An Improved JAYA Algorithm Based Test Suite Generation for Object Oriented Programs: A Model Based Testing Method

Madhumita Panda[1] and Sujata Dash[2(✉)]

[1] Maharaja Sriram Chandra Bhanja Deo University, Mayurbhanj, India
[2] Maharaja Sriram Chandra Bhanja Deo University, Baripada, India
sujata238dash@gmail.com

Abstract. The model based testing approaches are not always capable of suggesting exact optimized and prioritize test cases, like conventional approaches, therefore some popular metaheuristic algorithms are gradually fabricated with model based testing methodologies for generating optimized test data. The metaheuristic algorithms are complex in terms of their algorithm specific parameter settings; they cannot provide good results without proper parameter settings. In accordance with the above-described issues, here in this work a novel methodology is proposed for test suite generation, using an improved metaheuristic Jaya algorithm along with UML state machine model. Experimenting with the benchmark triangle classification problem, the results prove, the performance as well as exploitation capability of the improved JAYA algorithm is quite good; over the widely popular Differential Evolution algorithm.

Keywords: Model based testing · Improved Jaya algorithm · Metaheuristic algorithms

1 Introduction

The object-oriented program testing is quite complex and still remains a critical research area since decades [1, 2]. The traditional testing approaches are unsuitable for object-oriented testing, thus a different testing practice, model-based testing is followed for deriving test cases [3, 4]. Gradually the Nature inspired algorithms proved their efficiency in proving sub-optimal solutions in various fields of engineering [11], thus researchers started fabricating those nature inspired metaheuristics with the software testing process [5–7]. The popular and widely accepted nature inspired metaheuristics mainly includes the evolutionary and swarm based algorithms, starting from the Genetics algorithm to recently popular [19, 20, 22], Bacteria foraging algorithm (BFO), Grey wolf algorithm and the list goes on [11]. Every metaheuristic algorithm has its own specific set of parameters and without proper knowledge of those parameters the algorithms are unable to provide their best results [11]. Keeping in mind those problems arising in metaheuristics due to improper parameters settings, a parameter free algorithm known as teacher learning based algorithm (TLBO) was proposed [12]. The Teacher learning based algorithm

A. K. Luhach et al. (Eds.): ICAICR 2021, CCIS 1575, pp. 112–122, 2022.
https://doi.org/10.1007/978-3-031-09469-9_10

includes two stages and it has only two parameters, the size of the population and iteration numbers. Very recently keeping in mind the popularity of the TLBO algorithm, a parameter free algorithm, the JAYA algorithm was introduced, this algorithm is even simpler than the TLBO algorithm, having one stage only [12]. The JAYA algorithm is gradually getting popular, efficiently handling the engineering optimization problems [12], in facial emotion recognition [13], Dimensional optimization [14], economic optimization [15] etc. Thus, keeping in track with the above research findings, this paper proposes a novel improved JAYA algorithm as well as the framework for testing object-oriented programs. The metaheuristic Improved JAYA algorithm is employed in the proposed framework to automatically select a set of test suites to test the object-oriented triangle classification program. Results indicate that the JAYA algorithm provides good exploitation feature to generate test suits. The proposed work targets the following modules for fulfilling the above mentioned objectives,

- Generation of test suites for testing the feasible test sequences of triangle classification problem using a novel Improved JAYA algorithm.
- Generation of test suites for testing the feasible test sequences of triangle classification problem using Differential Evolution algorithm.
- A set of experiments were carried out using the standard triangle classification problem followed by an exhaustive comparison between the metaheuristics i.e. JAYA, Differential Evolution and improved JAYA algorithms.

The remaining sections of this work are systematized as follows, the Sect. 2 conveys a detailed explanation of the classical JAYA algorithm, the novel improved JAYA algorithm and their specific set of parameters; Sect. 3 explains the suggested framework for the generation of feasible test suits, Sect. 4 includes the extensive experimental set up Sect. 5 provides experimental results and discussions with the detailed statistical analysis. Lastly, the conclusions and prospective future directions are projected in Sect. 6.

