0 \le r_i \le 1,\ n_i \in Z^+,\quad 1 \le i \le m.
\end{aligned} \tag{8}$$

Here, m is the number of subsystems in the system; n_i is the number of components in subsystem i, $(1 \le i \le m)$; r_i is the reliability of each component in subsystem i; $q_i = 1 - r_i$ is the failure probability of each component in subsystem i; $R_i(n_i) = 1 - q^{ni}_i$ is the reliability of subsystem i; $f(r, n)$ is the system reliability. w_i is the weight of each component in subsystem i, and v_i is the volume of each component in subsystem i; furthermore, V is the upper limit on the sum of the subsystems' products of volume and

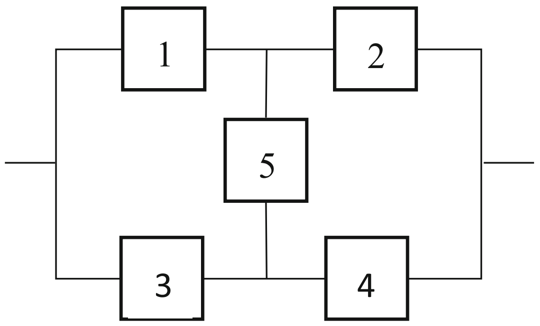

Fig. 3 The schematic diagram of complex (bridge) system (P1)

weight; C is the upper limit on the cost of the system; W is the upper limit on the weight of the system. The parameters β_i and α_i are physical features of system components. Constraint $g_1(r, n)$ is a combination of weight, redundancy allocation and volume; $g_2(r, n)$ is a cost constraint, while $g_3(r, n)$ is a weight constraint. The input parameters of the complex (bridge) system are shown in Table 4.

P2. Series system (Fig. 4).

P2 is a nonlinear mixed integer programming problem for a series system with five subsystems, and the problem formulation is as follows (Ouyang et al. 2018):

$$\begin{aligned}
&\max f(\mathbf{r},\mathbf{n}) = \prod_{i=1}^{m} R_i(n_i)\\
&s.t.\\
&g_1(\mathbf{r},\mathbf{n}) = \sum_{i=1}^{m} w_i v_i^2 n_i^2 - V \le 0,\\
&g_2(\mathbf{r},\mathbf{n}) = \sum_{i=1}^{m} \alpha_i \left(-\frac{1000}{\ln(r_i)}\right)^{\beta_i} [n_i + \exp(0.25 n_i)] - C \le 0,\\
&g_3(\mathbf{r},\mathbf{n}) = \sum_{i=1}^{m} w_i n_i \exp(0.5 n_i) - W \le 0,\\
&0 \le r_i \le 1,\ n_i \in Z^+,\ 1 \le i \le m.
\end{aligned} \tag{9}$$

P2 has three nonlinear constraints, and they are the same as P1. In addition, the input parameters of series system are also the same as those of the complex (bridge) system, and they are also shown in Table 4.

P3. Series–parallel system (Fig. 5).

P3 is a nonlinear mixed integer programming problem for a series–parallel system with five subsystems. The problem formulation is as follows:

Table 4 Data used in complex (bridge) system (P1) and series system (P2)

i	$10^5\alpha_i$	β_i	$w_i v_i^2$	w_i	V	C	W
1	2.330	1.5	1	7	110	175	200
2	1.450	1.5	2	8	110	175	200
3	0.541	1.5	3	8	110	175	200
4	8.050	1.5	4	6	110	175	200
5	1.950	1.5	3	9	110	175	200

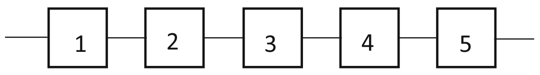

Fig. 4 Series system (P2)

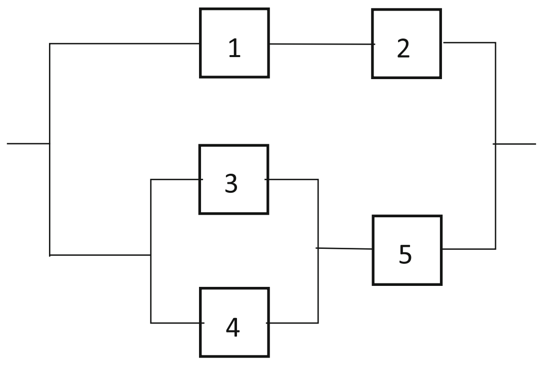

Fig. 5 Series–parallel system (P3)

$$\begin{aligned}
&\text{Max} f(r,n) = 1-(1-R_1R_2)(1-(R_3+R_4-R_3R_4)R_5)\\
&s.t.\\
&g1(r,n) = \sum_{i=1}^{m} w_i v_i^2 n_i^2 - V \le 0\\
&g2(r,n) = \sum_{i=1}^{m} \alpha_i \left(-\frac{1000}{\ln(r_i)}\right)^{\beta_i} [n_i + \exp(0.25 n_i)] - C \le 0\\
&g3(r,n) = \sum_{i=1}^{m} w_i n_i \exp(0.25 n_i) - W \le 0\\
&0 \le r_1 \le 1, n_i \in Z^+,\ \ 1 \le i \le m
\end{aligned} \tag{10}$$

P3 has the same nonlinear constraints as **P1**, but there are some differences in input parameters. The input parameters of **P3** are shown in Table 5.

To demonstrate the superiority of the Jaya_SCLMP algorithm in solving the reliability problems, we select the other five algorithms for comparison, and they are the Jaya algorithm, the SAMP_Jaya, the QO_Jaya, the GOPSO and the GOTLBO. The above three problems are used to compare performance of six algorithms on solving reliability problems. The parameters of the six algorithms are set as follows: For GOTLBO, population size NP = 50, jumping rate J_r = 1, teaching factor T_F = 1; For GOPSO, NP = 40, cognitive parameter c_1 = 1.49618, social parameter c_2 = 1.49618, inertia weight w = 0.72984, J_r = 0.3; For Jaya, NP = 10; For QO_Jaya, NP = 20; For

Table 5 Data used in series–parallel system (P3)

i	$10^5\alpha_i$	β_i	$w_i v_i^2$	w_i	V	C	W
1	2.500	1.5	2	3.5	180	175	100
2	1.450	1.5	4	4.0	180	175	100
3	0.541	1.5	5	4.0	180	175	100
4	0.541	1.5	8	3.5	180	175	100
5	2.100	1.5	4	4.5	180	175	100

Table 6 Experimental results of Jaya, MJaya, QO-Jaya and Jaya-SCLMP over 30 independent runs for the 28 unconstraint benchmark problems

