Research Article

Test scheduling for system on chip using modified firefly and modified ABC algorithms

Gokul Chandrasekaran[1] · Sakthivel Periyasamy[2] · P. R. Karthikeyan[2]

Abstract
The system-on-chip (SoC) is an integration of millions of electronic components, there is always a chance for faults to occur due to manufacturing defects. In order to solve this problem, it is essential to test the manufactured chips. The time spent on testing increases the testing cost which reflects on the cost of the chip. While testing the SoC, core accessibility and testing time are the main issues to be considered. In order to reduce the testing time, test scheduling has to be performed in an effective manner. In this article ACO, Modified ACO, ABC, Modified ABC, Firefly and Modified Firefly test scheduling algorithms were tested on two SoC benchmark circuits. Experimental results show that the Modified ABC algorithm performs better than the other algorithms used in test scheduling. When compared with ACO, Modified ACO, ABC, Firefly and Modified Firefly algorithms, the Modified ABC algorithm's testing time has been reduced by 82%, 69%, 25%, 43% and 48% for d695 SoC and 80%, 73%, 20%, 41% and 47% for p22810 SoC benchmark circuits respectively.

Keywords Integrated circuits · System-on-chip · Ant colony optimization · Firefly algorithm artificial bee colony optimization

1 Introduction

Semiconductor integrated circuits (ICs) are the basis of electronic products in the modern world. ICs are embedded in most of the systems and products. Several ICs introduced consequent to the growth in semiconductor technology are called system-on-chip (SoC). The SoC is an integration of IC that contains many transistors. The main issue in SoC testing is the IC complexity. To solve this issue the reusable core IC needs to be designed and verified. The cores may be manufactured internally or bought externally. System integrator brings these cores into the system. When the system becomes more complicated, faults need to be tested. This complication also increases the test cost. Test access mechanism (TAM) is employed to test cores individually [1, 2].

The SoC test model is used to test individual cores and inter connections. In the SoC model, the major components are a wrapper, TAM and test scheduling. The thin shell encircling the core called wrapper acts as an interface between the core and TAM. Test vectors are provided through TAM wires to the wrapper [3]. Automatic test equipment (ATE) stores the test vectors and through TAM wires, vectors are provided to the SoC. Interconnection is tested in external mode and the core is tested in internal mode. There are three modes in which testing of the SoC is done. They are bypass, soft and lower power modes. Scheduling of test is classified as partitioned, non-partitioned and pre-emptive testing. While cores are tested, the obtained response is compared with the expected response and the difference indicates the error. During the testing of interconnects, test stimuli are generated and the response is observed. All the cores need to be tested [4, 5].

✉ Gokul Chandrasekaran, gokul@velalarengg.ac.in; Sakthivel Periyasamy, psv@annauniv.edu; P. R. Karthikeyan, karthikeyanest@gmail.com | [1]Velalar College of Engineering and Technology, Affiliated to Anna University, Chennai, Erode, India. [2]Department of Electronics and Communication Engineering, Anna University, Chennai, India.

SN Applied Sciences (2019) 1:1079 | https://doi.org/10.1007/s42452-019-1116-x

Received: 4 May 2019 / Accepted: 17 August 2019 / Published online: 24 August 2019

The benchmark circuits d695 and p22810 are considered and the various algorithms are used for the benchmark circuits. The result is shown with regard to the test time which determines the test cost. The Modified Firefly Algorithm minimizes the test price by minimizing the total test application time. Hence, it is a better optimization technique than the other proposed algorithms.

An objective function is used to attain the tuning parameters of the system model to test under different load settings of d695 and p22810 SoC benchmark circuit. The purpose of algorithm for optimization is to reduce the test time which is the objective function stated in the following Eq. (1).

$$T(W_i) = (1 + \max(s_i + s_0) \cdot t_{pi} + \min(s_i, s_0)) \quad (1)$$

where S_i and S_o represent the input and the output scan chain lengths and t_{pi} represents the test pattern for core i of the benchmark circuit for optimization.