2 Proposed Algorithm

The metaheuristic Jaya algorithm [16] was introduced by Venkata Rao [14], it's a very simple and parameter free algorithm that has been already used for numerous optimization problems in diverse domains of continuous space. In this paper an improved Jaya algorithm with improved exploration and convergence speed is proposed by adding an efficient mutation scheme the conventional JAYA algorithm. This improved Jaya algorithm was applied in one research work for automatic ear image enhancement of the ear biometric system [18]. It was noticed that the improved JAYA algorithm show better performance for image enhancement in comparison to other two metaheuristics i.e. PSO and Differential Evolution based image enhancement techniques. Therefore, this paper used the Improved JAYA algorithm as well as the Conventional JAYA algorithm and compared the performance of the respective algorithms with widely popular Differential evolution algorithm, in terms of test data generation and computational speed, for the first time in the field of object-oriented testing.

2.1 JAYA Algorithm

The metaheuristic JAYA algorithm is a parameter free optimization algorithm The principle of the algorithm is to obtain good solutions avoiding bad solutions. Iteration-wise updating each solution is mathematically expressed in [18] as follows:

$$X_{i,j}(g+1) = X_{i,j}(g) + r_{1,i,j} * \left(X_{i,best}(g) - \left|X_{i,j}(g)\right|\right) - r_{2,i,j} * \left(X_{i,worst}(g) - \left|X_{i,j}(g)\right|\right) \tag{1}$$

where $X_{i,j}(g)$ is the j^{th} parameter value for i^{th} solution at g iteration. $X_{i,best}(g)$ is the value of the best solution for i^{th} parameter at g^{th} iteration and $X_{i,worst}(g)$ is the value of the i^{th} parameter for the worst solution at the same g^{th} iteration. Two random numbers $r_{1,i,j}$are $r_{2,i,j}$ generated in the range of [0, 1] at iteration g, $X_{i,j}(g+1)$ hold the updated values of the j^{th} parameter for i^{th} candidate solution in $(g+1)$. The neighborhood positions are exploited and candidate solutions are continuously upgraded in subsequent iteration using Eq. (1) to lead the convergence towards global solution. The two random numbers $r_{1,i,j}$ and $r_{2,i,j}$ help in improving the searching capability of Jaya algorithm. Initially the candidate solutions are updated at current iteration n, then based on fitness values the best individual solutions are updated after comparing the current solution $X_j(g)$ and updated solution $X_j(g+1)$ for the next iteration $(g+1)$ as described in [22]:

$$X_j(g+1) = \begin{cases} X_j(g), \mathit{iff}\left(X_j(g)\right) > f\left(X_j(g+1)\right) \\ X_j(g+1), \mathit{Otherwise} \end{cases} \tag{2}$$

Thus the solutions of subsequent iteration are better than the corresponding solutions of current iteration. The modified fitness values of the candidate solutions are the inputs for next iteration. In this manner the algorithm always converges towards best solution.

2.2 Improved JAYA Algorithm

In order to improve convergence rate, a mutation operator has been introduced with Jaya algorithm and the proposed technique is known as an improved version of Jaya algorithm (IJA). It is revealed in [18], that the Differential evolution metaheuristic algorithm shows better performance than the Particle Swarm Optimization algorithm in robust performance and faster convergence towards global optima. Therefore the mutation operator of the Differential Evolution algorithm has been used in the JAYA algorithm to add diversification. In order to establish balance between the exploitation and exploration strategies an adaptive mutation operator is introduced. Mathematically the mutation operator is defined in the following Eq. (3),

$$X_j(g+1) = X_{r_1}(g) + F * \left(X_{r_2}(g) - X_{r_3}(g)\right) \tag{3}$$

here $j \in \{1, \ldots, K\}$, j^{th} candidate solution of the population of size K. Correspondingly, r_1, r_2, and r_3 are the indices of the candidate solutions $\{1, \ldots, K\}$. Here, F is the scaling factor used to avoid the population stagnation and to control the difference vector in the mutation operation, it is in the range of [0, 1]. The an adaptive scaling factor has been used here is described as in Eq. (4).

$$F = 0.8 + rand * ((G_{max} - g)/G_{max}) \tag{4}$$

where g and G_{max} are the current iteration and maximum number of iterations respectively. The rate of mutation in Eq. (3) can also be evaluated adaptively, and when in this manner a random real number, greater than the rate of mutation. is generated then the mutation operation will be performed.

$$rand \geq \left(1 - \frac{g}{G_{max}}\right) \tag{5}$$

The rate of mutation fluctuates between 1 to 0, through initial iteration to a maximum number of iterations.

