Soft Computing (2019) 23:4383–4406
https://doi.org/10.1007/s00500-018-3095-z

METHODOLOGIES AND APPLICATION

An elitism-based self-adaptive multi-population Jaya algorithm and its applications

R. Venkata Rao[1] · Ankit Saroj[1]

Published online: 14 March 2018

Abstract
This study proposes an elitist-based self-adaptive multi-population (SAMPE) Jaya algorithm to solve the constrained and unconstrained problems related to numerical and engineering optimization. The Jaya algorithm is a newly developed metaheuristic-based optimization algorithm, and it does not require any algorithmic-specific parameters to be set other than the common control parameters of number of iterations and population size. The search mechanism of the Jaya algorithm is improved in this work by using the subpopulation search scheme with elitism. It uses an adaptive scheme for dividing the population into subpopulations. The effectiveness of the proposed SAMPE-Jaya algorithm is verified on CEC 2015 benchmark problems in addition to fifteen unconstrained, six constrained standard benchmark problems and four constrained mechanical design optimization problems considered from the literature. The Friedman rank test is also done for comparing the performance of the SAMPE-Jaya algorithm with other algorithms. It is also tested on three large-scale problems with the dimensions of 100, 500 and 1000. Furthermore, the proposed SAMPE-Jaya algorithm is used for solving a case study of design optimization of a micro-channel heat sink. The computational experiments have proved the effectiveness of the proposed SAMPE-Jaya algorithm.

Keywords Multi-population Jaya algorithm · CEC 2015 · Heat sink

1 Introduction

Solving the complex optimization problems within the restricted time is a crucial subject in the field of engineering optimization. The conventional methods become tedious and time-consuming for solving the complex optimization problems, and these techniques are not having any guarantee to achieve the global optimal solution. Therefore, in order to overcome this issue, metaheuristic-based optimization methods are developed. These methods are having the capability to achieve the global or near global optimum solution within the limited time. Some of the well-known metaheuristic optimization algorithms are: genetic algorithm (GA) and its variants (parallel GA, real coded GA, hybrid interval GA, etc.), tabu search (TS), ant colony optimization (ACO), particle swarm optimization (PSO) and its variants (e.g., culture-based PSO, niching PSO, aging theory inspired PSO, etc.), differential evolution (DE) and its variants (e.g., DE with self-adapting control parameter, DE with multi-population ensemble, DE with optimal external archive), artificial bee colony (ABC) algorithm, imperialist competitive algorithm (ICA), biogeography-based optimization (BBO), gravitational search algorithm (GSA), firefly algorithm (FFA), cuckoo search (CS), bat algorithm (BA), etc. (Lau et al. 2016; Rao and Patel 2012; Irawan et al. 2016).

Several metaheuristic algorithms have been proposed in the last decade. Some prominent algorithms are: spiral optimization, differential search algorithm, teaching-learning-based optimization (TLBO), cuckoo search algorithm (CSA), colliding bodies optimization algorithm, whale optimization algorithm (WOA), centripetal accelerated particle swarm optimization algorithm, crisscross optimization algorithm, ant lion optimization (ALO), cat swarm optimization (CSO), bacteria forging optimization (BFO), thermal exchange optimization algorithm (TEOA), gray wolf optimizer (GWO) algorithm, chemotherapy science-based optimization algo-

Communicated by V. Loia.

✉ R. Venkata Rao
ravipudirao@gmail.com

1 Department of Mechanical Engineering, Sardar Vallabhbhai National Institute of Technology, Surat 395 007, India

rithm (CSOA) and hybrid algorithms (Mirjalili and Lewis 2016; Salmani and Eshghi 2017; Kohli and Arora 2017; Rao and Saroj 2017; Kaveh and Dadras 2017).

The advanced optimization algorithms are having their individual merits, but they require tuning of their own algorithmic-specific parameters. For example, GA requires proper tuning of crossover probability, selection operator, mutation probability. SA algorithm requires the tuning of cooling schedule and initial temperature of annealing. PSO requires the setting of social and cognitive parameters and inertia weight. BBO algorithm needs the setting of emigration rate, immigration rate, etc. Similarly, ICA, DE and other algorithms mentioned in the above paragraph are having their own algorithmic-specific parameters to be tuned for the effective execution of the algorithm. These parameters are known as algorithm-specific parameters and required to be set in addition to the common control parameters (i.e., number of iterations and population size and elite size). All the existing population-based advanced optimization algorithms are required to set values of the common control parameters, but the algorithm-specific parameters are specific to the specific algorithm and these are also to be set as explained above.

The performance of the metaheuristic-based algorithms is strongly influenced by the algorithmic-specific parameters. The appropriate setting of these parameters is very much necessary. The improper tuning of these parameters may lead to an increase in the computational cost or tending toward the local optimal solution. Therefore, to resolve the issue of setting of algorithm-specific parameters, teaching-learning-based-optimization (TLBO) algorithm was developed which is an algorithm-specific parameter-less algorithm (Rao 2016a, b). Keeping in view of the good performance of the TLBO algorithm, another algorithm-specific parameter-less algorithm has been recently proposed and it is named as Jaya algorithm (Rao 2016c).

It is to be mentioned here that the subpopulation-based advanced optimization methods have been developed by the researchers in order to improve the diversity of search process by dividing the whole population into sub-groups and assigning these throughout the search space. This maintains the diversity of the search mechanism by assigning subpopulations to various search areas rather than concentrating on single area. In this method, each subpopulation is associated with either exploring or exploiting the search processes of the algorithm (Nguyen et al. 2012; Cruz et al. 2011). The interface between the subpopulations is done by a combine and split process when there is an effective improvement in the current global solution is observed. The subpopulation-based algorithms are found more effective as compared to the single population-based algorithms.

Branke et al. (2000) developed the self-organizing scout's multi-population evolutionary algorithm. Li and Yang (2008) developed the multi-swarm PSO algorithm. Yang and Li (2010) developed the clustering-based PSO. Rao and Patel (2012) developed the multiple teachers TLBO. A multi-population ABC algorithm was developed by Nseef et al. (2016). These subpopulation-based methods are helpful to maintain the population diversity. The subpopulation-based optimization algorithms are advantageous (Li et al. 2015) mainly because of the reason that the overall diversity of the search mechanism can be maintained with distribution of the whole population into groups and various subpopulations can be located in different regions of search space.

The algorithm's performance is affected by the selection of number of subpopulations which is associated with the complexity of the problems. This point cannot be recognized in advance for the given problems, and it regularly changes during the search process. The solutions in the subpopulations may also not be enough for enough diversity. In order to address these issues, the present work proposes a self-adaptive multi-population elitist (SAMPE) Jaya algorithm. In order to effectively monitor the problem landscape, the SAMPE-Jaya algorithm adaptively changes the number of subpopulations based on change strength of the solution. The objectives of the present study are:

(a) To propose a SAMPE-Jaya algorithm that adapts the number of subpopulations based on the change strength of the problem.
(b) To investigate the performance of the proposed method on standard benchmark problems.
(c) To investigate the performance of the proposed method for an engineering application of a micro-channel heat sink design.

The basic difference between island-model GA and proposed SAMPE-Jaya algorithm is that the island-model GA uses only two groups (i.e., Master Island and Slave Islands). However, the proposed SAMPE-Jaya algorithm uses the subpopulations adoptively. Multiple-population (or multiple-deme, or parallel) GAs are more sophisticated, as they consist of several subpopulations which exchange individuals occasionally. This exchange of individuals is called migration, and it is controlled by several parameters. However, the number of demes in this method is to be tuned for the better performance of the algorithms. The tuning of number of subpopulations (number of demes) is a critical issue in the parallel evolutionary algorithm (Cantu-Paz 1998; Mambrini and Sudholt 2014). Therefore, this issue is resolved by the proposed SAMPE-Jaya algorithm which decides the number of subpopulation adaptively.

The following section describes the proposed self-adaptive multi-population elitist-Jaya algorithm.

2 Self-adaptive multi-population elitist-Jaya algorithm

Rao and Saroj (2017) developed self-adaptive multi-population (SAMP) Jaya algorithm, and the proposed self-adaptive multi-population elitist (SAMPE) Jaya algorithm is an extension of the SAMP-Jaya algorithm to improve its performance. Let $Z(x)$ is an objective which is being optimized. Assume that at any iteration i, number of design variables is ‘d’ (i.e., $q = 1, 2, \ldots, d$), population size ‘P’ (i.e., $r = 1, 2, \ldots, P$). If $X_{\mathrm{q,r,i}}$ is the value of the qth variable for the rth candidate during the ith iteration, then this value is modified based upon the following Eq. (2.1).

$$\boldsymbol{X'_{q,r,i}} = \boldsymbol{X_{q,r,i}} + \boldsymbol{r}_1(\boldsymbol{X_{q,best,i}} - |\boldsymbol{X_{q,r,i}}|) \\ -\boldsymbol{r}_2(\boldsymbol{X_{q,worst,i}} - |\boldsymbol{X_{q,r,i}}|) \tag{2.1}$$

where $X_{q,best,i}$ is the value of the qth variable for the best solution and $X_{q,worst,i}$ is the value of the qth variable for the worst solution in the population. $X'_{q,ri}$ is the new value of $X_{q,r,i}$ and r_1, r_2 are random numbers in the range of [0, 1]. The term “$r_1(X_{q,best,i} - |X_{q,r,i}\ \ |)$” indicates that the solution tries to approach the best solution, and the term “$-r_2(X_{q,worst,i} - |X_{q,r,i}|)$” shows that the solution tries to avoid the worst solution. $X'_{q,r,i}$ is accepted if function value produced by it is better.

In SAMPE-Jaya algorithm, the following modifications are added to the basic Jaya algorithm:

(a) The proposed algorithm uses number of subpopulations by dividing it into number of groups based on the quality of the solutions (value of fitness function). Furthermore, the worst solutions of the inferior group (populations having poor fitness values) are replaced by the solutions of the superior group such as populations having higher fitness values (elite solutions). Use of the number of subpopulations distributes the solution over the search space rather than concentrating in a particular area. Therefore, the proposed algorithm should produce optimum solution and monitor the problem landscape changes.

(b) During the search process, SAMPE-Jaya algorithm modifies the number of subpopulations based on change strength of the problem for monitoring the landscape changes. It means that the number of subpopulations will be increased or decreased. In the SAMPE-Jaya algorithm, the number of subpopulations is modified adaptively based on the strength of the solution change (e.g., improvement in the fitness value). This feature supports the search process for tracing the optimum solution and improving the exploration and diversification of the search process. Furthermore, the duplicate solutions are replaced by the newly generated solutions for maintaining the diversity and enhancing the exploration procedure.

The basic steps of the SAMPE-Jaya algorithm are as follows:

Step 1 It starts with the setting of the number of design variables (d), number of populations (P), elite size (ES) and termination criterion. (Termination criterion for the present work is maximum number of function evaluations ($\mathrm{FE}_{\max}$).)

Step 2 Next step is to calculate the initial solutions based on the defined fitness function for the defined problem.

Step 3 The entire population is grouped into m number of groups based on the quality of the solutions (initially m=2 is considered) and replace the worst solutions (equals to ES) of the inferior group with solutions of the superior group (elite solutions).

Step 4 Each subpopulation uses Jaya algorithm for modifying the solutions in each group independently. Modified solutions are kept *if and only if* these are better than the old solutions.

Step 5 Combine the entire subpopulation. Check whether *Z(best_before)* is better than *Z(best_after).*

Here, *Z(best_before)* is the previous best solution of the entire population and Z(*best_after*) is the current best solution in the entire population. If the value of Z*(best_after)* is better than the value of Z(*best_before*), m is increased by 1 ($m = m + 1$) with the aim of increasing the exploration feature of the search process. Otherwise, m is decreased by 1 ($m = m - 1$) as the algorithm needs to be more exploitive than explorative.

Step 6 Check the stopping condition(s). If the search process is reached to the maximum number of function evaluations, then terminate the loop and report the best optimum solution. Otherwise, follow the steps of:

(a) Replace the duplicate solutions with randomly generated new solutions
(b) For re-dividing the populations go to Step 3.

The flowchart of the proposed SAMPE-Jaya algorithm is presented in Fig. 1.

All the metaheuristic algorithms (except TLBO algorithm) require tuning of algorithmic-specific parameters. The tuning of these parameters is in addition to the common control parameters (e.g., population size, elite size and number of iterations). The performance of the metaheuristic algorithms is much affected by the tuning of these algorithmic-specific parameters. The improper tuning of the algorithmic-specific parameters may lead to local optimal solution.