Problems	Jaya			MJaya			QO-Jaya			Jaya_SCLMP	
	Mean	SD		Mean	SD		Mean	SD		Mean	SD
CO1	9.54E + 03	1.66E + 03	+	8.30E + 03	1.36E + 03	+	8.65E + 03	3.38E + 03	+	**6.47E + 00**	**2.67E + 00**
CO2	1.28E + 08	2.48E + 07	+	1.24E + 08	2.29E + 07	+	1.49E + 08	3.08E + 07	+	**5.67E + 05**	**2.28E + 05**
CO3	3.23E + 10	8.41E + 09	+	3.07E + 10	5.90E + 09	+	3.34E + 10	1.10E + 10	+	**2.67E + 09**	**4.15E + 09**
CO4	4.12E + 04	8.13E + 03	+	3.80E + 04	9.00E + 03	+	3.77E + 04	7.36E + 03	+	**1.32E + 04**	**3.78E + 03**
CO5	2.84E + 03	1.23E + 03	+	2.34E + 03	4.15E + 02	+	**5.71E + 03**	**1.47E + 04**	≈	**7.00E + 00**	**7.89E + 00**
CO6	5.05E + 02	9.45E + 01	+	4.75E + 02	8.06E + 01	+	5.55E + 02	3.05E + 02	+	**3.06E + 01**	**2.03E + 01**
CO7	1.64E + 02	3.00E + 01	+	1.59E + 02	1.87E + 01	+	1.60E + 02	2.16E + 01	+	**8.97E + 01**	**3.98E + 01**
CO8	**2.09E + 01**	**6.27E-02**	≈	**2.09E + 01**	**6.43E-02**	≈	**2.10E + 01**	**4.91E-02**	≈	**2.09E + 01**	**4.41E-02**
CO9	3.92E + 01	1.55E + 00	+	3.89E + 01	1.33E + 00	+	3.88E + 01	1.38E + 00	+	**3.12E + 01**	**2.47E + 00**
CO10	1.39E + 03	1.95E + 02	+	1.33E + 03	1.33E + 02	+	1.34E + 03	4.75E + 02	+	**2.35E + 00**	**5.75E-01**
CO11	3.59E + 02	2.81E + 01	+	3.44E + 02	1.93E + 01	+	4.94E + 02	2.75E + 02	+	**1.49E + 02**	**2.44E + 01**
CO12	3.56E + 02	4.73E + 01	+	3.34E + 02	2.15E + 01	+	3.84E + 02	9.94E + 01	+	**1.52E + 02**	**2.38E + 01**
CO13	3.38E + 02	2.96E + 01	+	3.54E + 02	4.64E + 01	+	3.59E + 02	7.97E + 01	+	**1.98E + 02**	**3.01E + 01**
CO14	**7.17E + 03**	**3.24E + 02**	≈	**7.17E + 03**	**3.72E + 02**	≈	**7.20E + 03**	**3.27E + 02**	≈	**5.03E + 03**	**5.88E + 02**
CO15	**7.29E + 03**	**3.20E + 02**	–	**7.26E + 03**	**2.74E + 02**	–	**7.21E + 03**	**3.01E + 02**	–	5.47E + 03	5.70E + 02
CO16	**2.48E + 00**	**2.79E-01**	≈	**2.51E + 00**	**3.39E-01**	≈	**2.53E + 00**	**2.55E-01**	≈	**2.43E + 00**	**3.16E-01**
CO17	5.58E + 02	1.48E + 02	+	5.43E + 02	1.35E + 02	+	4.96E + 02	1.63E + 02	+	**2.42E + 02**	**2.99E + 01**
CO18	5.64E + 02	1.33E + 02	+	5.47E + 02	1.21E + 02	+	4.94E + 02	1.63E + 02	+	**2.73E + 02**	**2.07E + 01**
CO19	3.14E + 03	1.35E + 03	+	2.60E + 04	7.83E + 04	≈	2.07E + 05	3.12E + 05	+	**2.03E + 01**	**2.23E + 00**
CO20	**1.37E + 01**	**3.95E-01**	–	**1.37E + 01**	**2.99E-01**	–	1.47E + 01	5.85E-01	≈	1.48E + 01	5.98E-01
CO21	2.06E + 03	2.90E + 02	+	1.98E + 03	3.81E + 02	+	2.16E + 03	3.26E + 02	+	**4.16E + 02**	**1.24E + 02**
CO22	**7.70E + 03**	**4.58E + 02**	≈	**7.72E + 03**	**4.62E + 02**	≈	**7.56E + 03**	**4.01E + 02**	≈	**5.61E + 03**	**4.72E + 02**
CO23	**7.50E + 03**	**2.98E + 02**	–	**7.45E + 03**	**3.39E + 02**	–	7.63E + 03	2.50E + 02	≈	5.58E + 03	5.09E + 02
CO24	3.05E + 02	3.61E + 00	+	3.06E + 02	2.92E + 00	+	3.07E + 02	4.27E + 00	+	**2.83E + 02**	**7.32E + 00**
CO25	3.11E + 02	5.70E + 00	+	3.08E + 02	5.21E + 00	+	3.10E + 02	4.78E + 00	+	2.97E + 02	7.14E + 00
CO26	3.06E + 02	8.93E + 01	≈	2.61E + 02	7.86E + 01	–	**2.32E + 02**	**4.60E + 01**	–	2.94E + 02	8.89E + 01
CO27	1.31E + 03	3.88E + 01	+	1.32E + 03	2.29E + 01	+	1.31E + 03	3.74E + 01	+	**1.09E + 03**	**5.15E + 01**
CO28	2.41E + 03	1.57E + 02	+	2.40E + 03	1.36E + 02	+	2.34E + 03	2.49E + 02	+	**5.99E + 02**	**4.58E + 02**
+			20			19			19		
–			3			4			2		
≈			5			5			7		

Table 7 Experimental results of SAMP-Jaya, GOPSO, GOTLBO and Jaya-SCLMP over 30 independent runs for the 28 unconstraint benchmark problems