3 Projected Framework

This work proposes a framework; Fig. 1 for testing object oriented programs using the UML state machine model of the triangle classification problem and a novel improved JAYA algorithm. Initially the UML state machine model; Fig. 2 is developed and then it is converted to state chart graph. After that the nodes and edges are assigned weights [1, 2]. Then the SCG graph is traversed using the DFS algorithm in order to find out the total path cost and the feasible paths. The fitness function of the feasible paths is the total path weight [1, 2]. The JAYA, improved JAYA and DE metaheuristic algorithms are applied to generate test suits, fulfilling the transition coverage criteria.

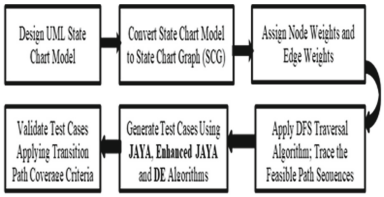


Fig. 1. Projected framework for model based testing

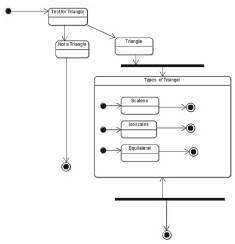


Fig. 2. The UML state machine model

4 Experiments

First of all, a UML state machine model was generated using ArgoUML, for the example problem, i.e., classification of triangles. After that metaheuristic JAYA, improved JAYA and DE algorithms, are used to generate test suits automatically using Matlab R2016b.

The fitness function of the problem is the total path weights of respective feasible paths, it is a maximization problem. The range of data variation is –10000 to 10000. To verify the exploitation and exploration capabilities of the metaheuristic algorithms, i.e. JAYA, Improved JAYA and DE a number of experiments were conducted with varying populations (10, 20, 30) and fixed generation (10), and then with varying generations (20, 30, 50) and fixed populations (10). After that again the different combination of generation, 20, 30 and population 50, 100 were taken to thoroughly test the time complexity and exploration capabilities of the algorithms.

Case study

Triangle classification problem is the benchmark problem in software testing domain [1, 2], specifically test data generation [3, 9]. The distinctive attribute of the problem is, it needs separate groups of test data to test first the triangle properties and then the types of triangles like scalene, equilateral and isosceles. [1]. This problem is selected as the case study to automatically generate test suites using Improved JAYA, DE and JAYA algorithms. This example problem has four feasible path sequences and six states (S1, S2, S3, S4, S5, S6). The statistical results after using JAYA, DE, and improved JAYA algorithms are depicted in Table 2 and Table 3. The test suits generated for respective path sequences using improved JAYA, algorithm are represented in Fig. 3 and Fig. 4 respectively. The Fig. 5 and Fig. 6 are showing the test suites generated by all the three algorithms i.e., JAYA, DE and Improved JAYA. The Table 2 and Table 3, show the detailed statistical analysis of the test suits generated using the proposed framework and metaheuristic algorithms. In these tables the minimum and maximum point outs the lower number of test cases and maximum number of test cases generation for the individual path. The maximum point outs the upper bound in test cases generation for a particular path. The table shows a minimum value to be zero when in at least one of the executions no data is available for covering a path, in the same way if the maximum value provided is zero then it signifies no data is generated at all for testing that path. Lastly the Table 4 shows the execution time of the three algorithms (Table 1).

Table 1. Test sequences of the triangle classification problem

Test sequence1	S1–S3	Not a triangle
Test sequence 2	S1–S2–S4	Scalene-triangle
Test sequence3	S1–S2–S5	Isosceles-triangle
Test sequence4	S1–S2–S6	Equilateral triangle

5 Results and Discussions

The aim of conducting the experiments with varying generations and populations was to figure out the exploration and exploitation capabilities of the algorithms. The detailed statistical analysis of the results like max, min, and average performance of the algorithms along with standard deviations were recorded in Table 2 and Table 3. The worst-case analysis of the algorithms shows that for path sequence 3, the critical path of the problem only improved JAYA achieved the best results, with large population size. The same trend is observed in average case analysis too. In most of the iterations the JAYA, improved JAYA, DE algorithms generated no data for path sequence2 and 3 whereas the Improved JAYA generated test data uniformly for every path sequence in case of maximum. The Fig. 3(a, b, c) depicts the test cases generated by improved JAYA with generation 10 and population variation 10, 20, 30, similarly Fig. 4(a, b, c) shows the test cases generated by improved JAYA with generation 20, 30,50 and population variation 10. In the next experiment the results of all the three algorithms for generating all four paths are recorded in Table 3 by varying the population to 10, 20, 50 and keeping the number of generations fixed at 10. The best, worst, and average case analysis of the results along with statistical analysis is provided in Table 3. Here it is observed that for path sequence 3, all of the three algorithms are providing almost zero results for best case in population 10 and 20, the DE is giving best result for only path sequence1 and JAYA for path sequence4 in all the three generations. The worst-case analysis shows that DE is giving best results for path sequence1, JAYA for path sequence 4, Improved JAYA for path sequence2 and 3. The average case analysis shows the same results. The Fig. 5(a, b, c) depicts the test cases generated by all the three algorithms at generation fixed at 10 and population size (30, 50,100). The Fig. 6(a, b, c) show the generated test cases, at generation fixed at 20 and population size (30, 50,100). When the results of Table 2, and Table 3 were compared, it was clear that the JAYA, DE and Improved JAYA, are not able to provide adequate number of test suits, when the population size and generations are small. The improved JAYA algorithm generated stable and uniform test suits, only when the population size is large.