The Jaya algorithm is an algorithmic-specific parameterless algorithm similar to teaching-learning-based optimization (TLBO) algorithm. However, it is much simpler and powerful than TLBO algorithm and it is having a single phase rather than two phases of TLBO (i.e., teacher phase

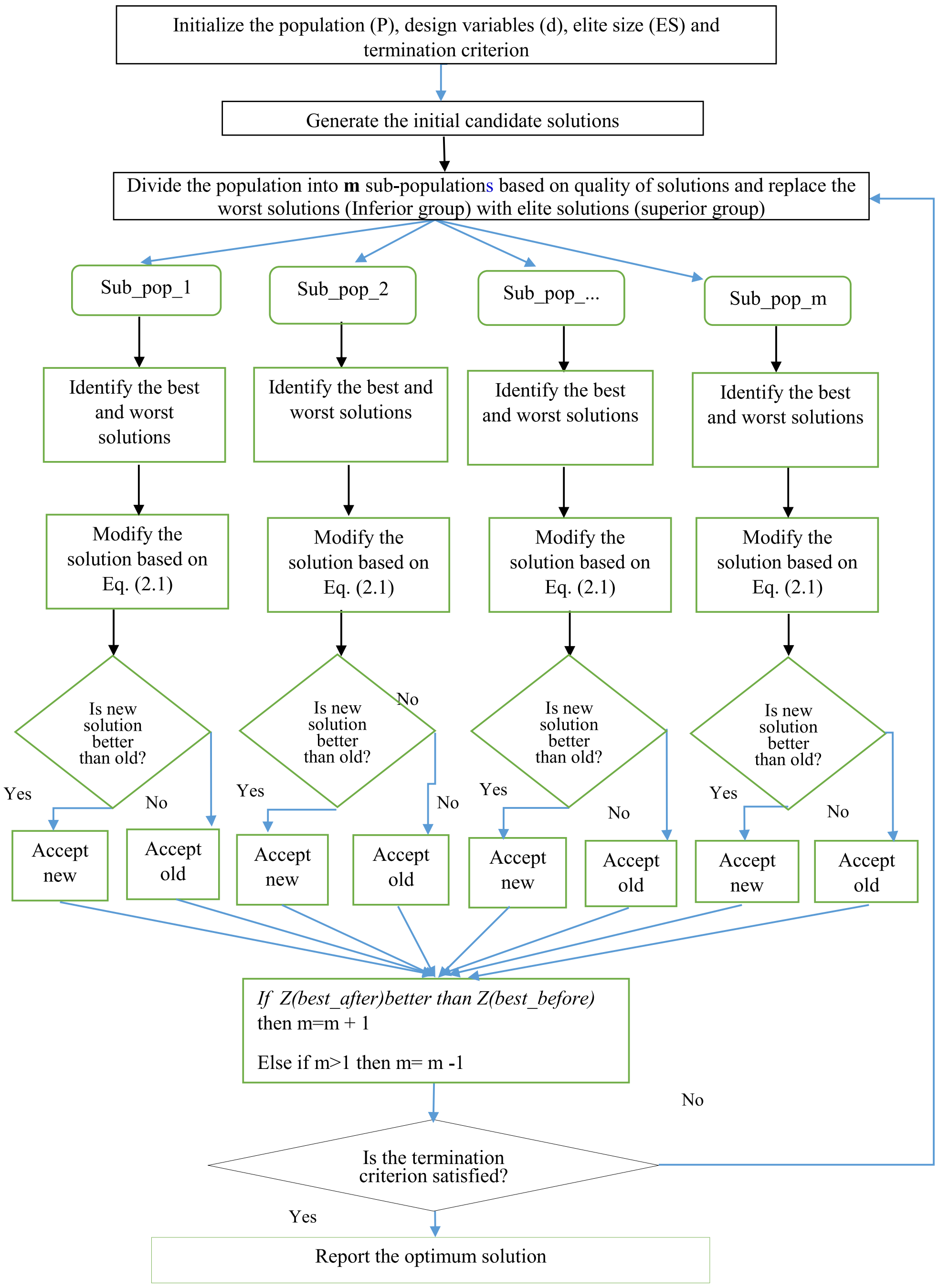
Initialize the population (P), design variables (d), elite size (ES) and termination criterion
Generate the initial candidate solutions
Divide the population into **m** sub-populations based on quality of solutions and replace the worst solutions (Inferior group) with elite solutions (superior group)
Sub_pop_1
Sub_pop_2
Sub_pop_...
Sub_pop_m
Identify the best and worst solutions
Modify the solution based on Eq. (2.1)
Is new solution better than old?
Yes
No
Accept new
Accept old
If Z(best_after)better than Z(best_before) then m=m + 1
Else if m>1 then m= m -1
No
Is the termination criterion satisfied?
Yes
Report the optimum solution

Fig. 1 Flowchart of SAMPE-Jaya algorithm

and learner phase). Equation (2.1) is used for upgrading the solution quality during the search process. For ensuring better exploration of the search space, two random numbers r1 and r2 are used. The absolute value of the candidate solution ($|Xq, r, i|$) in Eq. (2.1) helps the algorithm to further increase the exploration ability. These features make the algorithm to converge toward global optimal solution rather than toward a local optimal solution.

The following equation is used for generating the initial solutions:

$$X\_intial = X\_min + rand(0, 1)^*(X\_max - X\_min) \tag{2.2}$$

where $rand(0, 1)$ is random number between 0 and 1 with normal distribution whose mean is zero and standard deviation is 1. X_min and X_max are the minimum and maximum values of design variables, respectively.

The random numbers between 0 and 1 in Eq. (2.2) are used to distribute the populations into entire search space. Furthermore, r1 and r2 between 0 and 1 are used to explore the entire search space for getting global optimal solution within the specified limit.

Now, if the ranges of the random numbers used in producing the next solutions are changed and if the ranges of random numbers are increased beyond 1, it means we are allowing the algorithm to search beyond the specified limits of the design variables. This may lead to infeasible solutions, and complexity of the search process may be increased. Furthermore, if the ranges of random numbers are reduced (0 to less than l), it means the algorithm is not allowed to explore the entire search space. This may lead to trap the algorithm in local optima.

Furthermore, random numbers with normal distribution are used in metaheuristics for the search process. Normalizing the values of the random numbers in each iteration may not serve the purpose of metaheuristics. It may increase the exploration rate but may have poor exploitation rate. This may lead the algorithm toward local optima with higher computational cost. Furthermore, the elitist version of multi-population Jaya algorithm is purposed in this work.

Now, for investigating the performance of the proposed SAMPE-Jaya algorithm, it is used for solving 30 unconstrained benchmark problems (including 15 problems of CEC 2015), three large-scale problems, 6 constrained benchmark problems and 4 well-known constrained mechanical design optimization problems taken from the literature (Ngo et al. 2016). The next section presents the analysis of the results obtained by the SAMPE-Jaya algorithm and its comparison to the other approaches.

3 Optimization results and discussion

The SAMPE-Jaya algorithm is coded in MATLAB R2009b with a laptop of HP Pavilion g6 Notebook PC of 4 GB RAM memory, 1.9-GHz AMD A8 4500M APU CPU. The performance of the SAMPE-Jaya algorithm is compared with various optimization algorithms: GA and its variants, PSO and its variants, DE, TLBO, etc. The performance of the SAMPE-Jaya algorithm is evaluated on thirty unconstrained problems (including CEC 2015 function), three large-scale problems, six constrained problems and four well-known constrained mechanical design problems considered from the work of Ngo et al. (2016). Furthermore, the proposed SAMPE-Jaya algorithm is used for the design optimization of a micro-channel heat sink (MCHS). The performance analysis of SAMPE-Jaya algorithm on unconstrained benchmark problems is presented in the following section.

3.1 Unconstrained benchmark problems

The performance analysis of the SAMPE-Jaya algorithm is presented in this section on fifteen unconstrained benchmark problems and fifteen computationally expensive unconstrained benchmark problems taken from CEC 2015 (Ngo et al. 2016). The optimum value of function O_8 is -418.9829* dimension, and the values of the global optimum values of rest of the problems are zero.

3.1.1 Analysis of results related to unimodal and multimodal problems

The results obtained by SAMPE-Jaya algorithm for the unimodal and multimodal problems are compared with GA and its variants, PSO, its variants, SAMP-Jaya and Jaya algorithms. Tables 1 and 2 present the comparison of results. The results achieved by SAMPE-Jaya algorithm for functions (O_1 to O_{15}) with 50,000 and 200,000 function evaluations over 30 independent runs for the 30 dimension problems are presented in Tables 1 and 2, respectively. The results are compared with the Jaya algorithm (Rao and Saroj 2017), self-adaptive multi-population (SAMP) Jaya algorithm (Rao and Saroj 2017), extraordinariness particle swarm optimizer (EPSO) (Ngo et al. 2016), adaptive inertia weight PSO (AIWPSO) (Nickabadi et al. 2011), gravitational search algorithm (GSA) (Rashedi et al. 2009), Frankenstein's PSO (F-PSO) (Oca and Stutzle 2009), real coded genetic algorithm (RGA)(Haupt and Haupt 2004), comprehensive learning PSO (LPSO) (Liang and Qin 2006), cooperative PSO (CPSO) (Bergh and Engelbrecht 2004), PSO (Eberhart and Kennedy 1995) and fully informed particle swarm (FIPS)(Mendes et al. 2004).

It can be observed from Table 1 that the mean function value recorded by using the SAMPE-Jaya algorithm is better or equal in 10 cases out of 15 cases as compared to the other algorithms. The SAMPE-Jaya algorithm is able to get the global optimum values of the function $O_6$and O_{14}. Similarly, the values of standard deviation (SD) recorded by using the SAMPE-Jaya algorithm for the same case are better or equal in 10 cases out of 15. For the rest of cases (O_4, O_8, $O_{12}$and O_{13}), performance of the proposed SAMPE-Jaya algorithm is competitive except for the objective O_9 It can be concluded based on these results that the performance of the SAMPE-Jaya algorithm is better as compared to the other algorithms.

Table 2 presents the performance comparison of the SAMPE-Jaya algorithm with the other optimization algorithms for maximum function evaluations of 200,000. It can be observed from Table 2 that the mean function value recorded by using the SAMPE-Jaya algorithm is better or equal in 12 cases out of 15 in comparison with the other approaches. The proposed SAMPE-Jaya algorithm is able to get the global optimum value of the function $O_{3,}$ O_6 and $O_{14.}$ Similarly, the values of SD obtained by the proposed SAMPE-Jaya algorithm for these problems are better or equal in 11 cases out of 15. For the rest the cases, performance of the proposed SAMPE-Jaya algorithm is competitive except for

Table 1 Comparative results for the unimodal and multimodal problems with maximum function evaluations of 50,000

	RGA		GSA		EPSO		Jaya algorithm		SAMP-Jaya algorithm		SAMPE-Jaya algorithm			
	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	*P*	ES
O_1	2.313E+01	1.215E+01	6.800E−17	2.180E−17	7.776E−18	6.010E−18	7.321e−19	2.820e−18	2.799e−19	8.270e−19	**3.951e−22**	**6.517e−22**	15	2
O_2	1.073E+00	2.666E−01	6.060E−08	1.190E−08	6.787E−12	3.008E−12	4.532910503e−11	1.05439e−10	1.343855e−12	2.163e−11	**5.213e−15**	**7.931e−15**	15	2
O_3	5.617E+02	1.256E+02	9.427E+02	2.466E+02	2.121E−01	5.461E−01	5.623442438e−26	1.634578e−25	**1.546e−35**	**4.484e−35**	**1.546e−35**	**4.484e−35**	10	2
O_4	1.178E+01	1.576E+00	4.207E+00	1.122E+00	**9.941E−03**	**9.855E−03**	5.060152	7.352134	1.152804	2.808808	0.106004	0.049688	20	4
O_5	1.180E+03	5.481E+02	4.795E+01	3.956E+00	1.785E−02	2.136E−02	1.752707344e−06	1.75270e−05	1.8006e−09	1.84e−08	**5.273e−11**	**4.699e−09**	15	2
O_6	2.401E+01	1.017E+01	9.310E−01	2.510E+00	**0.00E+00**	**0.00E+00**	**0.00+00**	**0.00E+00**	**0.00E+00**	**0.00E 00**	**0.00E+00**	**0.00E 00**	10	2
O_7	6.750E−02	2.870E−02	7.820E−02	4.100E−02	6.470E−04	4.542E−04	7.4145641e−19	4.052875e−18	3.077026e−28	1.14e−27	**8.639e−30**	**2.716e−29**	15	2
O_8	− 1.248E+04	5.326E+01	− 3.604E+03	5.641E+02	**−1.257E+04**	**3.851E−12**	− 11466.336436	1288.140757	− 11763.9563	838.950	− 12072.030	428.7556	10	2
O_9	5.902E+00	1.171E+00	2.940E+01	4.727E+00	**2.274E−14**	**2.832E−14**	116.24670	58.29225	68.516505	32.5437	49.150865	11.442653	15	2
O_{10}	2.140E+00	4.014E−01	4.800E−09	5.420E−10	1.284E−09	7.280E−10	9.35277331e−11	1.4691025e−10	7.07370e−11	6.71e−11	**9.681e−14**	**1.502e−13**	10	2
O_{11}	1.168E+00	7.950E−02	1.669E+01	4.283E+00	2.533E−08	1.311E−07	2.71E−14	3.16095E−14	1.45E−14	1.94E−14	**2.220E−17**	**4.965E−17**	15	2
O_{12}	5.100E−02	3.520E−02	5.049E−01	4.249E−01	**6.055E−20**	**8.970E−20**	1.6426611e−06	1.03669020e−04	3.258634e−12	6.132e−10	3.2586e−12	6.13285e−10	10	2
O_{13}	8.170E−02	1.074E+01	3.405E+00	3.683E+00	**9.373E−16**	**1.802E−15**	9.13E−11	2.84099E−10	**2.80E−13**	1.19E−12	**2.80E−13**	1.1995E−12	15	2
O_{14}	–	–	–	–	–	–	0.0000E+000	0.00000E+000	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+000**	10	2
O_{15}	–	–	–	–	–	–	2.20E−25	2.74948E−26	8.00E−27	**2.7494E−27**	1.50E−32	**0.00E+000**	10	2

Bold value shows better solution
P population size, *ES* elite size
Source: RGA (Ngo et al. 2016); GSA (Ngo et al. 2016); EPSO (Ngo et al. 2016); Jaya algorithm (Rao and Saroj 2017); SAMP-Jaya (Rao and Saroj 2017)

Table 2 Comparative results for the unimodal and multimodal problems with maximum function evaluations of 200,000