Problems	SAMP_Jaya			GOTLBO			GOPSO			Jaya_SCLMP	
	Mean	SD		Mean	SD		Mean	SD		Mean	SD
CO1	1.44E + 04	2.22E + 03	+	3.75E + 04	4.35E + 03	+	2.27E + 04	5.21E + 03	+	**6.47E + 00**	**2.67E + 00**
CO2	1.59E + 08	4.29E + 07	+	4.55E + 08	1.03E + 08	+	3.29E + 08	9.72E + 07	+	**5.67E + 05**	**2.28E + 05**
CO3	5.20E + 10	1.33E + 10	+	3.99E + 14	7.83E + 14	+	2.83E + 12	7.94E + 12	+	**2.67E + 09**	**4.15E + 09**
CO4	8.06E + 04	1.23E + 04	+	6.97E + 04	8.65E + 03	+	5.86E + 04	7.75E + 03	+	**1.32E + 04**	**3.78E + 03**
CO5	3.39E + 03	6.45E + 02	+	9.88E + 03	2.63E + 03	+	3.68E + 03	1.57E + 03	+	**7.00E + 00**	**7.89E + 00**
CO6	8.77E + 02	1.95E + 02	+	5.02E + 03	9.42E + 02	+	3.03E + 03	9.31E + 02	+	**3.06E + 01**	**2.03E + 01**
CO7	1.99E + 02	3.00E + 01	+	1.11E + 04	9.54E + 03	+	7.07E + 02	5.10E + 02	+	**8.97E + 01**	**3.98E + 01**
CO8	2.10E + 01	6.36E − 02	+	**2.10E + 01**	**7.55E − 02**	≈	2.10E + 01	5.44E − 02	+	**2.09E + 01**	**4.41E − 02**
CO9	4.05E + 01	1.72E + 00	+	3.79E + 01	1.94E + 00	+	3.83E + 01	1.73E + 00	+	**3.12E + 01**	**2.47E + 00**
CO10	1.96E + 03	2.81E + 02	+	5.27E + 03	8.47E + 02	+	3.01E + 03	6.58E + 02	+	**2.35E + 00**	**5.75E-01**
CO11	4.15E + 02	3.91E + 01	+	6.68E + 02	6.81E + 01	+	5.12E + 02	4.35E + 01	+	**1.49E + 02**	**2.44E + 01**
CO12	4.15E + 02	3.69E + 01	+	6.77E + 02	7.19E + 01	+	5.24E + 02	5.61E + 01	+	**1.52E + 02**	**2.38E + 01**
CO13	4.36E + 02	3.84E + 01	+	6.91E + 02	5.45E + 01	+	5.43E + 02	4.96E + 01	+	**1.98E + 02**	**3.01E + 01**
CO14	**7.91E + 03**	**3.39E + 02**	+	5.92E + 03	3.61E + 02	+	7.42E + 03	3.37E + 02	+	**5.03E + 03**	**5.88E + 02**
CO15	7.95E + 03	2.88E + 02	+	7.34E + 03	4.31E + 02	+	7.50E + 03	2.61E + 02	+	5.47E + 03	5.70E + 02
CO16	3.22E + 00	4.80E-01	+	**2.55E + 00**	**2.49E − 01**	≈	**2.41E + 00**	**2.70E − 01**	≈	**2.43E + 00**	**3.16E − 01**
CO17	5.80E + 02	4.95E + 01	+	1.09E + 03	1.19E + 02	+	6.21E + 02	6.03E + 01	+	**2.42E + 02**	**2.99E + 01**
CO18	5.79E + 02	3.51E + 01	+	1.08E + 03	1.07E + 02	+	6.30E + 02	6.54E + 01	+	**2.73E + 02**	**2.07E + 01**
CO19	1.60E + 04	8.75E + 03	+	3.88E + 05	1.53E + 05	+	8.54E + 04	4.31E + 04	+	**2.03E + 01**	**2.23E + 00**
CO20	1.48E + 01	2.64E-01	≈	1.50E + 01	4.03E-02	≈	1.50E + 01	3.93E-12	≈	1.48E + 01	5.98E-01
CO21	1.44E + 04	2.22E + 03	+	2.90E + 03	2.31E + 02	+	2.12E + 03	1.75E + 02	+	**4.16E + 02**	**1.24E + 02**
CO22	1.59E + 08	4.29E + 07	+	6.72E + 03	3.74E + 02	+	7.82E + 03	4.09E + 02	+	**5.61E + 03**	**4.72E + 02**
CO23	5.20E + 10	1.33E + 10	+	7.75E + 03	4.73E + 02	+	8.08E + 03	2.86E + 02	+	5.58E + 03	5.09E + 02
CO24	8.06E + 04	1.23E + 04	+	3.59E + 02	1.23E + 01	+	3.17E + 02	5.67E + 00	+	**2.83E + 02**	**7.32E + 00**
CO25	3.39E + 03	6.45E + 02	+	3.61E + 02	1.27E + 01	+	3.36E + 02	8.47E + 00	+	2.97E + 02	7.14E + 00
CO26	8.77E + 02	1.95E + 02	−	2.64E + 02	2.70E + 01	≈	**2.25E + 02**	**1.02E + 01**	−	2.94E + 02	8.89E + 01
CO27	1.99E + 02	3.00E + 01	+	1.48E + 03	3.92E + 01	+	1.39E + 03	4.62E + 01	+	**1.09E + 03**	**5.15E + 01**
CO28	2.10E + 01	6.36E-02	+	5.39E + 03	4.93E + 02	+	3.96E + 03	4.12E + 02	+	**5.99E + 02**	**4.58E + 02**
+			26			24			25		
−			1			0			1		
≈			1			4			2		

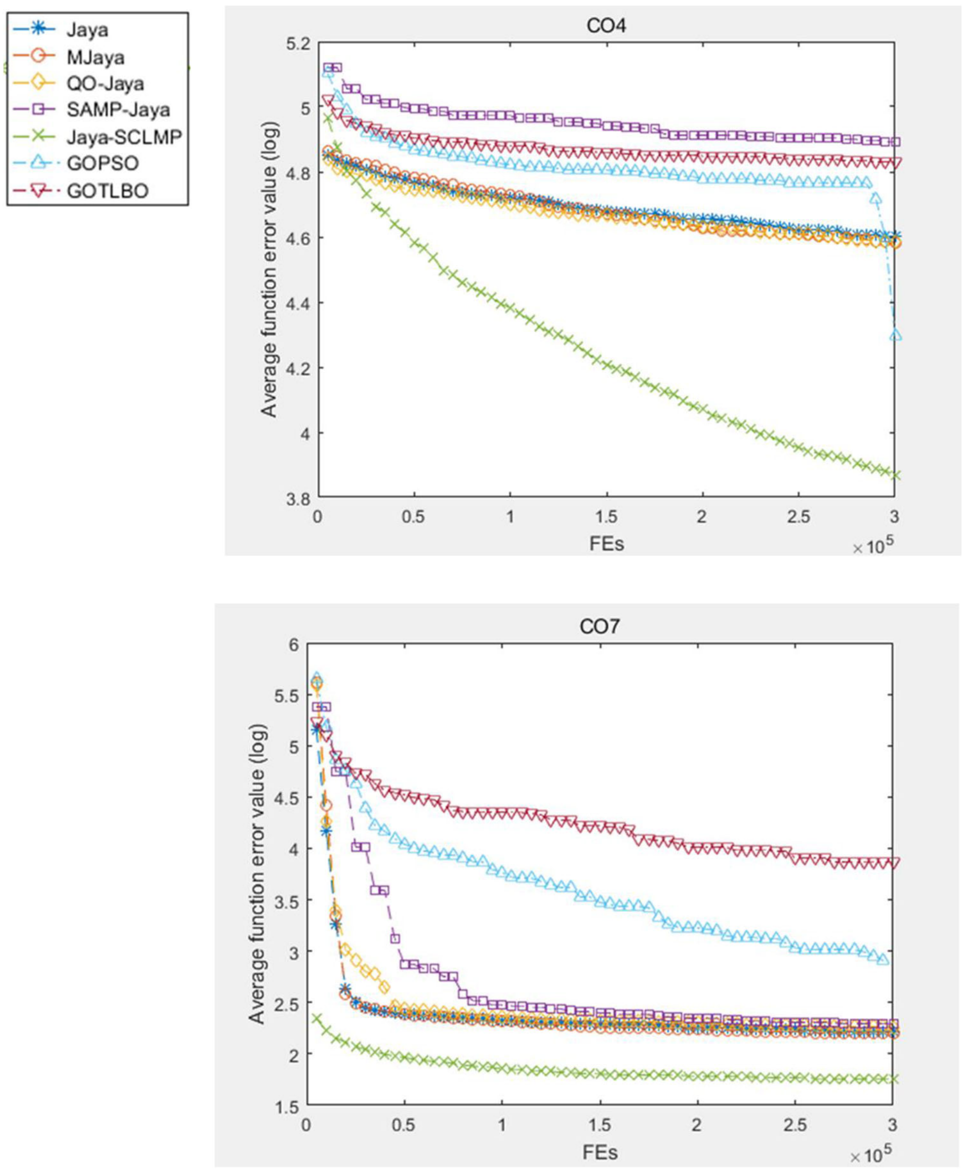

Fig. 6 Convergence curves of Jaya, MJaya, QO-Jaya, SAMP-Jaya, Jaya_SCLMP, GOPSO and GOTLBO for some typical benchmark functions

SAMP_Jaya, NP = 300, the minimal number of subpopulations is 2, the maximal number of subpopulations is 10; For Jaya_SCLMP, subpopulations size sNP = 25. In addition, we adopt a penalty function method to handle constraints. It is well known that the maximization of $f\,(r,\,n)$ can be transformed into the minimization of $-f\,(r,\,n)$; thus, the penalty function is described as:

$$\min F(r,n) = -f(r,n) + \lambda \sum_{j=1}^{ng} [\max(0, g_i(r,n))]^2 \qquad (11)$$

where λ represents penalty coefficient and it is set to 10^{10} in this paper. Due to the stochastic characteristics of EAs, we conduct 30 independent runs for each algorithm and each test problem with 15,000 function evaluations (FEs) as the termination criterion.