Table 2. The statistics for test suite generation using Jaya, De, Improved Jaya with population size (30) & generation number (10, 20, 30)

Generation/ population		Model-based algorithm	Sequence1	Sequence2	Sequence3	sequence4
Gen = 10 Pop = 30	Minimum	JA	2	0	0	1
		DE	4	0	0	8
		EJA	1	1	0	1

(*continued*)

Table 2. (*continued*)

Generation/ population		Model-based algorithm	Sequence1	Sequence2	Sequence3	sequence4
	Maximum	JA	17	19	19	20
		DE	13	12	14	18
		EJA	16	17	17	20
	Mean	JA	8.62	8.91	2.36	10.11
		DE	8.81	7.76	1.76	7.76
		EJA	7.57	9.51	4.97	7.91
	Standard deviation	JA	2.6285	3.398	1.879	3.349
		DE	3.06	4.87	7.86	9.05
		EJA	3.104	3.206	2.623	3.338
Gen = 20 Pop = 30	Minimum	JA	0	0	0	3
		DE	0	3	0	4
		EJA	1	1	0	1
	Maximum	JA	17	18	7	20
		DE	13	15	9	16
		EJA	19	19	15	17
	Mean	JA	8.33	9.025	2.19	10.405
		DE	5.19	15.26	5.7	12.03
		EJA	6.33	10.2	5.395	7.075
	Standard deviation	JA	2.767	3.14	1.403	3.071
		DE	2.81	4.44	5.06	5.3
		EJA	2.841	2.953	2.566	2.991
Gen = 30 Pop = 30	Minimum	JA	0	1	0	1
		DE	1	0	0	4
		EJA	1	2	0	2
	Maximum	JA	19	21	18	17
		DE	18	12	13	19
		EJA	18	17	19	21
	Mean	JA	6.037	11.784	6.041	6.125
		DE	5.53	18.9	5.03	10.53
		EJA	6.99	10.75	5.896	6.352
	Standard deviation	JA	2.81	3.096	2.438	2.562
		DE	3.34	5.36	3.99	4.35
		EJA	2.534	2.076	2.904	2.072

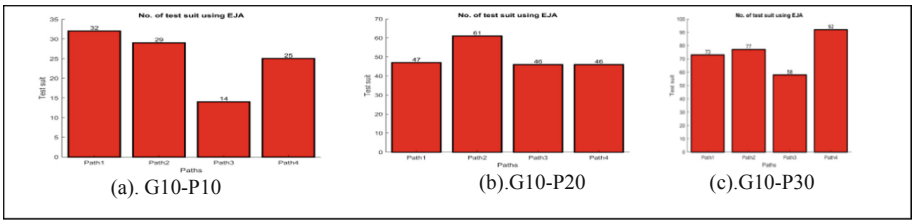
Table 3. The statistics for test suite generation using Jaya, De, Improved Jaya with population size (10, 20, 50) & generation number (10)

Generation/ population		Metaheuristic optimization algorithms	Sequence1	Sequence2	Sequence3	Sequence4
Gen = 10 Pop = 10	Minimum	JA	0	0	0	1
		DE	1	0	0	0
		EJA	1	0	0	1
	Maximum	JA	7	7	5	8
		DE	5	6	4	6
		EJA	7	7	9	6
	Mean	JA	2.4	3.7	2.3	4.3
		DE	2.3	1.8	1.5	2.7
		EJA	3.28	2.68	2.5	3.16
	Standard deviation	JA	1.56	1.67	1.04	1.67
		DE	1.45	2.28	2.43	3.84
		EJA	1.06	1.28	1.77	1.29
Gen = 10 Pop = 20	Minimum	JA	2	1	0	2
		DE	11	2	0	0
		EJA	1	1	0	2
	Maximum	JA	11	11	8	11
		DE	5	18	0	0
		EJA	12	12	10	11
	Mean	JA	5.74	5.68	2.34	6.24
		DE	3.25	1.57	2.7	4.5
		EJA	5.16	6.72	2.52	5.6
	Standard deviation	JA	2.036	2.42	1.47	2.43
		DE	3.07	14.05	2.71	3.4
		EJA	2.67	2.138	1.665	2.014
Gen = 10 Pop = 50	Minimum	JA	7	2	0	6
		DE	5	3	0	6
		EJA	6	3	1	4
	Maximum	JA	26	25	12	31