Test function		PSO	CPSO	CLPSO	FFIPS	F-PSO	AIWPSO	EPSO	Jaya Algorithm	SAMP-Jaya	SAMPE-Jaya	
O$_1$	M	5.198E−70	5.146E−13	4.894E−39	4.588E−27	2.409E−16	**33.370E−134**	1.662E−74	7.057e−90	4.939e−89	1.36E−102	P = 15 ES = 2
	SD	1.130E−74	7.759E−25	6.781E−39	1.958E−53	2.005E−31	**5.172E−267**	2.761E−75	3.445e−89	1.894e−88	3.61E−102	
O$_2$	M	2.070E325	1.253E−07	8.868E−24	2.324E−16	1.580E−11	13653E−58	1.903E−47	6.924e−69	**9.50e−96**	**3.43E−81**	P = 15 ES = 2
	SD	1.442E−49	1.179E−14	7.901E−49	1.141E−32	1.030E−22	**73735E−123**	2.152E−47	3.779ed68	4.37e−95	1.02E−79	
O$_3$	M	1.458E+00	1.889E+03	1.922E+02	9.463E+00	1.732E+02	1.9587E−10	2.014E−03	1.165e−132	**0.00E+00**	**0.0000E+00**	P = 10 ES = 2
	SD	1.78E+00	9.911E+06	3.843E+02	2.598E+01	9.158+03	1.201E−19	1.934E−3	2.860e−132	**0.00E+00**	**0.0000E+00**	
O$_4$	M	–	–	–	–	–	–	–	7.16E−05	**6.91E−08**	**6.91E−08**	P = 15 ES = 2
	SD	–	–	–	–	–	–	–	0.0002	**6.91E−08**	**6.91E−08**	
O$_5$	M	2.540E+01	8.265E−01	1.322E+01	2.671E+01	2.816E+01	2.500E+00	2.824E−05	1.765e−17	2.252e−25	**1.24E−26**	P = 20 ES = 2
	SD	5.903E+02	2.345E+00	2.148E+02	2.003E+02	2.313E+02	1.600E+01	3.650E−05	9.6721e−17	6.19346e−25	**3.01E−26**	
O$_6$	M	**0.000E+000**	**0.000E+00**	**0.00E+00**	**0.000E+00**	**0.000E+00**	**0.000E+00**	**0.000E+00**	**0.00E+0**	**0.000E+00**	**0.000E+00**	P = 20 ES = 2
	SD	**0.000E+000**	**0.000E+00**	**0.000E+0**	**0.00E+0**	**0.00E+00**	**0.000E+00**	**0.0000E+00**	**0.00E+00**	**0.000E+00**	**0.000E+00**	
O$_7$	M	1.238E−02	1.076E−02	4.064E−03	3.305E−03	4.169E−03	5.524E−03	2.580E−04	1.0322e−106	2.1803e−111	**4.12E−128**	P = 15 ES = 2
	SD	2.311E−05	2.770E−05	9.618E−07	83668E−07	2.401E−06	1.536E−05	1.871E−04	5.653e−106	1.1942e−110	**1.84E−127**	
O$_8$	M	− 1.100E+04	− 1.213E+04	− 1.255E+04	− 1.105E+04	− 1.122E+04	**−1.257E+04**	**−1.257E+04**	− 11224.9936	− 12276.045	− 12246.045	P = 10 ES = 2
	SD	1.375E+05	3.380E04	4.257E+03	9.442E+05	2.227E+05	**1.141E−25**	2.482E−12	1069.593223	569.593223	377.337923	
O$_9$	M	3.476E+01	3.601E−13	0.0000E+00	5.850E01	7.384E+01	1.658E−01	**0.0000E+0**	76.981618	59.245665	29.848726	P = 15 ES = 2
	SD	1.064E+02	1.540E−24	0.0000E+00	1.919E+02	3.706E+02	2.105E−01	**0.0000E+0**	26.329608	23.896502	11.512196	
O$_{10}$	M	1.492E−14	1.609E−07	9.237E−15	1.386E−14	2.179E−09	6.987E−15	1.214E−14	5.15143e−15	**4.440e−15**	**4.44E−15**	P = 10 ES = 2
	SD	1.863Ed29	7.861E−14	6.616E−30	2.323E−29	1.719E−18	4.207E−31	**3.106E−15**	2.7040e−15	**0.0000E+00**	**0.00E+00**	
O$_{11}$	M	2.162E−02	2.125E−02	0.000E+00	2.478E−04	1.474E−03	2.852E−02	**0.0000E+0**	**0.000E+00**	**0.000E+00**	**0.000E+00**	P = 10 ES = 2
	SD	4.502E−04	6.314E−04	0.000E+00	1.827E−06	1.285E−05	7.664E−04	0.0000E+0	0.000E+00	**0.000E+00**	**0.000E+00**	
O$_{12}$	M	–	–	–	–	–	–	–	5.0968e−12	**1.5705e−18**	**1.50054e−18**	P = 15 ES = 2
	SD	–	–	–	–	–	–	–	5.09684e−09	**6.49477e−11**	**6.5435e−12**	
O$_{13}$	M	–	–	–	–	–	–	–	5.64E−32	**1.54E−33**	**1.54E−33**	P = 15 ES = 2
	SD	–	–	–	–	–	–	–	2.739E−31	**2.2504E−34**	**2.2504E−35**	
O$_{14}$	M	2.096E+01	5.137E−13	5.944E−24	6.188E+01	7.035E+01	1.184E−16	**0.00E+0**	**0.00E+0**	**0.00E+0**	**0.00E+0**	P = 10 ES = 2
	SD	1.8333E+02	5.944E−24	1.333E−01	1.401E+02	2.960E+02	4.207E−31	**0.000E+0**	**0.00E+0**	**0.00E+0**	**0.00E+0**	
O$_{15}$	M	1.142E−29	2.091E−15	1.295E−29	1.027E−28	5.514E−18	**1.500E−32**	**1.500E−32**	**1.500E−32**	**1.500E−32**	**1.50E−32**	P = 15 ES = 2
	SD	3.233E−57	1.295E−29	1.500E−32	1.005E−56	1.450E−34	**1.240E−94**	1.113E−47	2.7837E−45	4.658e−48	3.780E−46	

Bold value shows better solution

P population size, *ES* elite size

Source: PSO (Ngo et al. 2016); CPSO (Ngo et al. 2016); CLPSO (Ngo et al. 2016); FFIPS (Ngo et al. 2016); F-PSO (Ngo et al. 2016); EPSO (Ngo et al. 2016); Jaya Algorithm (Rao and Saroj 2017); SAMP-Jaya (Rao and Saroj 2017) the objective O_9. It can be concluded based on these results that the performance of the SAMPE-Jaya algorithm is better as compared to the other algorithms.

3.1.2 Analysis of the results related to CEC computationally expensive benchmark problems

The benchmark problems considered in this section are from CEC 2015. The objective of all functions is minimization. Problems 1–9 are shifted and rotated problems; problems 10–12 are hybrid functions; and problems 13–15 are composite functions. The detailed information about CEC 2015 problems can be retrieved from the literature (Ngo et al. 2016).

The computational experiments are carried out by following the guidelines of CEC 2015. The maximum function evaluations (MFEs) of 500 and 1500 are considered as one of the termination criteria with ten dimension and thirty dimension problems, respectively. Second stopping criterion is, while the error value (current optimum value–global optimum value) is less than 1.00E−03. Average of the minimum error value is recorded over 20 independent runs. The averaged minimum values are used for the performance comparison of the proposed SAMPE-Jaya algorithm with the other algorithms.

The computational results obtained by proposed SAMPE-Jaya algorithm are compared with EPSO, DE, $(\mu + \lambda)$-evolutionary strategy (ES), specialized and generalized parameters experiments of covariance matrix adaption evolution strategy (CMAES-S and CMAES-G) (Andersson et al. 2015). Table 3 presents the comparison of the computational results. It can be observed from Table 3 that the results obtained by using the SAMPE-Jaya algorithm are better in 12 cases for 10-D and 9 cases for 30-D problems. An observation can be made from this table that the results achieved by the proposed SAMPE-Jaya algorithm are better or competitive in comparison with other algorithms used for this problem. The bold values in Table 3 show the minimum mean error values obtained by different algorithms for each function.

The computational complexity of the SAMPE-Jaya algorithm is evaluated for 30-D and 10-D problems as per the guidelines of the CEC 2015. The computational complexity of the algorithms is defined as the complexity in the computational time when the dimensions of the problem are increased. The test function provided by CEC 2015 is run on the same computer which is used for optimization of problems of CEC 2015. The time (T_0) required for the complete execution of this function was found to be 0.3447 s. Next step is to record the average processing time (T_1) for each problem. The computational complexity of the of the algorithm (T_1/T_o) is calculated and is shown in Table 4. In Table 4, the values T_B/T_A disclose the complexity of the algorithm when the dimension of the problems changes. The value of T_B/T_A equals to 1 means that it is not any having complexity when problem changes from 10-D to 30-D.

The values T_B/T_A more than one disclose complexity of computational time of the algorithms. Functions FCEC3 and FCEC5 are having the higher complexity in terms of computational time because of multi-modality of the functions. Similarly, hybrid functions FCEC11 and FCEC12 and composite functions FCEC13–FCEC15 have shown the higher complexity in terms of computational time. For the remaining problems, the value of T_B/T_A is almost equal to 3.5 which reveal that the computational complexity of the present algorithm is increased 3.5 times when the dimensions of the problems are changed from 10 to 30. The computational complexity of the SAMPE-Jaya algorithm is about 3 for the problems FCEC1, FCEC2, FCEC4, FCEC6, FCEC7, FCEC8, FCEC9, FCEC10 FCEC13 and FCEC14. It shows that computational complexity of the SAMPE-Jaya algorithm is increased about three times when the dimension of the problems changes from 10 to 30.

The computational complexity of SAMPE-Jaya algorithm is more than 4 for the problems FCEC3, FCEC5, FCEC11, FCEC12 and FCEC15. This increment in the computational complexity is due to the complexity and multi-modality of these problems. However, the computational complexity of the SAMPE-Jaya algorithm is less as compared to SAMP-Jaya and EPSO algorithms. As the computational complexity of the other algorithms for CEC 2015 problems is not available (except EPSO) in the literature, the computational complexity of the proposed SAMPE-Jaya algorithm cannot be compared with others. It can also be observed from Table 4 that the computational complexity (T_B/T_A) of the present approach is less in comparison with EPSO for all the problems of CEC 2015.

It can be observed from Tables 1, 2 and 3 that the performance of the SAMPE-Jaya algorithm is better or competitive as compared to the other algorithms. However, it becomes necessary to prove the significance of the proposed SAMPE-Jaya algorithm over the other algorithms with some statistical test. Therefore, a well-known statistical method known as 'Friedman test' (Joaquin et al. 2016) is used to compare the performance of the proposed SAMPE-Jaya algorithm with the other algorithms. Mean values of the fitness functions obtained by different methods are considered for this test. This method first finds the rank of algorithms for the individual problems and then calculates the average rank to get the final rank of the each algorithm for the considered problems. This test was performed with assuming χ^2 distribution and with $k - 1$ degree of freedom.

The mean ranks of the algorithms for the unimodal and multimodal problems with maximum function evaluations of 50,000 are presented in Table 5. It can be observed from this table that the performance of the proposed SAMPE-Jaya algorithm is better than the other methods. It has obtained

Table 3 Comparative results for mean error obtained by different approaches for CEC 2015 problems

Function	Dimension	DE (Ngo et al. 2016)	($\mu + \lambda$)ES (Ngo et al. 2016)	CMAES-S (Andersson et al. 2015)	CMAES-G (Andersson et al. 2015)	EPSO (Ngo et al. 2016)	Jaya algorithm (Rao and Saroj 2017)	SAMP-Jaya (Rao and Saroj 2017)	SAMPE-Jaya	*P*	ES
FCEC1	10	3.4143E+09	2.6325E+09	3.6620E+07	6.5990E+07	1.5785E+09	1.3175E+07	6.40376E+06	**3.0377E+06**	10	2
	30	2.3911E+10	3.5775E+10	**6.8700E+07**	1.1080E+08	8.4866E+09	7.1308E+08	3.02517E+08	6.2057E+08	15	2
FCEC2	10	7.4931E+04	4.8418E+04	5.8080E+04	1.0240E+05	1.8953E+04	5.4161E+03	**1.9699E+03**	2.5711E+03	15	2
	30	1.8254E+05	1.6179E+05	2.3630E+05	2.9530E+05	6.3748E+04	1.3021E+04	**1.1894E+03**	**1.1894E+03**	20	2
FCEC3	10	3.1093E+02	3.1048E+02	6.1200E+02	6.1570E+02	3.1009E+02	3.0566E+02	3.0021E+02	**3.0005E+02**	10	2
	30	3.4190E+02	3.4353E+02	6.3390E+02	6.5270E+02	3.3800E+02	3.0231E+02	3.0114E+02	**3.0008E+02**	15	2
FCEC4	10	2.2974E+03	1.4368E+03	3.1890E+03	4.1090E+03	2.0662E+03	4.6250E+02	**4.2458E+02**	**4.2458E+02**	10	2
	30	7.9627E+03	7.0557E+03	8.6730E+03	1.2040E+04	6.6946E+03	8.2815E+02	**7.77849E+02**	**7.77849E+02**	20	4
FCEC5	10	5.0286E+02	5.0318E+02	1.0010E+03	1.0060E+03	5.0305E+02	5.0001E+02	**5.00007E+02**	**5.00007E+02**	15	2
	30	5.0431E+02	5.0499E+02	1.0010E+03	1.0080E+03	5.0430E+02	**5.0000E+02**	**5.00000E+02**	**5.00000E+02**	15	2
FCEC6	10	6.0286E+02	6.0223E+02	1.2010E+03	1.2010E+03	6.0234E+02	6.0528E+02	**6.01758E+02**	**6.01758E+02**	10	2
	30	6.0365E+02	6.0433E+02	1.2010E+03	1.2010E+03	6.0276E+02	**6.0209E+02**	6.07713E+02	6.04121E+02	20	4
FCEC7	10	7.2588E+02	7.1668E+02	1.4010E+03	1.4020E+03	7.1725E+02	734.419551	706.745007	**705.0833**	15	2
	30	7.5438E+02	7.8216E+02	1.4010E+03	1.4010E+03	**7.2189E+02**	1.2025E+03	1062.726345	1.03644E+03	10	2
FCEC8	10	4.1637E+03	1.1691E+03	**1.6130E+03**	1.6480E+03	1.6412E+03	2.109E+03	1.760E+03	1.760E+03	15	2
	30	7.9963E+05	7.3789E+06	**1.7670E+03**	2.3210E+03	1.2746E+05	1.400E+05	7.109E+05	7.109E+05	15	2
FCEC9	10	9.0415E+02	9.0414E+02	1.8080E+03	1.8080E+03	9.0407E+02	903.426016	903.105930	**903.02836**	15	2
	30	9.1394E+02	9.1408E+02	1.8270E+03	1.8280E+03	9.1372E+0**2**	913.311482	**913.281842**	**912.43551**	15	2
FCEC10	10	1.3622E+06	1.4666E+06	1.7440E+05	1.7700E+06	1.3052E+06	43885.7623	22782.9766	**11669.4733**	20	4
	30	3.8759E+07	9.5323E+07	3.6310E+06	1.4730E+07	2.6363E+07	31356.0475	*85,248.5215*	**29088.6247**	15	2
FCEC11	10	1.1229E+03	1.1150E+03	2.2120E+03	2.2190E+03	**1.1140E+03**	1119.28310	1118.847733	1118.847733	10	2
	30	1.2870E+03	1.4378E+03	2.2460E+03	2.2580E+03	1.2288E+03	1165.52400	1164.500690	**1163.230763**	15	2
FCEC12	10	1.5980E+03	1.5797E+03	2.7390E+03	2.9810E+03	1.5291E+03	1423.57853	1352.880566	**1213.143563**	10	2
	30	3.0110E+03	3.8087E+03	3.4540E+03	4.0940E+03	2.4432E+03	1490.04466	1284.020952	**1211.077290**	15	2
FCEC13	10	1.7969E+03	**1.6663E+03**	3.2580E+03	3.3000E+03	1.6932E+03	1663.94774	1663.187051	**1.662289769**	10	2
	30	1.9613E+03	2.2208E+03	3.3840E+03	3.4260E+03	1.8839E+03	1716.39830	1700.033267	**1688.291320**	20	2
FCEC14	10	1.6135E+03	1.6148E+03	3.2090E+03	3.2170E+03	1.6064E+03	*1446.11153*	***1426.164181***	***1426.164181***	25	4
	30	1.7479E+03	1.8406E+03	3.2660E+03	3.3000E+03	1.7016E+03	***1675.18244***	1699.661232	1699.661232	10	2
FCEC15	10	1.9452E+03	1.9740E+03	3.7770E+03	3.9020E+03	**1.8662E+03**	1863.52977	**1862.935212**	**1862.170694**	15	2
	30	2.9304E+03	2.9154E+03	4.4270E+03	4.8360E+03	2.7488E+03	1932.59722	1936.961342	1919.523203	10	2