5.2 Results discussion and analysis

The mean and standard deviation of the optimization error values achieved by each algorithm for 28 unconstraint benchmark problems CO1 $\sim$ CO28 are shown in Tables 6 and 7. For convergence, the best results among all algorithms are highlighted in overstriking. In order to obtain statistically sound conclusions, the two-tailed t test at a 0.05 significance level is performed on the experimental results (Wang et al. 2011; Yao et al. 1999). The summary of the comparison results is shown in the last three rows of Tables 6 and 7. "Mean" and "SD" represent the mean and standard deviation of the optimization error values achieved by 30 independent runs, respectively. The symbols " + ", " − ", and "$\approx$" denote that Jaya-SCLMP achieves better, worse and similar results, respectively,

Fig. 6 continued

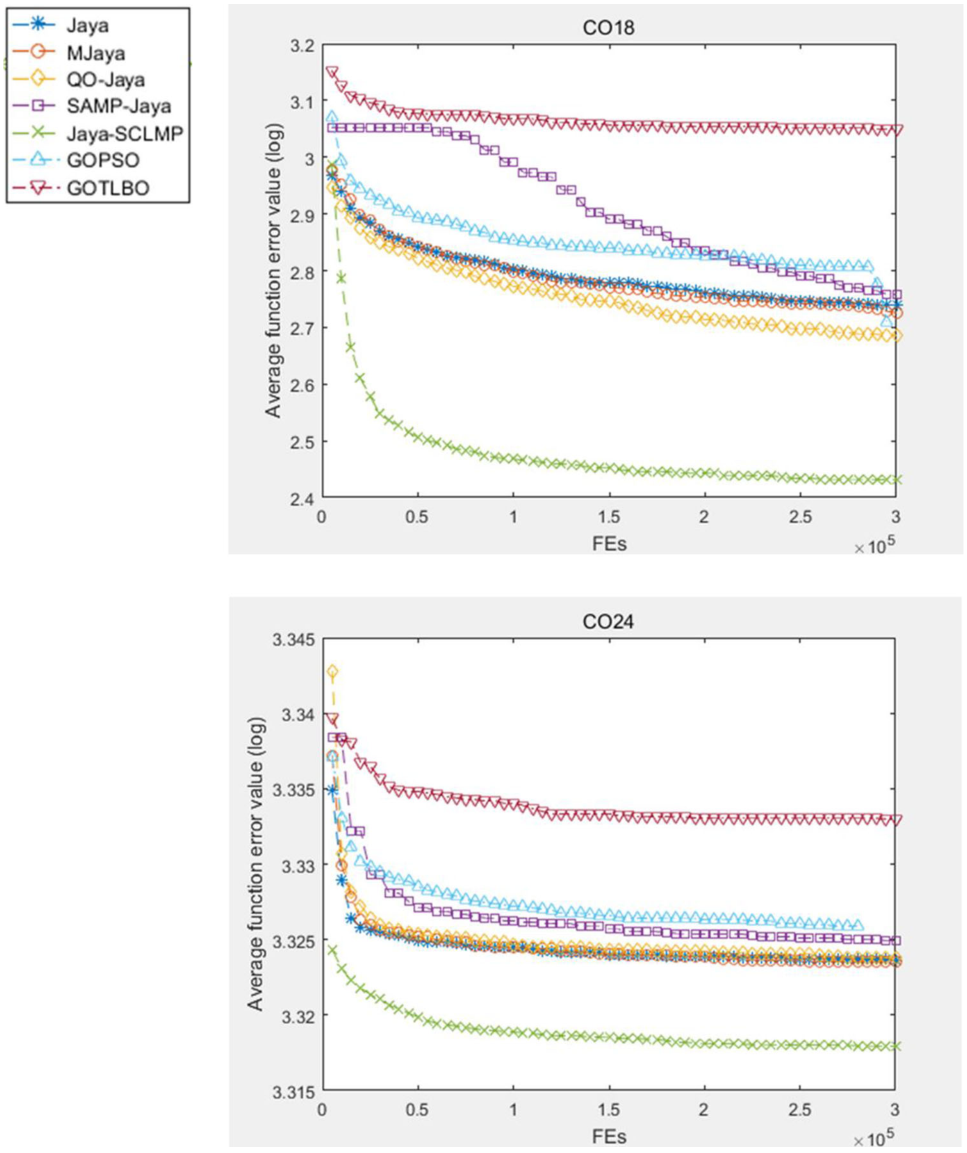

Table 8 Average rankings of the seven algorithms for 28 unconstraint benchmark problems gained by the Friedman test

Algorithm	Ranking
Jaya-SCLMP	1.95
MJaya	2.95
Jaya	3.36
QO-Jaya	3.52
SAMP-Jaya	5.18
GOPSO	5.23
GOTLBO	5.82

than the corresponding algorithms according to the two-tailed t test.

Based on the experimental results in Tables 6 and 7, we can see that Jaya-SCLMP is significantly better than Jaya, MJaya, QO-Jaya, SAMP-Jaya algorithms for the majority of the benchmark problems. Specifically, Jaya-SCLMP achieves better results than Jaya on 20 out of 28 benchmark problems. For the remaining eight benchmark problems, Jaya-SCLMP performs similarly to Jaya on benchmark problems CO8, CO14, CO16, CO22 and CO26 while Jaya performs better than Jaya-SCLMP on benchmark problems CO15, CO20 and CO23. MJaya outperforms Jaya-SCLMP on 4 benchmark problems (namely CO15, CO20, CO23 and CO26), while SCLMP-Jaya achieves better results than MJaya on 19 benchmark problems. Both Jaya-SCLMP and MJaya exhibit almost similar performance on 5 benchmark problems. For QO-Jaya, it performs similarly to Jaya-SCLMP on 7 benchmark problems. In addition, QO-Jaya is better than Jaya-SCLMP on CO15 and CO26, while Jaya-SCLMP outperforms QO-Jaya on 19 benchmark problems. Moreover, SAMP-Jaya surpasses Jaya-SCLMP on CO26. In contrast, Jaya-SCLMP is better than SAMP-Jaya on 26 out of 28 benchmark problems. Both SAMP-Jaya and Jaya-

Table 9 Comparative results for mean error obtained by different approaches for CEC' 15 problems