(*continued*)

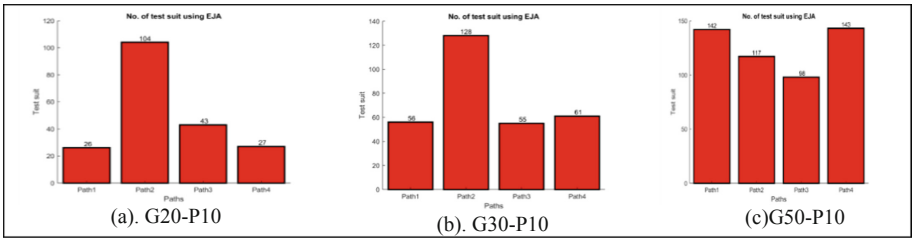
Table 3. (*continued*)

Generation/ population		Metaheuristic optimization algorithms	Sequence1	Sequence2	Sequence3	Sequence4
		DE	25	18	17	28
		EJA	22	23	21	30
	Mean	JA	12.52	14.28	3	18.2
		DE	11.5	13.8	10.2	14.9
		EJA	13.5	15.2	16.82	14.48
	Standard deviation	JA	4.33	5.55	2.26	4.24
		DE	2.94	4.45	4.46	6.29
		EJA	4.3	4.85	4.32	4.4



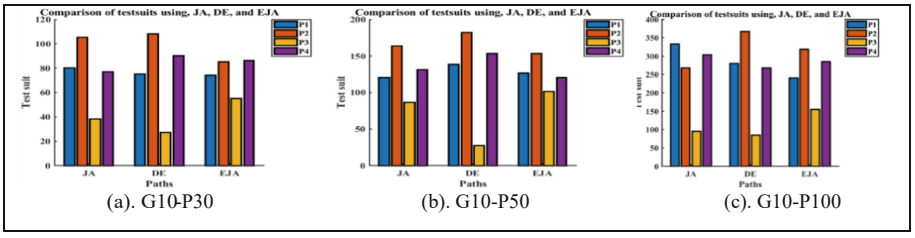
(a). G10-P10 (b).G10-P20 (c).G10-P30

Fig. 3. Test suites using Improved Jaya (A, B, C), with generations (10) and population size 10, 20, 30



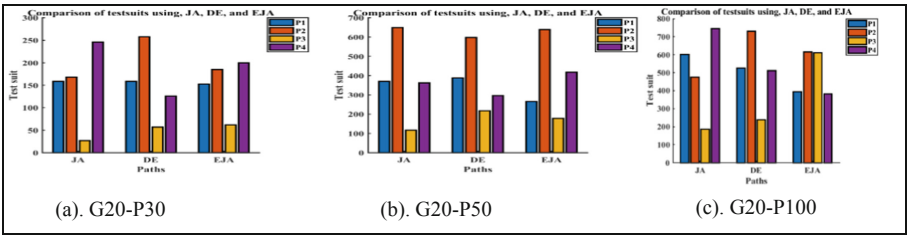
(a). G20-P10 (b). G30-P10 (c)G50-P10

Fig. 4. Test suite using Improved Jaya (A, B, C), with fixed population 10 and variations in generations 20, 30, 50



(a). G10-P30 (b). G10-P50 (c). G10-P100

Fig. 5. Test suites generation using Jaya, De and Improved Jaya (A, B, C), with generations **10** and population size **30, 50, 100**



(a). G20-P30 (b). G20-P50 (c). G20-P100

Fig. 6. Test suite generation using Jaya, De and Improved Jaya (A, B, C), with fixed number of generations (20) and variations in population size (30, 50, 100)

Table 4. Average run time (in seconds) for generation of test suits based on JAYA, DE, and Improved JAYA with generation 10 and population size 50.