Bold value shows better solution
P population size, *ES* elite size

Table 4 Computational complexity of the SAMP-Jaya algorithm

Function	$d = 10$		$d = 30$		Computational complexity		
	T_1 (s)	$T_A = T_1/T_0$	T_1 (s)	$T_B = T_1/T_0$	SAMPE-Jaya (T_B/T_A)	SAMP-Jaya (Rao and Saroj 2017)	EPSO (Ngo et al. 2016)
1.	0.039641	0.115003771	0.11094	0.321844116	2.798553	3.588393	3.605
2.	0.030814	0.089393772	0.102472	0.297278503	3.325495	3.413454	3.432
3.	0.152056	0.441127647	1.053516	3.056328595	6.928445	7.429281	7.968
4.	0.058882	0.170823325	0.14269	0.413952713	2.42328	3.547105	3.591
5.	0.248303	0.720346872	1.373376	3.984265835	5.531038	7.271586	7.511
6.	0.038857	0.112728653	0.120896	0.350727686	3.111256	3.351844	3.385
7.	0.031087	0.090187603	0.112622	0.326725172	3.622728	3.400896	3.444
8.	0.033716	0.097813654	0.129546	0.375821971	3.842224	3.443678	3.592
9.	0.029994	0.087015182	0.108133	0.313702253	3.605144	3.52979	3.692
10.	0.034821	0.101020404	0.128561	0.37296509	3.691978	3.500343	3.679
11.	0.060534	0.175613577	0.250034	0.725366212	4.130468	4.04166	6.158
12.	0.055663	0.161483706	0.254652	0.738763659	4.57485	4.676693	4.785
13.	0.071664	0.207903684	0.208266	0.604194179	2.906125	3.327989	5.702
14.	0.072664	0.210806112	0.263034	0.763081714	3.619827	3.406342	5.270
15.	0.188662	0.547323276	1.225011	3.55384663	6.49314	6.325926	8.036

higher rank as compared to the other algorithms used for this problem. Comparison of the same on the bar chart is presented in Fig. 2. The mean rank of the algorithms for the unimodal and multimodal problems with maximum function evaluations of 200,000 is presented in Table 6. It can be observed from this table that performance of the proposed SAMPE-Jaya algorithm is better than the other algorithms used for the optimization of the same problem. The proposed SAMPE-Jaya algorithm is having higher rank in this case also as compared to the other algorithms. Figure 3 presents the comparison of the ranks on the bar chart.

The comparison of performance of the proposed SAMPE-Jaya algorithm for CEC 2015 problems is shown in Tables 7 and 8 for 10-D and 30-D problems, respectively. The average rank obtained by the SAMPE-Jaya algorithm for the 10-D problems is 1.6667, which is better than the other algorithms. Similarly, average rank obtained by the SAMPE-Jaya algorithm for the 30-D problems is 2.2, which is better than the rest of algorithms. Figures 4 and 5 present the comparison of the average ranks of algorithm on bar charts for the 10-D and 30-D problems, respectively.

Thus, the Friedman rank test confirms that the performance of the proposed SAMPE-Jaya algorithm is better for the considered unconstrained benchmark problems and it has obtained the highest rank as compared to the other algorithms used for the optimization all these cases considered for the comparison.

3.2 Analysis of results related to constrained benchmark problems

This section presents comparison of performance of the proposed SAMPE-Jaya algorithm on six constrained benchmark optimization problems considered from literature (Ngo et al. 2016). In this study, the proposed SAMPE-Jaya algorithm is run 30 times for each case and performance comparison of the proposed SAMPE-Jaya algorithm with other algorithm is carried out based on the best function value in the entire run (*Best*), worst function value (*Worst*), mean of the best function value in entire run (*mean*), deviation of the best solution from the mean best solution (*SD*) and mean function evaluations (MFEs) required for the computation. In order to deal with the constraints imposed on these objectives, a static penalty function is used for penalizing the fitness value of the objective function.

The results obtained by the SAMPE-Jaya algorithm are compared with the other optimization algorithms, and these are: EPSO (Ngo et al. 2016), hybrid real-binary PSO (HPSO) (Jin and Rahmat-Samii 2010), α constraint simplex method (α simplex) (Takahama and Sakai 2005), improved stochastic ranking (ISR) (Runarsson and Xin 2005), GA hybrid Nelder–Mead simplex search and PSO (NM-PSO) (Zahara and Kao 2009), hybrid evolutionary algorithm and adaptive constraint handling technique (HEAA) (Wang et al. 2009), artificial bee colony (ABC) (Karaboga and Basturk 2007), cultural algorithms with evolutionary programming (CAEP) (Coello and Becerra 2004), differential evolution with dynamic stochastic selection (DEDS) (Zhang et al.

Table 5 Friedman rank test for unimodal and multimodal problems with 50,000 function evaluations

Problem	50,000 function evaluations					
Algorithm	RGA	GSA	EPSO	Jaya	SAMP-Jaya	SAMPE-Jaya
Friedman ranks	4.8333	4.9	3.0333	3.7	2.6667	1.8667
p value	4.16E−06					
χ^2	32.7822					

Fig. 2 Friedman rank test for unimodal and multimodal problems with 50,000 function evaluations

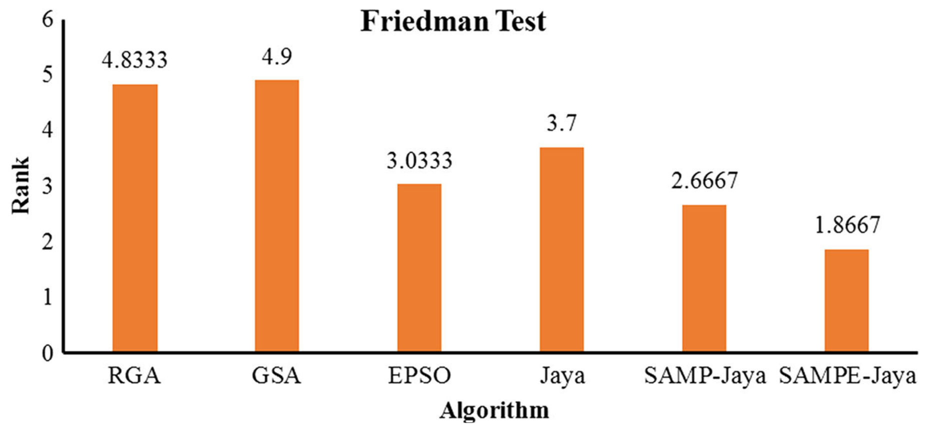

Table 6 Friedman rank test for unimodal and multimodal problems with 200,000 function evaluations

Problem	200,000 function evaluations									
Algorithm	PSO	CPSO	CLPSO	FFIPS	F-PSO	AIWOP	EPSO	Jaya	SAMP-Jaya	SAMPE-Jaya
Friedman ranks	7.3	7.6333	5.8	7.3	8.1	5.2	4.4667	3.866	2.9	2.4333
p value	3.13E−12									
χ^2	73.4921									

Fig. 3 Friedman rank test for unimodal and multimodal problems with 200,000 function evaluations

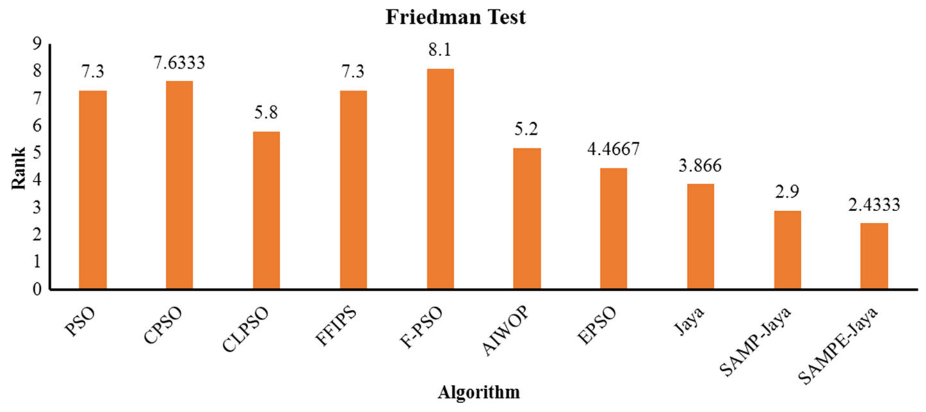

Table 7 Friedman rank test for CEC 2015 problems with 10 dimensions

Algorithm	DE	$(\mu + \lambda)$-ES	CMAES-S	CAMES-G	EPSO	Jaya	SAMP-Jaya	SAMPE-Jaya
Friedman ranks	5.9333	4.6	6.2667	7.4667	4.1333	3.8	2.1333	1.6667
p value	9.06E−13							
χ^2	71.0511							

Table 8 Friedman rank test for CEC 2015 problems with 30 dimensions

Algorithm	DE	$(\mu + \lambda)$-ES	CMAES-S	CAMES-G	EPSO	Jaya	SAMP-Jaya	SAMPE-Jaya
Friedman ranks	5.3333	6.0667	6	6.9333	3.9333	2.9333	2.6	2.2
p value	3.87E−10							
χ^2	57.9488							

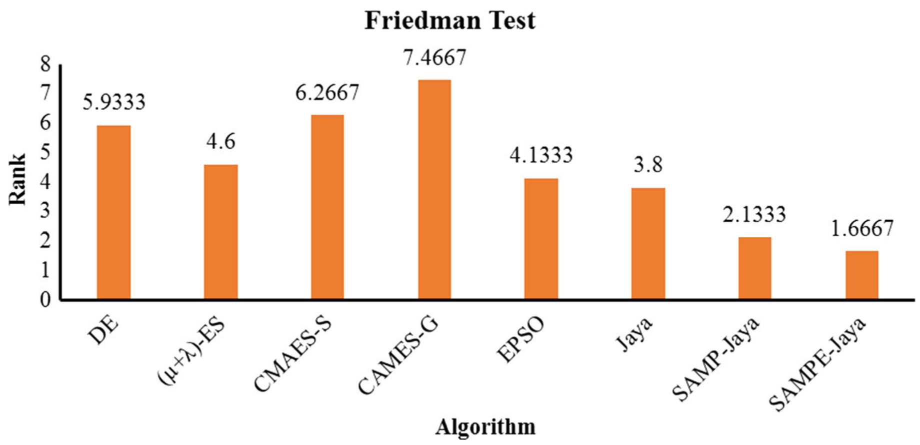

Fig. 4 Friedman rank test for CEC 2015 problems with 10 dimensions

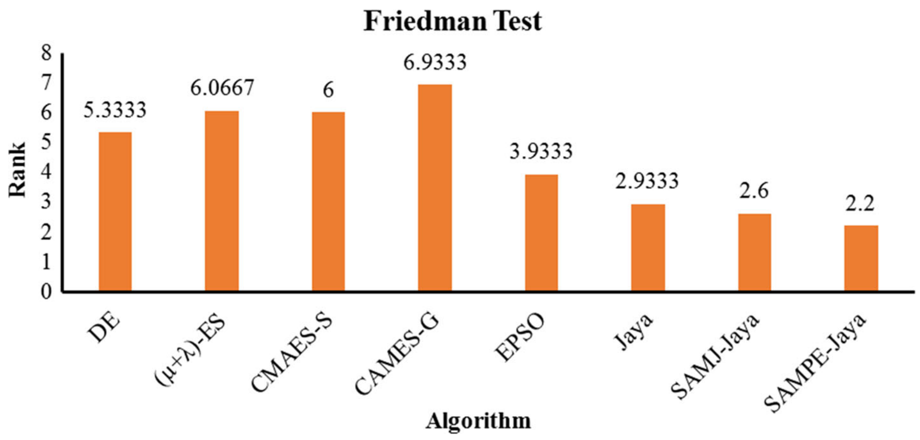

Fig. 5 Friedman rank test for CEC 2015 problems with 30 dimensions

2008), stochastic ranking (SR) (Runarsson and Xin 2000), differential evolution (DE) (Lampinen 2002), particle swarm optimization with differential evolution (PSO-DE) (Liu et al. 2010), simple multi-membered evolution strategy (SMES) (Mezura-Montes and Coello 2006), cultured differential evolution (CULDE) (Becerra and Coello 2006), changing range genetic algorithm (CRGA) (Amirjanov 2006), self-adaptive penalty function (SAPF) (Tessema and Yen 2006), adaptive segregational constraint handling evolutionary algorithm (ASCHEA) (Hamida and Schoenauer 2002), homomorphous mappings (HM) (Koziel and Michalewicz 1999), chaotic gray wolf optimization (CGWO) algorithm (Kohli and Arora 2017), gravitational search algorithm (GSA), particle swarm optimization (PSO), whale optimization algorithm (WOA) (Mirjalili and Lewis 2016), thermal exchange optimization algorithm (TEOA) (Kaveh and Dadras 2017), SAMP-Jaya and Jaya algorithms.

The comparison of results for the problem 1 is shown in Table 9. It can be observed from this table that the best value found by the SAMPE-Jaya algorithm is better or competitive in comparison with the other algorithms. Similarly, the value of mean and worst function value obtained by the SAMPE-Jaya algorithm are better as compared to the other algorithms. The proposed SAMPE-Jaya algorithm requires 96.52, 83.77, 91.13, 97.56, 94.92, 82.64, 86.84, 82.64, 59.43, 99.13, 94.59, 93.91, 87.84, 96.08, 99.18, 75.66, 7.64 and 20.26% fewer function evaluation in comparison with SR, SMES, ISR, SAPF, ABC, CRGA, PSO, DE, HM, DEDS, HEAA, CULDE, α-simplex, ASCHEA, EPSO, SAME-Jaya and Jaya algorithms, respectively.