Function	Dimension	DE	$(\mu + \lambda)$ES	CMAES-S	CMAES-G	EPSO	Jaya	SAMP-Jaya	Jaya-SCLMP
FCEC1	10	3.4143E + 09	2.6325E + 09	3.6620E + 07	6.5990E + 07	1.5785E + 09	1.3175E + 07	**6.4037E + 06**	4.08E + 07
	30	2.3911E + 10	3.5775E + 10	**6.8700E + 07**	1.1080E + 08	8.4866E + 09	7.1308E + 08	3.02517E + 08	2.13E + 08
FCEC2	10	7.4931E + 04	4.8418E + 04	5.8080E + 04	1.0240E + 05	1.8953E + 04	5.4161E + 03	**1.9699E + 03**	4.00E + 03
	30	1.8254E + 05	1.6179E + 05	2.3630E + 05	2.9530E + 05	6.3748E + 04	1.3021E + 04	**1.1894E + 03**	1.13E + 04
FCEC3	10	3.1093E + 02	3.1048E + 02	6.1200E + 02	6.1570E + 02	3.1009E + 02	3.0566E + 02	3.0021E + 02	**2.08E + 01**
	30	3.4190E + 02	3.4353E + 02	6.3390E + 02	6.5270E + 02	3.3800E + 02	3.0231E + 02	3.0114E + 02	**2.12E + 01**
FCEC4	10	2.2974E + 03	1.4368E + 03	3.1890E + 03	4.1090E + 03	2.0662E + 03	4.6250E + 02	4.2458E + 02	**1.68E + 02**
	30	7.9627E + 03	7.0557E + 03	8.6730E + 03	1.2040E + 04	6.6946E + 03	8.2815E + 02	7.77849E + 02	**5.56E + 02**
FCEC5	10	5.0286E + 02	5.0318E + 02	1.0010E + 03	1.0060E + 03	5.0305E + 02	**5.0001E + 02**	5.0007E + 02	2.20E + 03
	30	5.0431E + 02	5.0499E + 02	1.0010E + 03	1.0080E + 03	5.0430E + 02	**5.0000E + 02**	**5.0000E + 02**	8.86E + 03
FCEC6	10	6.0286E + 02	6.0223E + 02	1.2010E + 03	1.2010E + 03	6.0234E + 02	6.0528E + 02	**6.0175E + 02**	2.14E + 02
	30	6.0365E + 02	6.0433E + 02	1.2010E + 03	1.2010E + 03	6.0276E + 02	**6.0209E + 02**	6.07713E + 02	6.71E + 02
FCEC7	10	7.2588E + 02	7.1668E + 02	1.4010E + 03	1.4020E + 03	7.1725E + 02	734.419551	706.745007	**1.06E + 02**
	30	7.5438E + 02	7.8216E + 02	1.4010E + 03	1.4010E + 03	7.2189E + 02	1.2025E + 03	1062.726345	**4.81E + 02**
FCEC8	10	4.1637E + 03	**1.1691E + 03**	1.6130E + 03	1.6480E + 03	1.6412E + 03	2.109E + 03	1.760E + 03	1.23E + 03
	30	7.9963E + 05	7.3789E + 06	**1.7670E + 03**	2.3210E + 03	1.2746E + 05	1.400E + 05	7.109E + 05	1.26E + 05
FCEC9	10	9.0415E + 02	9.0414E + 02	1.8080E + 03	1.8080E + 03	9.0407E + 02	903.426016	903.105930	**1.39E + 02**
	30	9.1394E + 02	9.1408E + 02	1.8270E + 03	1.8280E + 03	9.1372E + 02	913.311482	913.281842	**4.29E + 02**
FCEC10	10	1.3622E + 06	1.4666E + 06	1.7440E + 05	1.7700E + 06	1.3052E + 06	43,885.7623	**22,782.9766**	1.49E + 06
	30	3.8759E + 07	9.5323E + 07	3.6310E + 06	1.4730E + 07	2.6363E + 07	**31,356.0475**	85,248.5215	5.73E + 07
FCEC11	10	1.1229E + 03	1.1150E + 03	2.2120E + 03	2.2190E + 03	1.1140E + 03	1119.28310	1118.847733	**4.74E + 02**
	30	1.2870E + 03	1.4378E + 03	2.2460E + 03	2.2580E + 03	1.2288E + 03	1165.52400	**1164.50069**	1.60E + 03
FCEC12	10	1.5980E + 03	1.5797E + 03	2.7390E + 03	2.9810E + 03	1.5291E + 03	1423.57853	1352.880566	**1.52E + 02**
	30	3.0110E + 03	3.8087E + 03	3.4540E + 03	4.0940E + 03	2.4432E + 03	1490.04466	1284.020952	**2.06E + 02**
FCEC13	10	1.7969E + 03	1.6663E + 03	3.2580E + 03	3.3000E + 03	1.6932E + 03	1663.94774	1663.187051	**2.78E + 02**
	30	1.9613E + 03	2.2208E + 03	3.3840E + 03	3.4260E + 03	1.8839E + 03	1716.39830	1700.033267	**9.05E + 01**
FCEC14	10	1.6135E + 03	1.6148E + 03	3.2090E + 03	3.2170E + 03	1.6064E + 03	1446.11153	**1426.16411**	1.79E + 03
	30	1.7479E + 03	1.8406E + 03	3.2660E + 03	3.3000E + 03	1.7016E + 03	**1675.18244**	1699.661232	1.96E + 04
FCEC15	10	1.9452E + 03	1.9740E + 03	3.7770E + 03	3.9020E + 03	1.8662E + 03	1863.52977	1862.93521	**1.17E + 03**
	30	2.9304E + 03	2.9154E + 03	4.4270E + 03	4.8360E + 03	2.7488E + 03	**1932.59722**	1936.961342	1.58E + 04

Bold values show better solutions

Table 10 Results of the three examples using six algorithms

System	Algorithm	Best	Median	Worst	Mean	SD	NFOS
Complex(bridge)system	Jaya_SCLMP	0.999883549	**0.999833971**	**0.999461664**	**0.999797487**	**9.42711E-05**	30
	Jaya	**0.999889346**	0.999437835	0.961121253	0.996131898	0.00908008	30
	SAMP_Jaya	0.99940552	0.997584964	0.959870003	0.993467548	0.009196379	28
	QO_Jaya	0.999884492	0.999618557	0.96110165	0.997024407	0.007636702	30
	GOPSO	0.999775289	0.985948626	0.845410525	0.973898103	0.031632413	30
	GOTLBO	0.999813211	0.9994845	0.998192219	0.99938652	0.000373204	30
Series system	Jaya_SCLMP	0.930857191	**0.921915622**	**0.812318905**	**0.913923462**	**0.025567467**	30
	Jaya	0.927192502	0	0	0.330671641	0.4141113	16
	SAMP_Jaya	0.778022092	0.2207907	0	0.262025706	0.270740581	18
	QO_Jaya	**0.931625154**	0.896286221	0	0.615559881	0.41828363	21
	GOPSO	0.85329627	0.518145845	0.419909711	0.560191097	0.087762806	30
	GOTLBO	0.890073556	0.770805532	0.6107089	0.770191226	0.073212912	30
Series–parallel system	Jaya_SCLMP	**0.999971474**	**0.999937817**	**0.999540351**	**0.999921054**	**7.58188E-05**	30
	Jaya	0.999957072	0.999815828	0.979288961	0.998992035	0.00376119	30
	SAMP_Jaya	0.999657598	0.997427065	0.775324133	0.989118538	0.040610439	13
	QO_Jaya	0.999969394	0.999862169	0.956967758	0.998096611	0.00786544	30
	GOPSO	0.999931245	0.986080476	0.882912107	0.98174293	0.027307023	30
	GOTLBO	0.999919509	0.999609273	0.999091319	0.999534884	0.000228179	30