2 Literature review

Earlier works mainly focused on optimization of the TAM and wrapper design for reducing the test time for core-based systems [6] [7]. The three scheduling techniques are partitioned, non-partitioned, and preemptive techniques. All the cores must be completed at the end of the testing. Internal testing and external testing, Built in self test (BIST) and ATE are used. The SoC testing problem consists of core wrapper design, test scheduling and test access mechanism design. The test shell acts as an interface between the core and the host containing three types of I/O terminals, namely function I/O, test rail I/O, and direct test I/O. In the test wrapper approach, the bypass feature is not allowed which means only one core is served at a time [8, 9]. For 2-D bin packing problem, Simulated Annealing Algorithm minimizes the testing time [10].

In non-partitioned test scheduling until all the tests in a section are completed no new test can be started which increases the test time. In the partitioned technique, the test time is reduced by allowing the test to be scheduled as soon as possible. Pre-emptive test scheduling minimizes idle time. In order to reduce the testing time of the core-based systems earlier works mainly focused on efficient wrapper design and TAM optimization design. Wrapper design and the related algorithm are based on access requirements of core terminals like functional access.

In open shop scheduling, the shop consists of m processors, J jobs each with m tasks of length ≥ 0. For each task, no job is processed on more than a single processor. The objective of open shop scheduling is the reduction of finish time of individual processor schedules. Simulated annealing (SA) Algorithm was used to resolve the bin packing problem of two dimensions by minimizing the TAT. The wrapper design method was used to generate a sequence of rectangles using Genetic Algorithm formulation. From the set of rectangles generated for the core one rectangle has to be selected to reduce total testing time [11].

In hierarchy aware test planning method used for the optimization of TAM, the issue of hierarchical SoCs is addressed. Two practical scenarios emerge: (1) For the child cores TAM architecture and Wrappers are fixed (hard) and for a parent, cores are determined (soft). (2) TAM and Wrappers for both the parent and the child cores are soft. A multilevel TAM architecture is proposed which describes the use of flattened SoCs for multilevel TAM optimization. The new wrapper architecture having two disjoint test modes for testing parent cores and child cores is proposed. particle swarm optimization (PSO) is an optimization algorithm dependent on the flocking behavior of the bird developed by Kennedy [12]. Real ant behavior leads to Ant Colony Optimization depending on the mechanism followed by an ant in search of the shortest route for food.

3 Proposed work

3.1 Ant colony optimization

Ant colony optimization (ACO) is [13, 14], an approach based on ant population to solve computational problems. This is based on the social nature of ants for identifying the best route to the food source from the nest by indirect communication among ants using Pheromone, a chemical.

It is a metaheuristic optimization technique. Ants leave pheromone behind while moving so that the other ants smell this pheromone and follow it. In ACO artificial ants develop newer solutions using the mixture of heuristic information and artificial pheromone trail. ACO has been shown to be efficient in solving many problems.

3.2 Modified ant colony optimization

In the existing techniques even if the number of cores increases, ants' count will remain constant. On each iteration, for the same ants' count when the number of test cores decreases, the testing time increases which leads to inefficient output. This problem is solved in a modified ACO [15–17] technique. In the proposed work, whenever the number of cores decreases, ants' count will also get reduced. This leads to a reduction in the testing time of the system. Hence more efficient output can be obtained.

3.3 Artificial bee colony (ABC) algorithm

A meta-heuristic technique based on honey bee intelligent behavior to solve numerical problems is called Artificial bee colony algorithm [18–20]. The steps involved in the ABC algorithm is shown as a flowchart in Fig. 1.

3.4 Modified artificial bee colony algorithm

For ABC technique among common control parameters like population size, only one control parameter is used. But when the ABC technique is used for a composite function, the convergence rate is poor. In this process perturbation frequency is a parameter that affects new solutions. The ABC technique can be modified by controlling the perturbation frequency. This is referred to as the modified artificial bee colony (MABC) algorithm [21–23]. In general, ABC algorithm frequency is constant. When a newer solution is produced through only one parameter of parent solution, the issue can be overcome by MABC algorithm. The modification is done by adding a control parameter additionally.

By this modification, for every parameter X_{ij}, a random number is generated and if it is lower than the modification rate, random variable X_{ij} is modified. Another change in the ABC algorithm is dependent on the variance operator ratio. In ABC, the current solution is added with random perturbation at local minima to obtain a new solution. This magnitude of perturbation is controlled by the scaling factor of the new control parameter.

Step 1 Randomly generate food sources S_i.