Table 10 presents the comparison of the results obtained by SAMPE-Jaya algorithm for problem 2 with different algorithms. It can be observed from this table that the best function value achieved by the SAMPE-Jaya algorithm and other

Table 9 Comparison of the statistical results for constrained problem 1

Method	Best	Worst	Average	SD	MFEs
SR	0.7500	0.7500	0.7500	8.00E−05	350,000
SMES	0.7500	0.7500	0.7500	1.52E−04	75,000
ISR	0.7500	0.7500	0.7500	**1.10E−16**	137,200
SAPF	0.7490	0.7570	0.7510	2.00E−03	500,000
ABC	0.7500	0.7500	0.7500	0.00E+00	240,000
PSO-DE	0.7500	0.7500	0.7500	2.50E−07	70,100
CRGA	0.7500	0.7570	0.7520	2.50E−03	3000
PSO	0.7500	0.9988	0.8605	8.40E−02	70,100
DE	0.7490	0.7490	0.7490	N/A	30,000
HM	0.7500	0.7500	0.7500	N/A	1400,000
DEDS	0.7499	0.7499	0.7499	0.00E+00	225,000
HEAA	0.7500	0.7500	0.7500	3.40E−16	200,000
CULDE	0.7499	0.7964	0.7580	1.71E−02	100,100
α-simplex	0.7499	0.7499	0.7499	4.90E−16	308,125
ASCHEA	0.7500	N/A	0.7500	N/A	1500,000
EPSO	0.7500	0.7508	0.7501	1.62E−04	50,000
Jaya algorithm	**0.74750**	0.747515	0.747502	0.000005	15261.333
SAMP-Jaya	**0.74750**	0.747513	0.747501	0.000003	13176.333
SAMPE-Jaya (for $P = 15$ and $\mathrm{ES} = 2$)	**0.74750**	**0.747501**	**0.747500**	0.0000002	**12168.200**

Source: The results of this table except the results of Jay, SAMP-Jaya and SAMPE-Jaya algorithm are taken from Ngo et al. (2016)
P population size, ES elite size

Table 10 Comparison of the statistical result for constrained problem 2

Method	Best	Worst	Average	SD	MFEs
ISR	−0.0958	−0.0958	−0.0958	2.70E−17	160,000
DEDS	−0.0958	−0.0958	−0.0958	4.00E−17	225,000
ASCHEA	−0.0958	N/A	−0.0958	N/A	1500,000
GA	−0.0958	−0.0958	−0.0958	2.70E−09	4486
PSO-DE	−0.0958	−0.0958	−0.0958	1.30E−12	10,600
NM-PSO	−0.0958	−0.0958	−0.0958	3.50E−08	**2103**
CAEP	−0.0958	−0.0958	−0.0958	0.00E+00	50,020
HPSO	−0.0958	−0.0958	−0.0958	1.20E−10	81,000
DE	−0.0958	−0.0958	−0.0958	N/A	10,000
CRGA	−0.0958	−0.0958	−0.0958	4.40E−06	64,900
SR	−0.0958	−0.0958	−0.0958	2.60E−17	76,200
SAPF	−0.0958	−0.0927	−0.0956	1.06E−03	500,000
SMES	−0.0958	−0.0958	−0.0958	0.00E+00	240,000
CULDE	−0.0958	−0.0958	−0.0958	1.00E−07	100,100
PSO	−0.0958	−0.0291	−0.0945	9.40E−03	10,600
HEAA	−0.0958	−0.0958	−0.0958	2.80E−17	200,000
HM	−0.0958	−0.0291	−0.0892	N/A	1400,000
α-simplex	−0.0958	−0.0958	−0.0958	3.80E−13	306,248
ABC	−0.0958	−0.0958	−0.0958	0.00E+00	240,000
EPSO	−0.0958	−0.0958	−0.0958	1.26E−17	5000
Jaya algorithm	−0.0958	−0.0958	−0.0958	1.288e−17	3,510.0
SAMP-Jaya	−0.0958	−0.0958	−0.0958	**1.062e−17**	2888.0
SAMPE-Jaya (for $P = 10$ and $\mathrm{ES} = 2$)	−0.0958	−0.0958	−0.0958	1.062e−20	**2428.0**

Source: The results of this table except the results of Jaya and SAMP-Jaya algorithm are taken from Ngo et al. (2016)
P population size, ES elite size

algorithms are same. The values of *worst*, *average* and*SD* achieved by using the proposed SAMPE-Jaya algorithm are better or competitive in comparison with the rest of the algorithms. The value of MFEs required by the SAMPE-Jaya algorithm is 98.48, 98.92, 99.83, 98.26, 97.57, 83.81, 34.37, 97.94, 98.26, 98.98, 98.98, 98.56, 98.88, 87.96, 99.17, 28.12 and 33.16% less as compared to ISR, DEDS, ASCHEA, GA, PSO-DE, NM-PSO, CAEP, HPSO, DE, CRGA, SR, SAPF, SMES, CULDE, PSO, HEAA, α-simplex, ABC, EPSO, SAMP-Jaya and Jaya algorithms, respectively.

Comparison of the statistical result for problem 3 is shown in Table 11. It can be observed from this table that the best function value achieved by the SAMPE-Jaya algorithm is same or better as compared to the rest of the algorithms. The values of *worst mean* and *SD* obtained by the SAMPE-Jaya algorithm are better or competitive in comparison with the rest of the algorithms. The value MFEs required by the SAMPE-Jaya algorithm is 99.78, 99.77, 98.57, 97.71, 96.80, 78.67, 13.54, 97.28, 97.71, 98.66, 98.66, 98.66, 98.10, 99.36, 98.90, 5.32 and 12.63% less as compared to ASCHEA, HM, DEDS, PSO-DE, CULDE, DE, CRGA, SR, PSO, ABC, SMES, ISR, SAPF, HEAA, EPSO, α-simplex, EPSO, SAMP-Jaya and Jaya algorithms, respectively.

Comparison of the statistical result for problem 4 is presented in Table 12. It can be observed from this table that the best function value obtained by the SAMPE-Jaya algorithm is same or better in comparison with the other algorithms. The values of *worst, mean* and*SD* obtained by the proposed SAMPE-Jaya algorithm are also better or competitive in comparison with the other algorithms. The value of MFEs required by the SAMPE-Jaya algorithm is 98.05, 97.91, 91.66, 91.66, 99.08, 81.73, 87.84, 41.64, 94.14, 91.66, 85.41, 87.84, 70.85, 87.03, 89.24, 90.97, 91.66, 88.23, 87.84, 76.65, 3.95 and 4.83% less as compared to ASCHEA, HM, GA2, GA1, GA, HS, SMES, CRGA, SAPF, SR, HEAA, DE, CULDE, PSO, DEDS, ISR, α-simplex, PESO, CoDE, ABC, EPSO, SAMP-Jaya and Jaya algorithms, respectively.

The comparison of the statistical result for problem 5 is presented in Table 13. It can be observed from this table that

Table 11 Comparison of the statistical result for constrained problem 3

Method	Best	Worst	Average	SD	MFEs
ASCHEA	−6961.810	N/A	−6961.810	N/A	1500,000
HM	−6952.100	−5473.900	−6342.600	N/A	1400,000
DEDS	−6961.814	−6961.814	−6961.814	0.00E+00	225,000
PSO-DE	−6961.814	−6961.814	−6961.814	2.30E−09	140,100
CULDE	−6961.814	−6961.814	−6961.814	1.00E−07	100,100
DE	−6961.814	−6961.814	−6961.814	N/A	15,000
CRGA	−6956.251	−6077.123	−6740.288	2.70E+02	3700
SR	−6961.814	−6350.262	−6875.940	160E+00	118,000
PSO	−6961.814	−6961.814	−6961.814	6.50E−06	140,100
2ABC	−6961.814	−6961.805	−6961.813	2.00E−03	240,000
SMES	−6961.814	−6962.482	− 6961.284	1.85E+00	240,000
ISR	−6961.814	−6961.814	−6961.814	**1.90E−12**	168,800
SAPF	−6961.046	−6943.304	−6953.061	5.87E+00	500,000
HEAA	−6961.814	−6961.814	−6961.814	4.60E−12	200,000
EPSO	−6961.814	−6961.811	−6961.813	6.20E−04	20,000
α-simplex	−6961.814	−6961.814	−6961.814	1.30E−10	293,367
CGWO	–	–	−6493.18	–	–
Jaya algorithm	−6961.811	−6961.811	−6961.811	1.469e−11	3661.166
SAMP-Jaya	−6961.814	−6961.814	−6961.814	1.068e−11	3378.666
SAMPE-Jaya (for $P = 10$ and ES = 2)	−6961.814	−6961.814	−6961.814	1.068e−14	**3198.666**

Source: The results of this table except the results of Jaya and SAMP-Jaya algorithm are taken from Ngo et al. (2016)
P population size, ES elite size

Table 12 Comparison of the statistical result for constrained problem 4

Method	Best	Worst	Average	SD	MFEs
IGA	680.630	680.630	680.630	1.00E−05	N/A
ASCHEA	680.630	–	680.641	–	1,500,000
HM	680.910	683.180	681.160	4.11E−02	1,400,000
GA2	680.642	–	–	–	350,070
GA1	680.634	680.651	680.642	N/A	350,070
GA	680.630	680.654	680.638	6.61E−03	320,000
HS	680.641	–	–	–	160,000
SMES	680.632	680.719	680.643	1.55E−02	240,000
CRGA	680.726	682.965	681.347	5.70E−01	50,000
SAPF	680.773	682.081	681.246	3.22E−01	500,000
SR	680.630	680.763	680.656	3.40E−02	350,000
HEAA	680.630	680.630	680.630	5.80E−13	200,000
DE	680.771	680.144	680.503	6.70E−01	240,000
CULDE	680.630	680.630	680.630	1.00E−07	100,100
PSO	680.635	684.529	680.971	5.10E−01	140,100
CPSO-GD	680.678	681.371	680.781	1.48E−02	N/A
DEDS	680.630	680.630	680.630	2.90E−13	225,000
ISR	680.630	680.630	680.630	3.20E−13	271,200
α-simplex	680.630	680.630	680.630	2.90E−10	323,426
PESO	680.631	680.630	680.630	–	350,000
CoDE	680.771	685.144	681.503	–	248,000
ABC	680.634	680.638	680.64	4.00E−03	240,000
EPSO	680.637	680.673	680.649	8.14E−03	125,000
CGWO	–	–	7046.13	–	–
Jaya algorithm	680.673	680.673	680.673	1.4692e−11	30,661
SAMP-Jaya	680.673	680.673	680.673	1.0689e−11	30,378
SAMPE-Jaya (for $P = 15$ and ES = 2)	680.673	680.673	680.673	**1.0689e−14**	**29,178**

Source: The results of this table except the results of Jaya and SAMP-Jaya algorithm are taken from Ngo et al. (2016)
P population size, ES elite size

the best function value recorded by using the SAMPE-Jaya algorithm is better as compared to the other algorithms. The values of *worst mean* and*SD* obtained by the SAMPE-Jaya algorithm are also better or competitive in comparison with the rest of the algorithms. The value of MFEs required by the proposed SAMPE-Jaya algorithm is 99.19, 99.25, 87.24, 77.51, 83.95, 86.11, 83.95, 88.86, 95.31, 82.69, 79.32, 97.75, 95.31, 95.31,77.50, 95.00, 94.37, 94.14, 96.31, 77.50, 4.25 and 68.80% less as compared to HM, ASCHEA, SR, CAEP, PSO, HPSO, PSO-DE, CULDE, DE, HS, CRGA, SAPF, SMES, ABC, DELC, DEDS, HEAA, ISR, α-simplex, EPSO, SAMP-Jaya and Jaya algorithms, respectively.

The comparison of the statistical results for problem 6 is shown in Table 14. It can be observed from this table that the best function value achieved by using the SAMPE-Jaya algorithm is better competitive in comparison with the other algorithms. The values of *worst* and *mean* obtained by the proposed SAMPE-Jaya algorithm are also better or competitive as compared to the rest of the algorithms. The value of SD obtained by the proposed SAMPE-Jaya algorithm is minimum as compared to the other algorithms. The value MFEs required by the proposed SAMPE-Jaya algorithm is 98.52, 77.57, 90.80, 90.80, 84.25, 90.80, 95.58, 93.10, 93.68, 90.36, 88.96, 88.96, 90.19, 99.72, 92.90, 93.69, 84.25, 77.93, 98.42, 4.74 and 19.11% less as compared to ASCHEA, CULDE, ABC, PSO-DE, SMES, SAPF, GA, ISR, SR, HEAA, DELC, DEDS, DE, α-simplex, PESO, ABC, EPSO, SAMP-Jaya and Jaya algorithms, respectively.

It can be observed from results of Tables 9, 10, 11, 12, 13 and 14 that the proposed SAMPE-Jaya algorithm has per-

Table 13 Comparison of the statistical result for constrained problem 5

Method	Best	Worst	Mean	SD	MFEs
HM	−30664.500	−30,645.900	−30,665.300	N/A	1,400,000
ASCHEA	−30,665.500	–	– 30,665.500	–	1,500,000
SR	−30,665.539	−30,665.539	−30,665.539	2.00E−05	88,200
CAEP	−30,665.500	−30,662.200	−30,662.500	9.30E+00	50,020
PSO	−30,663.856	−30,252.325	−30,570.9286	8.10E+01	70,100
HPSO	−30,665.539	−30,665.539	−30,665.539	1.70E−06	81,000
PSO-DE	−30,665.538	−30,665.538	−30,665.538	8.30E−10	70,100
CULDE	−30,665.538	−30,665.538	−30,665.538	1.00E−07	100,100
DE	−30,665.539	−30,665.509	−30,665.536	5.07E−03	240,000
HS	−30,665.500	–	–	–	65,000
CRGA	−30,665.520	−30,660.313	−30,664.398	1.60E+00	54,400
SAPF	−30,665.401	−30,656.471	−30,655.922	2.04E+00	500,000
SMES	−30,665.539	−30,665.539	−30,665.539	0.00E+00	240,000
ABC	−30,665.539	−30,665.539	−30,665.539	0.00E+00	240,000
DELC	−30,665.539	−30,665.539	−30,665.539	1.00E−11	50,000
DEDS	−30,665.539	−30,665.539	−30,665.539	2.70E−11	225,000
HEAA	−30,665.539	−30,665.539	−30,665.539	7.40E−12	200,000
ISR	−30,665.539	−30,665.539	−30,665.539	1.10E−11	192,000
α-simplex	−30,665.539	−30,665.539	−30,665.539	4.20E−11	305,343
EPSO	−30,665.538	−30,665.538	−30,665.538	1.07E−11	50,000
Jaya algorithm	**−30,665.5403**	**−30,665.5403**	**−30,665.5403**	**1.3861e−11**	35,200.3
SAMP-Jaya	**−30,665.5403**	**−30,665.5403**	**−30,665.5403**	4.5922e−11	11,747.6
SAMPE-Jaya (for $P = 10$ and $ES = 2$)	**−30,665.5403**	**−30,665.5403**	**−30,665.5403**	4.5251e−11	**11,247.6**

Source: The results of this table except the results of Jaya and SAMP-Jaya algorithm are taken from Ngo et al. (2016)

P population size, ES elite size

formed better or competitive in comparison with the rest of the algorithms. The SAMPE-Jaya algorithm requires less number of mean function evaluations in comparison with the other algorithms. It can be concluded from above results that the proposed SAMPE-Jaya algorithm is performing well on constrained benchmark problems as compared to the rest of the algorithms.