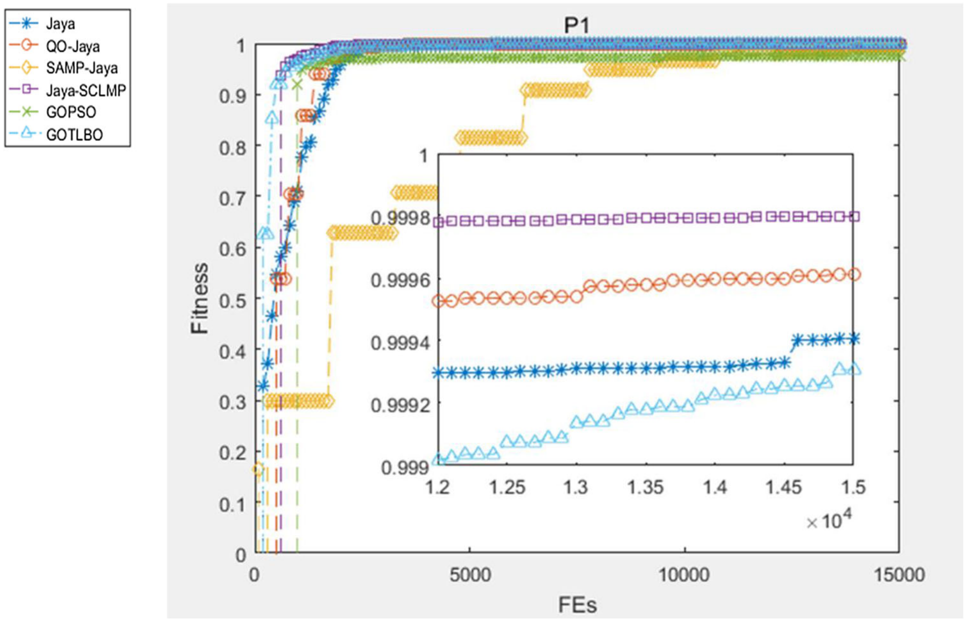

Fig. 7 The result of P1 using six algorithms

SCLMP demonstrate similar performance on benchmark problems CO20. From the comparison results among Jaya-SCLMP, Jaya, MJaya, QO-Jaya and SAMP-Jaya, it is known that the multi-population strategy and the commensal learning strategy work together to improve the performance of Jaya-SCLMP. It can be known that Jaya-SCLMP outperforms GOPSO and GOTLBO 25 and 24 out of 28 benchmark problems, respectively. GOPSO is better than Jaya-SCLMP on 1 benchmark problem. GOTLBO cannot be better than Jaya-SCLMP on any benchmark problems. In addition, Jaya-SCLMP is similar to GOPSO and GOTLBO on 2 and 4 benchmark problems. Thus, Jaya-SCLMP is significantly better than many algorithms on the majority of the benchmark problems.

Figure 6 shows the convergence curves of Jaya-SCLMP, MJaya,QO-Jaya, SAMP-Jaya, GOPSO and GOTLBO for

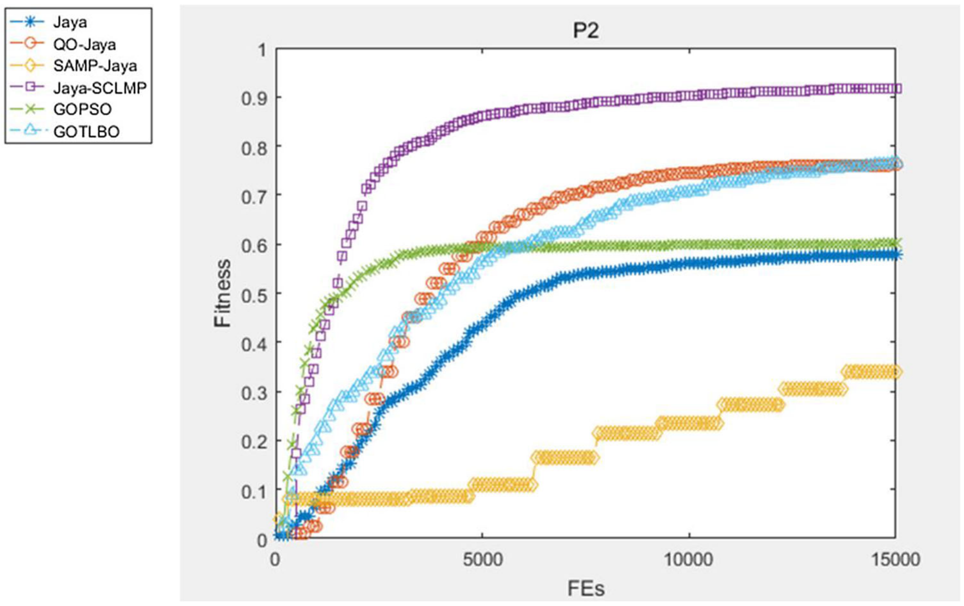

Fig. 8 The result of P2 using six algorithms

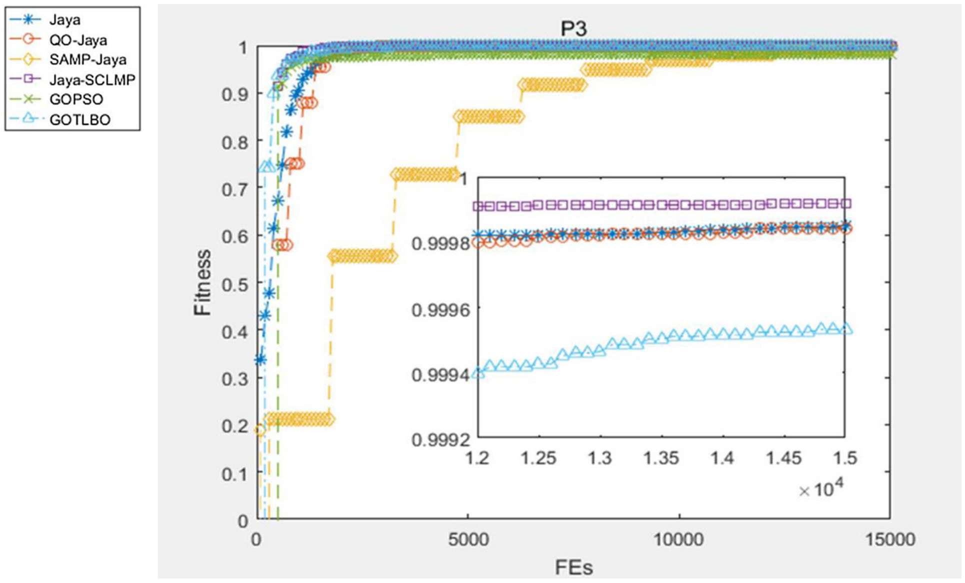

Fig. 9 The result of P3 using six algorithms

some typical benchmark problems. It can be seen from Fig. 6 that Jaya-SCLMP exhibits faster convergence speed than Jaya, MJaya, QO-Jaya, SAMP-Jaya, GOPSO and GOTLBO. The outstanding convergence performance of Jaya-SCLMP should be due to the incorporated multi-populations and commensal learning strategy, which can enhance the search ability. In order to compare the total performance of multiple algorithms on all benchmark problems, the Friedman test is conducted on the experimental results following (Yao et al. 1999). The average rankings of the seven algorithms for all benchmark problems are shown in Table 8. The best average ranking among the comparison algorithms is shown in italics. The seven algorithms can be sorted by the average ranking into the following order: Jaya-SCLMP, MJaya, Jaya, QO-Jaya, GOPSO, SAMP-Jaya and GOTLBO. Jaya-SCLMP rank first, which exhibits better total performance than MJaya, Jaya, QO-Jaya, GOPSO, SAMP-Jaya and GOTLBO on all benchmark problems. The better performance of Jaya-SCLMP can be because both the multi-population scheme and the commensal learning strategy work together to improve the performance of Jaya-SCLMP.