JAYA	DE	Improved JAYA
0.004	0.025	0.018

6 Conclusions

The testing of object-oriented programs, particularly the test suite generation from design artifacts is a very difficult task. This work provided a novel Improved JAYA algorithm-based framework to generate optimized test suits. The optimal test suites were generated using the proposed framework and UML behavioral model. The proposed framework efficiently generated uniform test suits for all the feasible paths. The obtained simulation results ensured the efficiency of the improved JAYA algorithm over the performances of the individual JAYA and DE algorithm in terms of the exploration and exploitation capabilities. The framework can be further improved by using the hybrid version of this metaheuristic JAYA algorithm along with different UML diagrams.

References

1. Panda, M., Dash, S.: Test-case generation for model-based testing of object-oriented programs. In: Jena, A.K., Das, H., Mohapatra, D.P. (eds.) ICDCIT 2019. SBPR, pp. 53–77. Springer, Singapore (2020). https://doi.org/10.1007/978-981-15-2455-4_3
2. Panda, M., Dash, S.: Test suit generation for object oriented programs: a hybrid firefly and differential evolution approach. IEEE Access **8**, 179167–179188 (2020)
3. Shirole, M., Kumar, R.: UML behavioral model-based test case generation: a survey. ACM SIGSOFT Softw. Eng. Notes **4**(38), 1–13 (2013)
4. Utting, M., Pretschner, A., Legeard, B.: Taxonomy of model-based testing approaches. Softw. Test. Verif. Reliab. **5**(22), 297–312 (2012)
5. Saeed, A.S., AbHamid, H., Mustafa, M.B.: The experimental applications of search-based techniques for model-based testing: taxonomy and systematic literature review. J. Appl. Soft Comput. **49**, 1094–1117 (2016)
6. Harman, M., Jones, B.F.: The seminal workshop: reformulating software engineering as a metaheuristic search problem. ACM SIGSOFT Softw. Eng. Notes **6**(26), 62–66 (2001)

7. Harman, M., McMinn, P.: A theoretical and empirical study of search-based testing: local, global and hybrid search. IEEE Trans. Softw. Eng. **2**(36), 226–247 (2010)
8. Price, K.V.: Differential evolution. In: Zelinka, I., Snášel, V., Abraham, A. (eds.) Handbook of Optimization. Intelligent Systems Reference Library, vol. 38. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-30504-7_8
9. Panda, M., Dash, S.: Automatic test data generation using bio-inspired algorithms: a travelogue. In: Handbook of Research on Modeling, Analysis and Application of Nature-Inspired Metaheuristic Algorithms, pp.140–159. IGI Global (2018)
10. Samual, P., Mall, R., Bothra, A.K.: Automatic test case generation using unified modeling language (UML) state diagrams. IET J. Softw. **2**(2), 79–93 (2008)
11. Molina, D., et al.: Comprehensive Taxonomies of Nature and Bio-inspired Optimization: Inspiration versus Algorithmic Behavior, Critical Analysis and Recommendations. arxivpreprint arXiv:2002.08136 (2020)
12. Rao, R.V.: A self-adaptive multi-population based Jaya algorithm for engineering optimization. Swarm Evol. Comput. **37**, 1–26 (2017). https://doi.org/10.1016/j.swevo.2017.04.008
13. Wang, S.H., et al.: Intelligent facial emotion recognition based on stationary wavelet entropy and Jaya algorithm. Neurocomputing **272**, 668–676 (2017). https://doi.org/10.1016/j.neucom.2017.08.015
14. Rao, R.V., et al.: Dimensional optimization of a micro-channel heat sink using Jaya Algorithm. J. Appl. Therm. Eng. **103**, 572–582 (2016). ISSN: 1359-4311
15. Rao, R.V., et al.: Economic optimization of shell-and-tube heat exchanger using Jaya algorithm with maintenance consideration. Appl. Therm. Eng. **116**, 473–487 (2017)
16. Rao, R.: Jaya: a simple and new optimization algorithm for solving constrained and unconstrained optimization problems. Int. J. Ind. Eng. Comput. **7**(1), 19–34 (2016)
17. Cohen, B.: The Maturation of Search-Based Software Testing: Successes and Challenges. In: 12th International workshop on SBST (2019). ISBN: 978-1-7281-2233-5
18. Sarangi, P.P., et al.: An evaluation of ear biometric system based on improved Jaya algorithm and SURF descriptors. J. Evol. Intell. **13**, 443–461 (2019)