3.3 Analysis of results related to constrained engineering design problems

Furthermore, the capability of proposed the SAMPE-Jaya algorithm is tested in this section by applying it on four constrained mechanical designs benchmark problems taken from the literature. Description of the problems can be found from the literature (Ngo et al. 2016). Problem 1 is the minimization of the welded beam design cost. Problem 2 is the minimization of the total cost of a pressure vessel. Problem 3 is the problem of tension/compression spring design. Minimization of the weight of the spring is the objective of this problem. Problem 4 is a minimization problem of design of speed reducer.

Table 15 presents the statistical results over 30 runs of 22 algorithms for the test problem 1. It can be observed from this table that for the welded beam design problem proposed SAMPE-Jaya algorithm has obtained same best and mean values as compared to elitist-TLBO (Rao and Waghmare 2014) EPSO, SAMP-Jaya, Jaya algorithm and DE-PSO but the same is better in comparison with the rest of the algorithms. NM-PSO had obtained better value of the objective function. However, mean function value obtained by this approach is inferior to the proposed SAMPE-Jaya algorithm. Standard deviation (SD) and the worst solution obtained by the SAMPE-Jaya algorithm are superior to the rest of the algorithms considered for comparison. It requires lesser numbers of function evaluations for obtaining the best value as comparison to the remaining algorithms considered for the comparison. The SAMPE-Jaya algorithm is superior in

Table 14 Comparison of the statistical result for constrained problem 6

Method	Best	Worst	Average	SD	MFEs
ASCHEA	−1	N/A	−0.9999	N/A	1,500,000
CULDE	−0.9954	−0.6399	−0.7886	1.15E−01	100,100
ABC	−1.0000	−1.0000	−1.0000	0.00E+00	240,000
PSO-DE	−1.0050	−1.0050	−1.0050	3.80E−12	140,100
SMES	−1.0000	−1.0000	−1.0000	2.09E−04	240,000
SAPF	−1.0000	−0.8870	−0.9640	3.01E−01	500,000
GA	0.9999	−0.9998	0.9999	5.99E−05	320,000
ISR	−1.0010	−1.0010	−1.0010	8.20E−09	349,200
SR	−1.0000	−1.0000	−1.0000	1.90E−04	229,000
HEAA	−1.0000	−1.0000	−1.0000	5.20E−15	200,000
DELC	−1.0000	−1.000	−1.0000	2.10E−06	200,000
DEDS	−1.0005	−1.0005	−1.0005	1.90E−08	225,000
DE	−1.0252	−1.0252	−1.0252	0.00E+00	8000,000
α-simplex	−1.0005	−1.0005	−1.0005	8.50E−14	310,968
PESO	−0.9939	−0.4640	−0.7648	N/A	350,000
PSO	− 1.0050	−1.0043	−1.0049	1.00E+00	140,100
EPSO	−1.0000	−0.9998	−1.0000	4.35E−05	100,000
HM	−0.9997	−0.9978	−0.9989	N/A	1,400,000
CGWO	N/A	N/A	−0.9681	N/A	N/A
Jaya algorithm	−1.000000	−1.000000	−1.000000	6.41429e−16	27,278.3
SAMP-Jaya	−1.000000	−1.000000	−1.000000	**6.41429e−20**	23,165.0
SAMPE-Jaya (for $P = 25$ and ES = 2)	−1.000000	−1.000000	−1.000000	6.41429e−19	**22,065.0**

Source: The results of this table except the results of Jaya and SAMP-Jaya algorithm are taken from Ngo et al. (2016)
P population size, *ES* elite size

Table 15 Comparison of the statistical result for welded beam design problem

Method	Best	Worst	Mean	SD	MFEs
CDE	1.733460	–	1.768150	–	240,000
UPSO	1.921990	–	2.837210	6.83E−01	100,000
DE	1.733461	1.824105	1.768158	2.21E−02	204,800
SC	2.385435	6.399679	3.002588	9.60E−01	33,095
MGA	1.824500	1.995000	1.919000	5.37E−02	–
NM-PSO	**1.724717**	1.733393	1.726373	3.50E−03	80,000
PSO-DE	1.724852	**1.724852**	**1.724852**	6.70E−16	66,600
HPSO	1.724852	1.814295	1.749040	4.01E−02	81,000
CPSO	1.728024	1.782143	1.748831	1.29E−02	240,000
CAEP	1.724852	3.179709	1.971809	4.43E−01	50,020
GA4	1.728226	1.993408	1.792654	7.47E−02	80,000
GA3	1.748309	1.785835	1.771973	1.12E−02	900,000
EPSO	1.724853	1.747220	1.728219	5.62E−03	50,000
WOA	–	–	1 .7320	0 .0226	9,900
PSO	–	–	1 .7422	0 .01275	13,770
GSA	–	–	3 .5761	1 .2874	10,750
TEOA	1.725284	1.931161	1.768040	0.0581661	–
CGWO	1.725450	2.435700	2.428900	1.35780	–
ETLBO	1.724852	**1.724852**	**1.724852**	0.033000	99,999
Jaya algorithm	1.724852	**1.724852**	**1.724852**	2.2e−08	4739.00
SAMP-Jaya	1.724852	**1.724852**	**1.724852**	6.70e−16	**3618.25**
SAMPE-Jaya (for $P = 20$ and ES = 2)	1.724807	**1.724807**	**1.724807**	**2.75E−17**	4641.00

Bold value shows better solution
P population size, *ES* elite size
Source: The results of this table except the results of Jaya and SAMP-Jaya algorithm are taken from Ngo et al. (2016)

terms of robustness as compared to the remaining algorithms for the welded beam design problem. It requires approximately 98.06, 95.35, 97.73, 85.97, 94.19, 93.03, 94.27, 98.06, 90.72, 94.19, 99.48, 90.71, 95.35 and 20.06% lesser function evaluations in comparison with the GA3, GA4, CAEP, CPSO, HPSO, PSO-DE, MGA, SC, DE, UPSO, EDE, EPSO, ETLBO, WOA, PSO, GSA (Mirjalili and Lewis 2016) and Jaya algorithm, respectively. It can be concluded based on this results that for the welded beam design problem the SAMPE-Jaya algorithm performs better as compared to the rest of the algorithms.

Table 16 presents the comparison of statistical results of 25 algorithms for the test problem 2. Performance of the SAMPE-Jaya algorithm is superior to the rest of the algorithm for pressure vessel design problem in terms of mean solution, best solution and worst solution. The value reported by using CGWO (Kohli and Arora 2017) has obtained better values of best function value and mean function value. It is due to the consideration of value of decision variables x_1(thickness of shell), and x_2(thickness of head) as continuous variables instead of discrete variables. Therefore, the results reported by the CGWO are not feasible. The SD value produced by Jaya algorithm is better in comparison with the rest of the algorithms. The number of function evaluations required for this problem is lesser for the present methods in comparison with the remaining algorithms expect elitist-TLBO. It can be concluded based on these results that the SAMPE-Jaya algorithm performs better for this problem also as compared to the rest of the algorithms in term of quality of solution.

The comparison of the statistical results over 30 independent runs of 27 algorithms for the test problem 3 is shown in Table 17. For tension/compression problem, SAMPE-Jaya algorithm is better in comparison with the other algorithms in terms of best value except NM-PSO. In terms of mean value

Table 16 Comparison of the statistical result for pressure vessel design problem

Method	Best	Worst	Mean	SD	MFEs
GA3	6288.7445	6308.4970	6293.8432	7.41E+00	900,000
GA4	6059.9463	6469.3220	6177.2533	1.30E+02	80,000
CPSO	6061.0777	6363.8041	6147.1332	8.64E+01	240,000
HPSO	6059.7143	6288.6770	6099.9323	8.62E+01	81,000
NM-PSO	5930.3137	5960.0557	5946.7901	9.16E+00	80,000
G-QPSO	6059.7208	7544.4925	6440.3786	4.48E+02	8000
QPSO	6059.7209	8017.2816	6440.3786	4.79E+02	8000
PSO	6693.7212	14076.324	8756.6803	1.49E+03	8000
CDE	6059.7340	6371.0455	6085.2303	4.30E+01	204,800
UPSO	6544.2700	N/A	9032.5500	9.95E+02	100,000
PSO-DE	6059.7140	N/A	6059.7140	N/A	42,100
ABC	6059.7140	N/A	6245.3080	2.05E+02	30,000
$(\mu+\lambda)$-ES	6059.7016	N/A	6379.9380	2.10E+02	30,000
TLBO	6059.7143	N/A	6059.7143	N/A	10,000
MBA	5889.3216	6392.5062	6200.6477	1.60E+02	70,650
EPSO	5885.3383	7315.6752	6254.1804	4.24E+02	10,000
	5885.3328	6076.6205	5920.8442	5.21E+01	100,000
WOA	N/A	N/A	6068 .05	65 .6519	6300
GSA	N/A	N/A	8932 .95	683 .5475	7110
PSO	N/A	N/A	6531 .10	154 .3716	14790
TEOA	5887.511073	6134.187981	5942.565917	62.2212	N/A
WOA[a]	5034.180	6188.110	5783.582	254.505	N/A
ETLBO	5885.3336	5887.3338	5956.6921	11.0000	**4992**
Jaya algorithm	**5872.21287**	**5872.21287**	**5872.2127**	**4.1369e−13**	7004.33
SAMP-Jaya	**5872.21287**	**5872.21287**	**5872.2127**	5.0424e−12	6513.33
SAMPE-Jaya (for $P = 10$ and ES = 2)	**5872.21287**	**5872.21287**	**5872.2127**	2.3286E−12	6003.00

Bold value shows better solution
P population size, ES elite size
Source: The results of this table except the results of Jaya and SAMP-Jaya algorithm are taken from Ngo et al. (2016)
[a]Infeasible solution

and worst value of function, NM-PSO is superior. Function evaluations required by SAMPE-Jaya algorithm are lesser in comparison with the remaining algorithms expect EPSO. It is reported that the CGWO (Kohli and Arora 2017) has obtained the better values of the best, worst and mean for this problem. However, second constrained (g_2) imposed on this is violated. It seems that lower value of penalty was used. Hence, results reported by using CGWO are infeasible.

The comparison of the statistical results over 30 independent runs of fourteen algorithms for the test problem 4 is shown in Table 18. For speed reducer problem, the performance of the SAMPE-Jaya algorithm is better as compared to the rest of the algorithms in term of all parameters considered for the comparison except SD. The Jaya algorithm has achieved better value of SD for this problem. However, the MFEs required for this problem using SAMPE-Jaya algorithm is reduced by 13.61 and 51.86% in comparison with SAMP-Jaya and Jaya algorithms, respectively. The present approach has produced improved results for this problem. The best value of the objective function is improved by 8%.

It can be concluded from Tables 15, 16, 17 and 18 that the performance of the proposed SAMPE-Jaya algorithm is better or competitive to the other algorithms for the standard mechanical design problems. It requires comparatively less function evaluations for achieving the optimal solutions.

Furthermore, the performance of the proposed SAMPE-Jaya algorithm is tested on three large-scale problems taken from the literature (Cheng and Jin 2015). The dimensions of the considered problems are: 100, 500 and 1000. Table 19 presents comparison of the proposed SAMPE-Jaya algorithm

Table 17 Comparison of the statistical result for tension/compression spring problem

Method	Best	Worst	Average	SD	MFEs
HPSO	0.012665	0.012719	0.012707	1.58E−05	81,000
GA4	0.012681	0.012973	0.012742	5.90E−05	80,000
NM-PSO	**0.012630**	**0.012633**	**0.012631**	**8.47E−07**	80,000
CPSO	0.012675	0.012924	0.012730	5.20E−04	240,000
CAEP	0.012721	0.015116	0.013568	8.42E−04	50,020
PSO	0.012857	0.071802	0.019555	1.16E−02	2000
DELC	0.012665	0.012666	0.012665	1.30E−07	20,000
G-QPSO	0.012665	0.017759	0.013524	1.27E−03	**2000**
QPSO	0.012669	0.018127	0.013854	1.34E−03	2000
HEAA	0.012665	0.012665	0.012665	1.40E−09	24,000
DE	0.012670	0.012790	0.012703	2.70E−05	204,800
UPSO	0.013120	N/A	0.022940	7.20E−03	100,000
DEDS	0.012665	0.012738	0.012669	1.30E−05	24,000
$(\mu + \lambda)$-ES	0.012689	N/A	0.013165	3.90E−04	30,000
SC	0.012669	0.016717	0.012923	5.90E−04	25,167
PSO-DE	0.012665	0.012665	0.012665	1.20E−08	24,950
CDE	0.012670	N/A	0.012703	N/A	240,000
ABC	0.012665	N/A	0.012709	1.28E−02	30,000
GA3	0.012705	0.012822	0.012769	3.94E−05	900,000
EPSO	0.012670	0.016911	0.014056	1.27E−03	2000
	0.012669	0.014218	0.013030	3.64E−04	10,000
WOA	N/A	N/A	0 .0127	0.0003	4410
GSA	N/A	N/A	0.136	0.0026	4980
PSO	N/A	N/A	0 .0139	0.0033	5460
TEOA	0.012665	0.012715	0.012685	4.4079e−06	–
CGWO[a]	0.0119598	0.0121791	0.0121749	1.039E−05	–
ETLBO	0.012665	0.012678	0.012758	0.0004900	7022
Jaya algorithm	0.012665	0.012697	0.012732	1.89511e−05	7744.66
SAMP-Jaya	0.012664	0.013193	0.012714	9.25221e−05	6861.00
SAMPE-Jaya (for $P = 15$ and ES = 2)	0.012650	0.012654	0.012651	6.2558E−07	6095.20

Bold value shows better solution;
P population size, ES elite size
[a]Infeasible solution

with other algorithms. It can be observed from Table 19 that the performance of SAMPE-Jaya algorithm is better for Rosenbrock and Griewank function and competitive for Rastrigin function as compared to the other algorithms for the considered large-scale problems. Hence, it can be concluded based on these results that the proposed SAMPE-Jaya algorithm is performing satisfactorily for the large-scale problems also.