In order to further show the performance of the proposed algorithm, according to the literature (Rao and Saroj 2017a), the summary of the CEC 2015 expensive optimization test problem is used in our paper. The detail of the 15 CEC 2015 test functions can be seen in the literature (Rao and Saroj 2017a) (Appendix A2). Based on the

Table 11 Comparison of the number of objective function evaluations

Function	SCLMP-Jaya		Jaya		MJaya		QO-Jaya		SAMP-Jaya		SAMPE-Jaya	
	Mean	STD	Mean	STD	Mean	STD	Mean	STD	Mean	STD	Mean	STD
FCEC1	3.00E + 01	8.33E + 04	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05
FCEC2	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05
FCEC3	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05
FCEC4	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05
FCEC5	3.00E + 01	6.72E + 04	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05
FCEC6	3.00E + 01	2.82E + 04	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05
FCEC7	3.00E + 01	6.03E + 04	1.00E + 05	1.00E + 05	3.00E + 01	9.00E + 04	1.00E + 05	1.00E + 05	3.00E + 01	9.67E + 04	1.00E + 05	1.00E + 05
FCEC8	1.20E + 01	1.20E + 01	3.00E + 01	3.00E + 01	3.00E + 01	3.00E + 01	4.50E + 01	4.50E + 01	3.00E + 01	3.00E + 01	3.00E + 01	3.00E + 01
FCEC9	1.20E + 01	1.20E + 01	3.00E + 01	3.00E + 01	3.00E + 01	3.00E + 01	4.50E + 01	4.50E + 01	3.00E + 01	3.00E + 01	3.00E + 01	3.00E + 01
FCEC10	3.00E + 01	5.76E + 04	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05	1.00E + 05

reference (Rao and Saroj 2017a), maximum function evaluations (MFE) of 500 and 1500 are considered as one of the stopping criteria for 10-dimensional and 30-dimensional problems, respectively. All the experiment conditions are same to the reference (Rao and Saroj 2017a). The results obtained by our proposed algorithm are compared with cooperative PSO with stochastic movements (EPSO), DE, ($\mu + \lambda$)-evolutionary strategy (ES), specialized and generalized parameters experiments of covariance matrix adaption evolution strategy (CMAES-S and CMAES-G). The comparison of the computational results is presented in Table 9.

From Table 9, it can be seen that the performance of the proposed algorithm is superior to DE algorithm in 12 problems. Only 3 functions, DE algorithm achieves better results for all the functions with 10 and 30 dimensions. Therefore, the proposed algorithm has better performance than DE algorithm. In contrast to Jaya algorithm, our proposed algorithm performs significantly better in 11 problems. Besides the rest of 4 functions, the proposed algorithm achieves similar results as Jaya algorithm. Compared to ($\mu + \lambda$)ES algorithm, the proposed algorithm have better results in 11 functions. The proposed algorithm also get 11 better results compared to CMAES-S and CMAES-G algorithm. Compared to our proposed algorithm, EPSO have better results only 3 functions in 15 functions. SAMP-Jaya perform better in 5 functions, and similar in 2 functions, but worse in 8 cases. According to the experimental results, it can be observed that the proposed algorithm performs better on the more complex shifted and rotated problems. In all, the proposed algorithm has some advantages compared to all the compared algorithms.

The comparison of three example problems between the optimization results of the Jaya_SCLMP and those of the five other algorithms are presented in Table 10. NFOS represents the number of feasible "optima" solution found out of 30 runs and SD represents standard deviation. For convenience, the best results among all algorithms are highlighted in overstriking. Based on the experimental results in Table 10, we can see that for P1 and P3, the best, worst, mean results obtained by the Jaya_SCLMP are all very close to each other in each case, and the standard deviations are 9.42711E−05 and 7.58188E−05, respectively. From Figs. 7, 8 and 9, it can be seen that the Jaya_SCLMP has strong convergence and stability than five other algorithms. In addition, the mean results for P1 and P3 using Jaya_SCLMP algorithm are 0.9997974871 and 0.9999210543, respectively, and both mean results are better than those obtained by five other algorithms. For P2, the best result obtained by QO_Jaya is better than the best result obtained by Jaya_SCLMP, but the mean result obtained by Jaya_SCLMP is better than those obtained by five other algorithms. In addition, for P2, we can see that Jaya, QO_Jaya and SAMP_Jaya are easy to be trapped in local optima. For P1 and P3, the Jaya_SCLMP obtains larger NFOS than the SAMP_Jaya. For P2, the Jaya_SCLMP obtains larger NFOS than Jaya, QO_Jaya and SAMP_Jaya, and the same NFOS is obtained by Jaya_SCLMP, GOPSO and GOTLBO for **P1–P3**. On the whole, the Jaya_SCLMP has demonstrated a higher efficiency in finding an "optima" solution (or near "optima" solution) when compared to the other three Jaya variants.

In order to compare the search efficiency and running time of the proposed algorithm and the other different algorithms, we select a threshold value for each benchmark problems. To give a fair chance to all the metaheuristics compared. We run each algorithm on a function and stop as soon as the best value determined by the algorithm falls below the predefined threshold or a maximum number of OFEs is exceeded. Here, the threshold is 10^2, the maximum number of objective function evaluations (OFE) is fixed as 1×10^5. The experiment results are recorded in Table 11.

From Table 11, it can be obviously seen that SCLMP-Jaya take less number of objective function evaluations for the same predefined threshold compared to all the other algorithms, which is demonstrated that the SCLMP-Jaya has better search efficiency. In other words, under the same optimization accuracy, the computation effort of SCLMP-Jaya is better than the other compared algorithms.

6 Conclusion

Jaya algorithm has gained great success in science and engineering field. In order to enhance the performance of the Jaya algorithm, a self-adaptively commensal learning-based multi-populations Jaya algorithm, called Jaya-SCLMP, is proposed in this study. On the one hand, Jaya-SCLMP employs commensal learning strategy to accelerate the convergence speed. On the other hand, Jaya-SCLMP uses multi-populations scheme to increase the probability of finding the global optimum. In the numerical experiments, 28 benchmark test functions and three example problems of reliability problems are utilized to evaluate the performance of Jaya-SCLMP. Moreover, Jaya-SCLMP is compared with six algorithms, namely MJaya, Jaya, QO-Jaya, SAMP-Jaya, GOPSO and GOTLBO. The experimental results reveal that Jaya-SCLMP can exhibit better performance than these algorithms on the majority of the test functions and that the Jaya-SCLMP is superior to the other five algorithms in finding the maximal reliability for the three example problems. In the future research, we will utilize the proposed Jaya-SCLMP to solve other real-world optimization problems, such as constraint and multi-objective optimization problems.