Step 2 Food sources are assigned to the employed bee. For every food source fitness value f (S_i) is evaluated.

Step 3 Initialize M = 0 and $L_1 = 0$, $L_2 = 0 \ldots L_N = 0$

M = Number of times of repeating an entire foraging process.

L_i = Number of times of applying a neighbor operator to food source 'i' where i = 1… N.

Step 4 Repeat foraging process

For Employed Bee Phase

Each food source is employed by a neighborhood operator: $S_i \rightarrow S$.

If f (S) > f (S_i), S_i is replaced by S and Li = 0. Or else Li = Li + 1.

For Onlooker Bee Phase

Based on fitness values onlooker bee selects a source of food by means of the roulette wheel selection method.

If f (S) > f (S_i), select S_j where Lj is the maximum of all food sources.

If f (S) > f (S_j), S_j is replaced by S and Lj = 0. Otherwise Li = Li + 1.

For Scout Bee Phase

If L_i = Limit for each food source, applied by a neighborhood operator: S_i is replaced by S. Where M = M + 1.

Step 5 Stop the foraging process where M reaches Maximum Cycle.

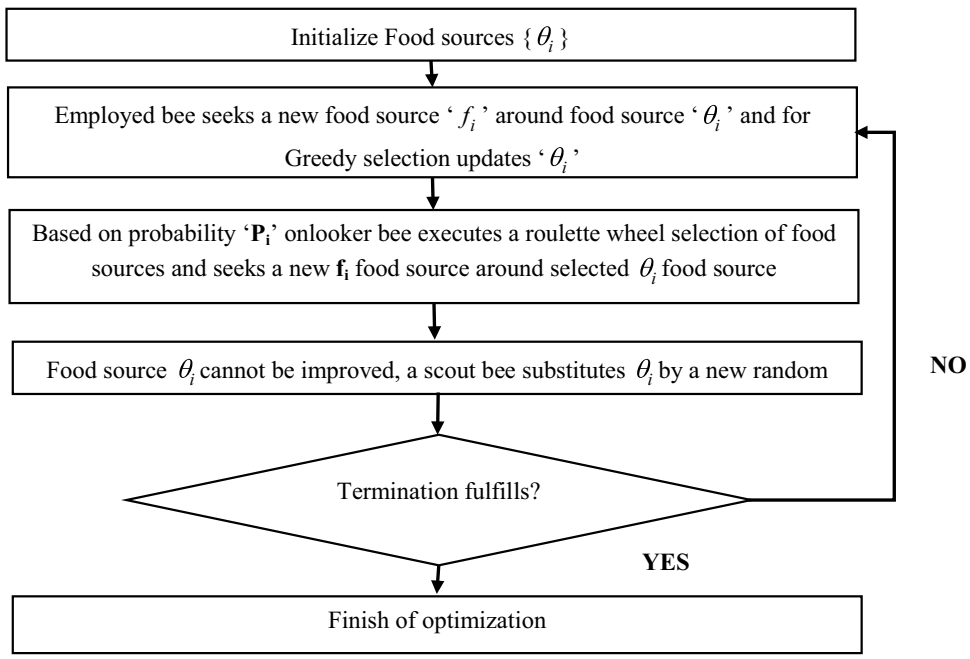

Fig. 1 Flowchart for ABC algorithm

3.5 Firefly algorithm

A metaheuristic technique based on the flashing behavior of the firefly was proposed by Yang and came to be called Firefly Algorithm [24, 25]. This works on the attractiveness between the fireflies while the mating process is based on the firefly brightness. Figure 2 demonstrates the Firefly Algorithm flowchart giving the steps in which it proceeds.

3.6 Modified firefly algorithm

The Modified Firefly Algorithm [26–28] overcomes the problem of the Firefly Algorithm. It reduces the randomness of the technique and enhances the movement of Fireflies. Figure 3 illustrates the flowchart of Modified Firefly Algorithm and the steps by which it proceeds. In modified firefly Algorithm, randomization parameter α varies between α_0 and α_∞, where α_0 and α_∞ are initial and final values during iteration. Exploitation and exploration abilities are kept under balance. If α is larger, the convergence will be better. Distance function r_i and movement of ith firefly are shown by the Eqs. (2) and (3).