After evaluating the performance of the SAMPE-Jaya algorithm on standard benchmark problems, a practical case study of micro-channel heat sink (MCHS) design optimization is considered. The motivation of the present work is to make an attempt to see whether any improvement is possible in the design of MCHS by using a SAMPE-Jaya algorithm.

The next section presents the application of the SAMPE-Jaya algorithm for the design optimization of MCHS.

4 Application of SAMPE-Jaya algorithm for the case study of a heat sink design

This case study was introduced by Husain and Kim (2010). A 10 mm × 10 mm × 0.42 mm silicon-based micro-channel heat sink (MCHS) considered by Husain and Kim (2010) is

Table 18 Comparison of the statistical result for speed reducer design problem

Best	Worst	Mean	SD	MFEs	Best
SC	2994.744241	3009.964736	3001.758264	4.00E+00	54,456
$(\mu+\lambda)$-ES	2996.348000	N/A	2996.348000	0.00E+00	30,000
DEDS	2994.471066	2994.471066	2994.471066	3.60E−12	30,000
MDE	2996.356689	N/A	2996.367220	8.20E−03	24,000
DELC	2994.471066	2994.471066	2994.471066	1.90E−12	30,000
HEAA	2994.499107	2994.752311	2994.613368	7.00E−02	40,000
PSO-DE	2996.348167	2996.348204	2996.348174	6.40E−06	54,350
ABC	2997.058000	N/A	2997.058000	0.00E+00	30,000
TLBO	2996.348170	N/A	2996.348170	0.00E+00	10,000
MBA	2994.482453	2999.652444	2996.769019	1.56E+00	6300
EPSO	2994.471072	2994.471227	2994.471119	4.71E−05	5000
ETBO	2996.348	2996.348	2996.348	0.000045	9.988
Jaya algorithm	**2760.673988**	**2760.673988**	**2760.673988**	1.3875e−12	6720
SAMP-Jaya	**2760.673988**	**2760.673988**	**2760.673988**	2.5412e−11	3744.66
SAMPE-Jaya (for $P = 15$ and ES = 2)	**2760.673988**	**2760.673988**	**2760.673988**	**4.7934E−13**	**3235**

Bold value shows better solution
P population size, ES elite size
Source: The results of this table except the results of Jaya and SAMP-Jaya algorithm are taken from Ngo et al. (2016)

Table 19 Performance of SAME-Jaya with large-scale problems

Function		Dimension		
		100	500	1000
Rosenrock	SAMPE-Jaya	**4.68E−27** ($P = 15$ and ES = 2)	**2.73E−15** ($P = 20$ and ES = 2)	**1.60E−19** (P = 15 and ES = 2)
	CCPSO	7.73E−14	7.73E−14	5.18E−13
	MLCC	9.02E−15	4.30E−13	8.46E−13
	Sep-CMA-ES	9.02E−15	2.25E−14	7.81E−13
Rastrigin	SAMPE-Jaya	3.25E+01(P = 15 and ES = 2)	7.80E+01(P = 20 and ES = 2)	1.01E+02 (P = 25 and ES = 4)
	CCPSO	**6.08E+00**	**5.79E+01**	**7.82E+01**
	MLCC	2.31E+01	6.67E+01	1.09E+02
	Sep-CMA-ES	2.31E+01	2.12E+02	3.65E+02
Griewank	SAMPE-Jaya	**9.15E−04** (P = 15 and ES = 2)	**1.85E+02** (P = 15 and ES = 2)	**8.01E+02** (P = 20 and ES = 2)
	CCPSO	4.23E+02	7.24E+02	1.33E+03
	MLCC	1.50E+02	9.25E+02	1.80E+03
	Sep-CMA-ES	4.31E+00	2.93E+02	9.10E+02

CCPSO: cooperatively coevolving particle swarm optimization; MLCC: multilevel cooperative coevolution; Sep-CMA-ES: separable covariance matrix adaptation evolution strategy
P population size, ES elite size
Source: results of this table except the results of SAMPE-Jaya algorithm are taken from Cheng and Jin (2015)

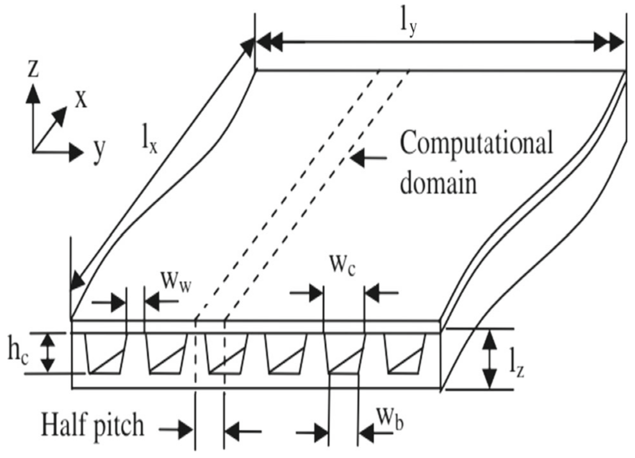

Fig. 6 Conventional diagram of trapezoidal MCHS (Husain and Kim 2010)

Table 20 Design variables and their ranges for case study

Limits	Variables		
	$\alpha(w_c/h_c)$	$\beta(w_w/hc)$	$\gamma(w_b/w_c)$
Upper	0.10	0.02	0.50
Lower	2.50	1.0	1.00

shown in Fig. 6. Water was used as coolant liquid, and it flowed into the micro-channel and left at the outlet. The silicon substrate occupied the remaining portion of heat sink. No slip condition was assumed at the inner walls of the channel, i.e., $u = 0$.

The thermal condition in the z-direction was given as:

$$- k_s((\partial T_s)/(\partial x_i)) = q \text{ at } z = 0 \text{ and } k_s((\partial T_s)/(\partial x_i)) = 0 \text{ at } z = l_z \tag{4.1}$$

The design variables considered by Husain and Kim (2010) were $\alpha = w_c/h_c$, $\beta = w_w/h_c$, and $\gamma = w_b/w_c$, where w_c is the micro-channel width at bottom; w_b is the micro-channel width at top; w_w is the fin width and hc is the micro-channel depth. h_c is kept 400μm during the whole optimization procedure.

In this case study, two objective functions were considered and those were (i) thermal resistance associated with heat transfer performance and (ii) the pumping power to drive the coolant or to pass the coolant through the micro-channel. Table 20 shows design variables α, β and γ, and their limits for both rectangular ($w_b/w_c = 1$) and trapezoidal ($0.5 < w_b/w_c < 1$) cross sections of MCHS.

The two objective functions considered are, thermal resistance and pumping power. The thermal resistance is given by:

$$R_{TH} = \Delta\text{Tmax}/(As * q) \tag{4.2}$$

where As is area of the substrate subjected to heat flux and ΔTmax is the maximum temperature in MCHS, which is given as:

$$\Delta\text{Tmax} = T_{s,o} - T_{f,i} \tag{4.3}$$

The pumping power to move the coolant (water) through MCHS is calculated as:

$$\bar{P} = n^* u_{avg}{}^* A_c{}^* \Delta p \tag{4.4}$$

where Δp was the pressure drop and u_{avg} was the mean velocity.

Pumping power and thermal resistance compete with each other because a decrease in pumping power contributes to an increase in thermal resistance. Husain and Kim (2010) calculated the objectives by using Navier–Stokes and heat conduction equations at specific design points. The response surface approximation (RSA) was then used to obtain the functional forms of the two objective functions. The polynomial responses are expressed as:

$$\begin{aligned} R_{TH} = {} & 0.096 + 0.31^*\alpha - 0.019^*\beta - 0.003^*\gamma \\ & -0.007^*\theta^*\beta + 0.031^*\alpha^*\gamma - 0.039^*\beta^*\gamma \\ & +0.008^*\alpha^2 + 0.027^*\beta^2 + 0.029^*\gamma^2 \qquad (4.5) \\ \bar{P} = {} & 0.94 - 1.695^*\alpha - 0.387^*\beta - 0.649^*\gamma \\ & -0.35^*\alpha^*\beta + 0.557^*\alpha^*\gamma - 0.132^*\beta^*\gamma \\ & +0.903^*\alpha^2 + 0.016^*\beta^2 + 0.135^*\gamma^2 \qquad (4.6) \end{aligned}$$

The design variables α, β and γ are in terms of the ratios of the micro-channel width at bottom to depth (i.e., w_c/h_c), fin width to the micro-channel depth (i.e., w_w/h_c) and micro-channel width at top to width at bottom (w_b/w_c), respectively. Solving Eqs. (4.5) and (4.6) for α, β and γ will give the optimum values of the dimensions of the micro-channel, i.e., w_c, w_w, w_b and h_c. The three design variables α, β and γ have significant effect on the thermal performance of micro-channel heat sink. Design and manufacturing constraints can be handled in a better way, and Pareto optimal solutions can be spread over the whole range of variables. The Pareto optimal analysis provides information about the active design space and relative sensitivity of the design variables to each objective function which is helpful in comprehensive design optimization. Thus, Eqs. (4.5) and (4.6) have the physical meaning. The design variables and rages are shown in Table 20.

The solution obtained by a priori approach depends on the weights assigned to various objective functions by designer or decision maker. By changing the weights of importance of different objective functions, a dense spread of the Pareto points can be obtained. Following a priori approach in the present work, the two objective functions are combined into

Table 21 Design variables of objective functions by using TLBO, Jaya, SAMPE-Jaya and hybrid MOEA for case study 3

S. no.	Design variables											
	α				B				γ			
	TLBO	Jaya	SAMPE-Jaya	Hybrid MOEA	TLBO	Jaya	SAMPE-Jaya	Hybrid MOEA	TLBO	Jaya	SAMPE-Jaya	Hybrid MOEA
1	–	–	0.63399	–	–	–	0.02	–	–	–	1	–
2	0.7952	0.448167	0.7541864	0.994	0.040	0.879132	0.8945767	0.140	0.534	0.9999	1	0.528
3	0.595	0.068346	0.5704825	0.459	0.735	0.526466	0.6177118	0.693	0.5912	0.55973	0.5903021	0.991
4	0.325	0.324	0.2725593	0.096	0.745	0.721	0.706121	0.638	0.601	0.59701	0.7248777	0.982
5	0.132	0.0324	0.1	0.000	0.7601	0.857001	0.7258657	0.886	0.6299	0.9692	0.68915	0.971
6	0.1067	0.124	0.1	0.000	0.69	0.67	0.7480976	0.609	0.528	0.5692	0.6818574	0.456
7	–	–	0.0324	–	–	–	0.857001	–	–	–	0.9692	–

Table 22 Comparison of results of hybrid MOEA, numerical analysis, TLBO, Jaya and SAMPE-Jaya for case study 3

S. no.	Hybrid MOEA (Husain and Kim 2010)		Numerical Analysis (Husain and Kim 2010)		TLBO(Rao et al. 2016)		Jaya (Rao et al. 2016)		SAMPE-Jaya			
	R_{TH}	$\bar{P}$	R_{TH}	$\bar{P}$	R_{TH}	$\bar{P}$	R_{TH}	$\bar{P}$	R_{TH}	$\bar{P}$	P	ES
1	–	–	–	–	–	–	–	–	0.163280	0.068143	15 & 20	2 & 4
2	0.145	0.097	0.143	0.094	0.143	0.0931	0.138308	0.086174	**0.13830**	**0.086144**	15 & 20	2 & 4
3	0.118	0.195	0.119	0.175	0.1172	0.1933	0.116959	0.192433	**0.11694**	**0.19211**	15 & 20	2 & 4
4	0.100	0.455	0.100	0.410	0.1033	0.4054	0.103345	0.402558	**0.10297**	**0.40195**	15 & 20	2 & 4
5	0.094	0.633	0.094	0.634	0.094	**0.6282**	0.093279	**0.6282**	0.**09225**	0.63266	15 & 20	2 & 4
6	0.093	0.828	0.094	0.821	0.0927	0.6966	0.093779	0.6444	**0.09321**	**0.6427**	15 & 20	2 & 4
7	–	–	–	–	–	–	–	–	0.09327	0.6253	15 & 20	2 & 4

P population size, *ES* elite size

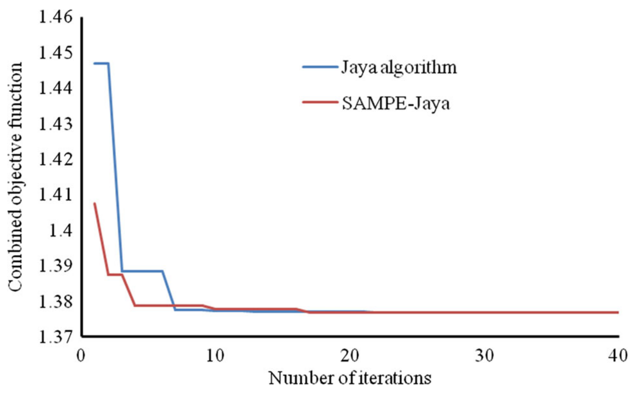

Fig. 7 Convergence of Jaya and SAMPE-Jaya algorithms for MCHS problem with equal weights of the objective functions

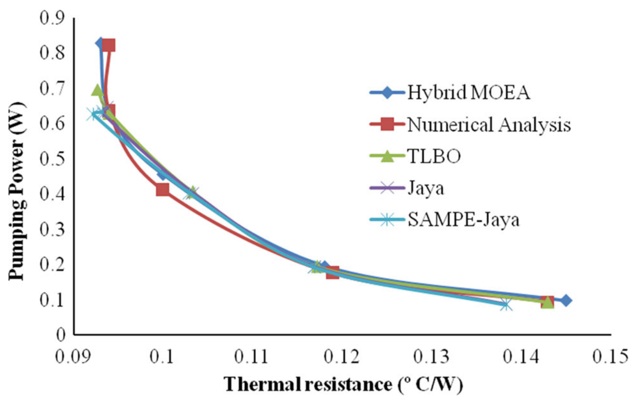

Fig. 8 Pareto optimal curve for MCHS problem

a single objective function. The combined objective function Z is formed as:

$$\text{Minimize; } Z = w_1\left(\frac{Z1}{Z1min.}\right) + w_2\left(\frac{Z2}{Z2min.}\right) Z1 = R_{\text{th}} \text{ and } Z2 = \bar{P} \tag{4.7}$$

where w_1 and w_2 are the weighs assigned to the objective functions $Z1$ and $Z2$, respectively, between 0 and 1. These weights can be assigned to the objective functions according to the designer's/decision maker's priorities. $Z1min$ and $Z2min$ are the optimum values of the $Z1$ and $Z2$, respectively, obtained by solving the optimization problem when only one objective is considered at a time and ignoring the

Table 23 Hypervolume for case study of heat sink

	Hybrid MOEA	Numerical method	TLBO	Jaya	SAMPE-Jaya
Hypervolume	0.072884	0.074248	0.07389326	0.074665546463	0.0750687665

Table 24 Standard deviation of the solutions in each iteration using Jaya and SAMPE-Jaya algorithms

Iteration no.	Jaya algorithm	SAMPE-Jaya	Iteration No.	Jaya algorithm	SAMPE-Jaya
1	2.068073	2.92929	21	0.000676	0.001095
2	0.694924	1.687512	22	0.00068	0.00082
3	0.624445	0.998083	23	0.00068	0.000553
4	0.379873	0.598621	24	0.000669	0.000377
5	0.204628	0.392295	25	0.000598	0.000314
6	0.121914	0.243063	26	0.00051	0.000314
7	0.102022	0.1718	27	0.000442	0.000314
8	0.080869	0.108057	28	0.000369	0.000314
9	0.06223	0.056515	29	0.000313	0.000314
10	0.038211	0.047308	30	0.000232	0.000314
11	0.020686	0.011159	31	0.000182	0.000314
12	0.010589	0.008848	32	0.000139	0.000314
13	0.007294	0.007002	33	0.000136	0.000314
14	0.004371	0.005212	34	7.29E−05	0.000319
15	0.003819	0.00377	35	5.58E−05	0.000319
16	0.003189	0.002354	36	4.48E−05	0.000319
17	0.003203	0.002365	37	3.82E−05	0.000319
18	0.002541	0.002365	38	3.81E−05	0.000319
19	0.000869	0.002259	39	3.81E−05	0.000319
20	0.000738	0.001232	40	3.81E−05	0.000319

other. Now, Eq. (4.7) can be used to optimize both the objectives simultaneously.