Acknowledgements The authors are grateful to the editor and the anonymous referees for their constructive comments and recommendations, which have helped to improve this paper significantly. The authors would also like to express their sincere thanks to P. N. Suganthan for the useful information about metaheuristic algorithm and optimization problems on their homepages. We appreciate R.V. Rao for providing the original Jaya algorithm code. This work is supported by Guangzhou Science and Technology Plan Project (201804010299), National Nature Science Foundation of China (Grant No. 61806058), Nature Science Foundation of Guangdong province (2018A030310063).

Data availability All data used to support the findings of this study are available from our experiments. Upon request, please contact the author to provide them.

Declarations

Conflict of interest All the authors have no conflict of interests. This article does not contain any studies with human participants performed by any of the authors.

References

Akay B, Karaboga D (2012) A modified artificial bee colony algorithm for real-parameter optimization. Inform Sci 192:120–142

Azizi M, Ghasemi SAM, Ejlali RG et al (2019) Optimum design of fuzzy controller using hybrid ant lion optimizer and Jaya algorithm. Artif Intell Rev 53:1–32

Chen X, Yu K, Du W et al (2016) Parameters identification of solar cell models using generalized oppositional teaching learning based optimization. Energy 99:170–180

Deb K et al (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evolut Comput 6(2):182–197

Degertekin SO, Lamberti L, Ugur IB (2018) Sizing, layout and topology design optimization of truss structures using the Jaya algorithm. Appl Soft Comput 70:903–928

Degertekin SO, Lamberti L, Ugur IB (2019) Discrete sizing/layout/topology optimization of truss structures with an advanced Jaya algorithm. Appl Soft Comput 79:363–390

Eberhart R, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceedings of the sixth international symposium on micro machine and human science, 1995. MHS '95

Farah A, Belazi A (2018) A novel chaotic Jaya algorithm for unconstrained numerical optimization. Nonlinear Dyn 93(3):1451–1480

Gao K, Yang F, Zhou M et al (2019) Flexible job-shop rescheduling for new job insertion by using discrete Jaya algorithm. IEEE Trans Syst Man Cybern 49(5):1944–1955

Geem ZW, Kim JH, Loganathan GV (2001) A new heuristic optimization algorithm: harmony search. SIMULATION 76(2):60–68

Grzywinski M, Dede T, Ozdemir YI (2019) Optimization of the braced dome structures by using Jaya algorithm with frequency constraints. Steel Comp Struct 30(1):47–55

Huang C, Wang L, Yeung RS et al (2018) A prediction model-guided Jaya algorithm for the PV system maximum power point tracking. IEEE Trans Sustain Energy 9(1):45–55

Liang JJ, Qu BY, Suganthan PN et al (2013) Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization. Comput Intell Lab Zhengzhou Univ

Zhengzhou China Nanyang Technol Univ Singap Techn Rep 201212(34):281–295

Ocłoń P, Cisek P, Rerak M et al (2018) Thermal performance optimization of the underground power cable system by using a modified Jaya algorithm. Int J Therm Sci 123:162–180

Ouyang H, Gao L, Li S et al (2015) Improved novel global harmony search with a new relaxation method for reliability optimization problems. Inform Sci 305:14–55

Ouyang H, Wu W, Zhang C et al (2018) Improved harmony search with general iteration models for engineering design optimization problems. Soft Comput 23:1–36

Papa JP, Scheirer W, Cox DD (2016) Fine-tuning deep belief networks using harmony search. Appl Soft Comput 46:875–885

Peng H, Wu Z, Deng C et al (2017) Enhancing differential evolution with commensal learning and uniform local search. Chin J Elect 26(4):725–733

Rao RV (2019) Application of Jaya algorithm and its variants on constrained and unconstrained benchmark functions. Jaya: an advanced optimization algorithm and its engineering applications. Springer, Cham, pp 59–90

Rao RV, Keesari HS (2018) Multi-team perturbation guiding Jaya algorithm for optimization of wind farm layout. Appl Soft Comput 71:800–815

Rao RV, More KC (2017) Design optimization and analysis of selected thermal devices using self-adaptive Jaya algorithm. Energy Conv Manag 140:24–35

Rao RV, Rai DP (2017) Optimization of submerged arc welding process parameters using quasi-oppositional based Jaya algorithm. J Mech Sci Technol 31(5):2513–2522

Rao RV, Saroj A (2017a) A self-adaptive multi-population based Jaya algorithm for engineering optimization. Swarm Evolut Comput 37:1–26

Rao RV, Saroj A (2017b) Constrained economic optimization of shell-and-tube heat exchangers using elitist-Jaya algorithm. Energy 128:785–800

Rao RV, Saroj A (2018a) An elitism-based self-adaptive multi-population Jaya algorithm and its applications. Soft Comput 23:1–24

Rao RV, Saroj A (2018b) Constrained economic optimization of shell-and-tube heat exchangers using a self-adaptive multipopulation elitist-Jaya algorithm. J Therm Sci Eng Appl 10(4):041001

Rao RV, Saroj A (2018c) Multi-objective design optimization of heat exchangers using elitist-Jaya algorithm. Energy Syst 9(2):305–341

Rao RV, Waghmare GG (2016) A new optimization algorithm for solving complex constrained design optimization problems. Eng Optim 49(1):1–24

Rao RV, Savsani VJ, Vakharia DP (2011) Teaching-learning-based optimization: a novel method for constrained mechanical design optimization problems. Comput Aided Des 43(3):303–315

Rao RV, More KC, Taler J et al (2016) Dimensional optimization of a micro-channel heat sink using Jaya algorithm. Appl Therm Eng 103:572–582

Rashedi E, Nezamabadi-Pour H, Saryazdi S (2009) GSA: a gravitational search algorithm. Elsevier, Amsterdam

Storn R, Price KV (1997) Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. J Global Optim 11(4):314–359

Wang L, Huang C (2018) A novel Elite Opposition-based Jaya algorithm for parameter estimation of photovoltaic cell models. Optik 155:351–356

Wang H, Wu Z, Rahnamayan S et al (2011) Enhancing particle swarm optimization using generalized opposition-based learning. Inform Sci 181(20):4699–4714

Wang L, Zhang Z, Huang C et al (2018a) A GPU-accelerated parallel Jaya algorithm for efficiently estimating Li-ion battery model parameters. Appl Soft Comput 65:12–20

Wang SH, Muhammad K, Lv Y, et al (2018) Identification of Alcoholism based on wavelet Renyi entropy and three-segment encoded Jaya algorithm. Complexity 2018.

Warid W, Hizam H, Mariun N et al (2018) A novel quasi-oppositional modified Jaya algorithm for multi-objective optimal power flow solution. Appl Soft Comput 65:360–373

Wu P, Gao L, Zou D et al (2011) An improved particle swarm optimization algorithm for reliability problems. ISA Trans 50(1):71–81

Yao X, Liu Y, Lin G (1999) Evolutionary programming made faster. IEEE Trans Evolut Comput 3(2):82–102

Yu K, Liang JJ, Qu BY et al (2017) Parameters identification of photovoltaic models using an improved Jaya optimization algorithm. Energy Conv Manag 150:742–753

Yu K, Qu B, Yue C et al (2019) A performance-guided Jaya algorithm for parameters identification of photovoltaic cell and module. Appl Energy 237:241–257

Zhang YD, Zhao G, Sun J et al (2018) Smart pathological brain detection by synthetic minority oversampling technique, extreme learning machine, and Jaya algorithm. Multimedia Tools Appl 77(17):22629–22648