Husain and Kim (2010) used these surrogate models and a hybrid MOEA involving NSGA-II and sequential quadratic programming (SQP) method to find out the Pareto optimal solutions. Husain and Kim (2010) used NSGA-II algorithm to obtain Pareto optimal solutions, and the solutions were refined by selecting local optimal solutions for each objective function using a sequential quadratic programming (SQP) method with NSGA-II solutions as initial solutions. Then K-means clustering method was then used to group the global Pareto optimal solutions in to five clusters. The whole procedure was termed as a hybrid multi-objective optimization evolutionary algorithm (MOEA).

Now, the model considered by Husain and Kim (2010) is attempted using SAMPE-Jaya algorithm. Husain and Kim (2010) used a hybrid MOEA coupled with surrogate models to obtain the Pareto optimal solutions. Rao et al. (2016) used TLBO and Jaya algorithms to obtain the Pareto optimal solutions.

The values of the design variables given by the SAMPE-Jaya algorithm, Jaya algorithm, TLBO algorithm, hybrid MOEA and numerical analysis are shown in Table 21. Table 22 shows the results comparison of SAMPE-Jaya algorithm, Jaya algorithm, TLBO algorithm, hybrid MOEA and numerical analysis. It can be observed from Table 22 that the SAMPE-Jaya algorithm has performed better as compared with hybrid MOEA, Numerical analysis, TLBO and Jaya algorithms for different weights of the objective functions for the bi-objective optimization problem considered. The performance of TLBO algorithm comes next to Jaya algorithm. Figure 7 presents the convergence of Jaya and SAMPE-Jaya algorithms for MCHS problem with equal weights.

Figure 8 shows Pareto fronts obtained by using SAMPE-Jaya algorithm, Jaya algorithm, TLBO algorithm and hybrid MOE algorithm representing five clusters. Also, it can be observed that the SAMPE-Jaya algorithm has provided better results than the hybrid MOEA proposed by Husain and Kim (2010). Every peak end of the Pareto curve represents the higher value of one objective and lower value of another.

In order to make a fair comparison between the performances of the algorithms for the multi-objective optimization problems a quantity measure index known as hypervolume is calculated. Hypervolume is defined as the *n*-dimensional space that is enclosed by a set of points. It encapsulates in a single unary value a measure of the spread of the solutions along the Pareto front, as well as the closeness to the Pareto optimal front.

Table 23 presents the value of hypervolume obtained by the various algorithms for this case study. An observation can be made from Table 23 that value of the hypervolume obtained by the SAMPE-Jaya for the heat sink design optimization is better than the MOGA, numerical method, TLBO and Jaya algorithm. Hence, it can be concluded that the performance of the SAMPE-Jaya algorithm is better than the hybrid MOEA, TLBO and Jaya algorithms.

Furthermore, for checking the diversity of the search process, standard deviation of the objective function values is calculated and recorded after each iteration and shown in Table 24. It can be observed from this table that the value of standard deviation is not zero after any iteration, and it is different also for each iteration. This shows that the algorithm is continuously exploring the search process, and it does not fall in the trap of local minima. The early convergence of Fig. 7 shows that the algorithm has reached to global optimum value or near global optimum value of the objective function in a few iterations.

The next section presents the conclusions of this work.

5 Conclusions

This study proposes an elitist-based self-adaptive multi-population Jaya algorithm. The performance of the proposed algorithm is examined on the small as well as large-scale unconstrained and constrained benchmark problems in addition to the computationally expensive problems of the CEC 2015. The Friedman rank test is used to find the average rank of the algorithm, and it is observed that the proposed algorithm is better than the other algorithms. Furthermore, the proposed method is used for the design optimization problem of a micro-channel heat sink (MCHS). In the proposed method, multi-population search scheme is used for enhancing the search mechanism of the Jaya algorithm which divides the population into a number of subpopulations adaptively. This subpopulation-based scheme can be easily integrated with single population-based advanced optimization algorithms. The results of the SAMPE-Jaya for the benchmark problems are found better or competitive to the latest reported methods used for optimization of the same problems. In the case of MCHS, the proposed SAMPE-Jaya algorithm has obtained better Pareto optimal solutions as compared to those of hybrid MOEA, numerical analysis, TLBO and Jaya algorithms.

The concept of SAMPE-Jaya algorithm is simple, and it is not having any algorithmic-specific parameters to be tuned. Therefore, it may be easily implemented on the engineering problems where the problems are usually complicated with a number of design parameters and having the discontinuity in the objective function.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Amirjanov A (2006) The development of a changing range genetic algorithm. Comput Methods Appl Mech Eng 195:2495–2508

Andersson M, Bandaru S, Ng AHC, Syberfeldt A (2015) Parameter tuned CMA-ES on the CEC'15 expensive problems. In: IEEE congress on evolutionary computation, Sendai, Japan, 2015

Becerra R, Coello CAC (2006) Cultured differential evolution for constrained optimization. Comput Methods Appl Mech Eng 195:4303–4322

Bergh FV, Engelbrecht AP (2004) A cooperative approach to particle swarm optimization. IEEE Trans Evolut Comput 8(3):225–239

Branke J, Kaußler T, Schmidt C, Schmeck H (2000) A multi-population approach to dynamic optimization problems. Adaptive computing in design and manufacturing. Springer, Berlin, pp 299–308

Cantu-Paz E (1998) A survey of parallel genetic algorithms. IllGAL report 97003, The University of Illinois, 1997. ftp://ftp-lligal.ge.uiuc.edu/pub/papers/IlliGALs/97003.ps.Z

Cheng R, Jin Y (2015) A competitive swarm optimizer for large scale optimization. IEEE Trans Cybern 45(2):191–204

Coello CAC, Becerra RL (2004) Efficient evolutionary optimization through the use of a cultural algorithm. Eng Optim 36:219–236

Cruz C, González JR, Pelta DA (2011) Optimization in dynamic environments: a survey on problems, methods and measures. Soft Comput 15(7):1427–1448

Eberhart RC, Kennedy J (1995) A new optimizer using particle swarm theory. In: Sixth international symposium on micro machine and human science, Nagoya, Japan, pp 39–43

Hamida SB, Schoenauer M (2002) ASCHEA: new results using adaptive segregational constraint handling. In: Proceedings of the world on congress on computational intelligence, pp 884–889

Haupt RL, Haupt SE (2004) Practical genetic algorithms, 2nd edn. Wiley, Hoboken

Husain V, Kim KY (2010) Enhanced multi-objective optimization of a micro-channel heat sink through evolutionary algorithm coupled with multiple surrogate models. Appl Therm Eng 30:1683–1691

Irawan CA, Salhi S, Drezner ZJ (2016) Heuristics: hybrid metaheuristics with VNS and exact methods: application to large unconditional and conditional vertex p-centre problems. J Heuristics 22(4):507–537

Jin N, Rahmat-Samii Y (2010) Hybrid real-binary particle swarm optimization (HPSO) in engineering electromagnetic. IEEE Trans Antennas Propag 58(12):3786–3794

Joaquin D, Salvador G, Daniel M, Francisco H (2016) A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evolut Comput 1(1):3–18

Karaboga D, Basturk B (2007) Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems. In: LNAI 4529. Springer, Berlin, pp 789–798

Kaveh A, Dadras A (2017) A novel meta-heuristic optimization algorithm: thermal exchange optimization. Adv Eng Softw 110:69–84

Kohli M, Arora S (2017) Chaotic grey wolf optimization algorithm for constrained optimization problems. J Comput Des Eng. https://doi.org/10.1016/j.jcde.2017.02.005

Koziel S, Michalewicz Z (1999) Evolutionary algorithms, homomorphous mappings and constrained parameter optimization. IEEE Trans Evolut Comput 7:19–44

Lampinen J (2002) A constraint handling approach for the differential evolution algorithm. In: IEEE congress on evolutionary computation, vol 2, pp 1468–1473

Lau HC, Raidl GR, Van Hentenryck PJ (2016) New developments in metaheuristics and their applications. J Heuristics 22:359

Li C, Yang S (2008) Fast multi-swarm optimization for dynamic optimization problems. In: Fourth international conference on natural computation, ICNC'08, vol 7. IEEE, pp 624–628

Li C, Nguyen TT, Yang M, Yang S, Zeng S (2015) Multi-population methods in un-constrained continuous dynamic environments: the challenges. Inf Sci 296:95–118

Liang JJ, Qin AK (2006) Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Trans Evolut Comput 10(3):281–295

Liu H, Cai Z, Wang Y (2010) Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization. Appl Soft Comput 10:629–640

Mambrini A, Sudholt D (2014) Design and analysis of adaptive migration intervals in parallel evolutionary algorithms. In: Proceedings of the 2014 annual conference on genetic and evolutionary computation, pp 1047–1054

Mendes R, Kennedy J, Neves J (2004) The fully informed particle swarm: simpler, may be better. IEEE Trans Evolut Comput 8(3):204–210

Mezura-Montes E, Coello CAC (2006) A simple multi membered evolution strategy to solve constrained optimization problems. IEEE Trans Evolut Comput 9:1–17

Mirjalili S, Lewis A (2016) The whale optimization algorithm. Adv Eng Softw 95:51–67

Ngo TT, Sadollahb AJ, Kim H (2016) A cooperative particle swarm optimizer with stochastic movements for computationally expensive numerical optimization problems. J Comput Sci 13:68–82

Nguyen TT, Yang S, Branke J (2012) Evolutionary dynamic optimization: a survey of the state of the art. Swarm Evolut Comput 6:1–24

Nickabadi A, Ebadzadeh MM, Safabakhsh R (2011) A novel particle swarm optimization algorithm with adaptive inertia weight. Appl Soft Comput 11(4):3658–3670

Nseef SK, Abdullah S, Turky A, Kendall G (2016) An adaptive multi-population artificial bee colony algorithm for dynamic optimisation problems. Knowl Based Syst 104:14–23

Oca MA, Stutzle T (2009) Frankenstein's PSO: a composite particle swarm optimization algorithm. IEEE Trans Evolut Comput 13(5):1120–1132

Rao RV (2016a) Review of applications of TLBO algorithm and a tutorial for beginners to solve the unconstrained and constrained optimization problems. Dec Sci Lett 5:1–30

Rao RV (2016b) Teaching learning based optimization algorithm and its engineering applications. Springer, London

Rao RV (2016c) Jaya: a simple and new optimization algorithm for solving constrained and unconstrained optimization problems. Int J Ind Eng Comput 7(1):19–34

Rao RV, Patel VK (2012) An elitist teaching–learning-based optimization algorithm for solving complex constrained optimization problems. Int J Ind Eng Comput 3(4):535–560

Rao RV, Saroj A (2017) A self-adaptive multi-population based Jaya algorithm for engineering optimization. Swarm Evolut Comput. https://doi.org/10.1016/j.swevo.2017.04.008

Rao RV, Waghmare GG (2014) Complex constrained design optimisation using an elitist teaching–learning-based optimisation algorithm. Int J Metaheuristic 3(1):81–102

Rao RV, More KC, Taler J, Ocłoń P (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. Inf Sci 179(13):2232–2248

Runarsson TP, Xin Y (2000) Stochastic ranking for constrained evolutionary optimization. IEEE Trans Evolut Comput 4:284–294

Runarsson TP, Xin Y (2005) Search biases in constrained evolutionary optimization. IEEE Trans Syst Man Cybern C Appl Rev 35:233–243

Salmani HS, Eshghi K (2017) A metaheuristic algorithm based on chemotherapy science: CSA. J Optim. https://doi.org/10.1155/2017/3082024

Takahama T, Sakai S (2005) Constrained optimization by applying the constrained method to the nonlinear simplex method with mutations. IEEE Trans Evolut Comput 9(5):437–451

Tessema B, Yen GG (2006) A self-adaptive penalty function based algorithm for constrained optimization. In: IEEE congress on evolutionary computation, pp 246–253

Wang Y, Cai Z, Zhou Y, Fan Z (2009) Constrained optimization based on hybrid evolutionary algorithm and adaptive constraint handling technique. Struct multidiscip Optim 37:395–413

Yang S, Li C (2010) A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. IEEE Trans Evolut Comput 14(6):959–974

Zahara E, Kao YT (2009) Hybrid Nelder–Mead simplex search and particle swarm optimization for constrained engineering design problems. Expert Syst Appl 36:3880–3886

Zhang M, Luo W, Wang X (2008) Differential evolution with dynamic stochastic selection for constrained optimization. Inf Sci 178:3043–3074