$$r_{i,best} = \sqrt{(x_i - x_{gbest})^2 + (y_i - y_{gbest})^2} \tag{2}$$

$$xi = xi + \beta_0 e^{-\gamma r_{i,j}^2}(xj - xi) + \beta_0 e^{-\gamma r_{i,best}^2}(x_{gbest} - x_i) + \alpha\varepsilon + \lambda\varepsilon(x_i - gbest) \tag{3}$$

where $\varepsilon = rand - {}^1/_2$, gbest = global best. If any local best solution does not occur in the neighborhood, ith firefly gets attracted to the best solution. Modified Firefly Algorithm reduces the randomness to minimize the probability into local optima. Hence convergence is achieved quickly and fireflies move to global optima.

Figure 3 gives the flowchart of Modified Firefly Algorithm, giving the steps by which it proceeds.

4 Results and discussion

The results are presented for two international test conference (ITC'02) benchmark circuits d695 and p22810.

4.1 d695 SoC Benchmark

Figures 4, 5, 6, 7 and 8 show the initialization of cores for d695 SoC benchmark circuit using Ant Colony, Modified Ant Colony Algorithm, Artificial Bee Colony Algorithm,

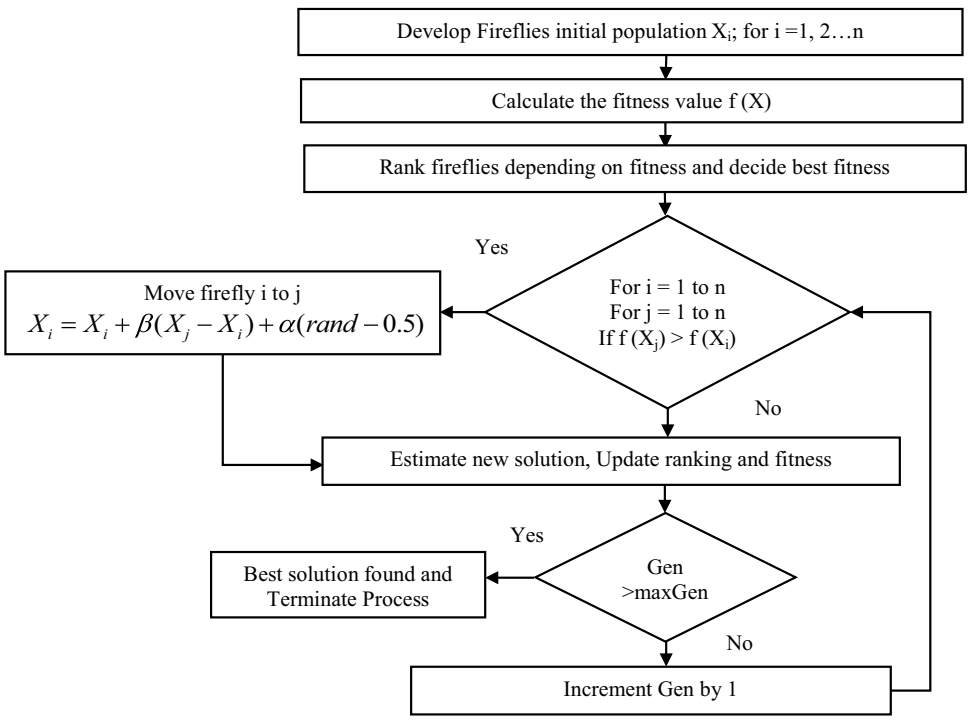

Fig. 2 Flowchart of firefly algorithm

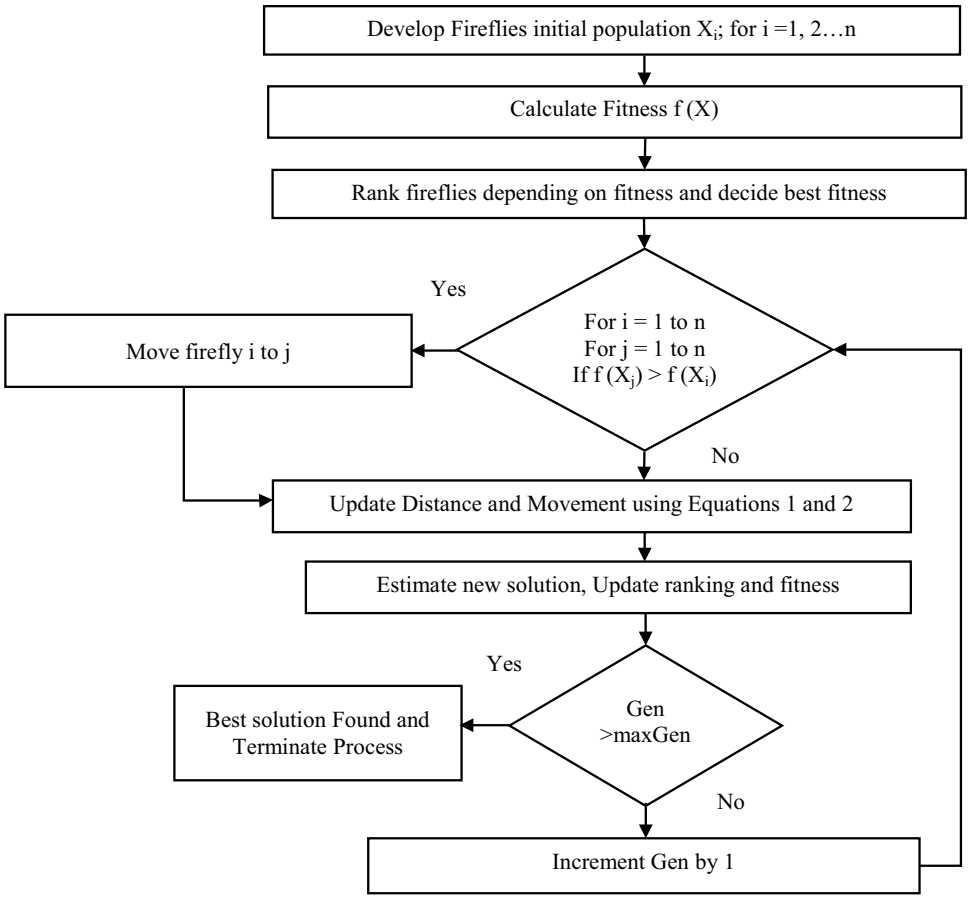

Fig. 3 Flowchart of modified firefly algorithm

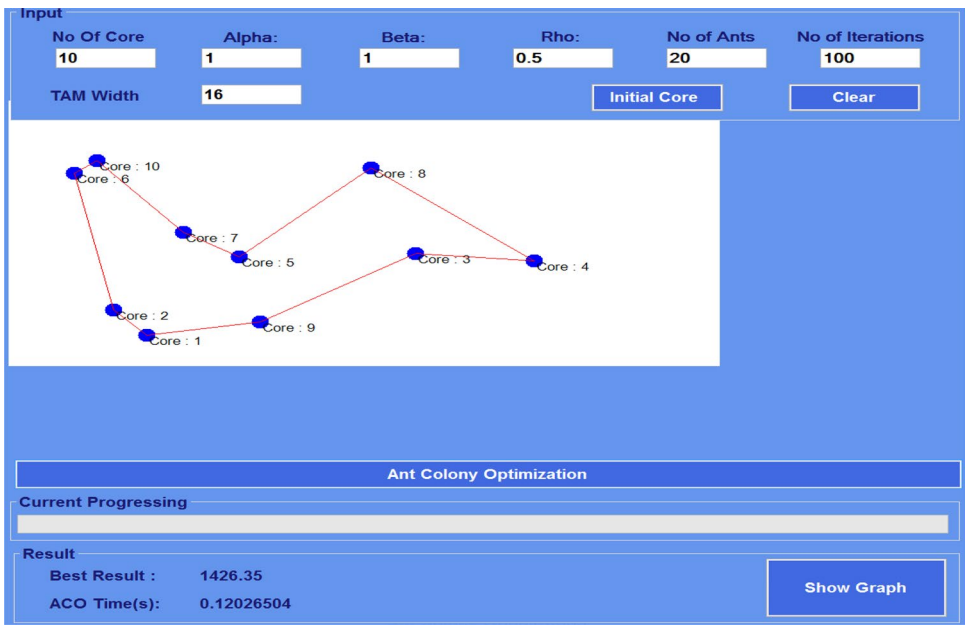

Fig. 4 Core initialization of ACO optimization—d695 SoC

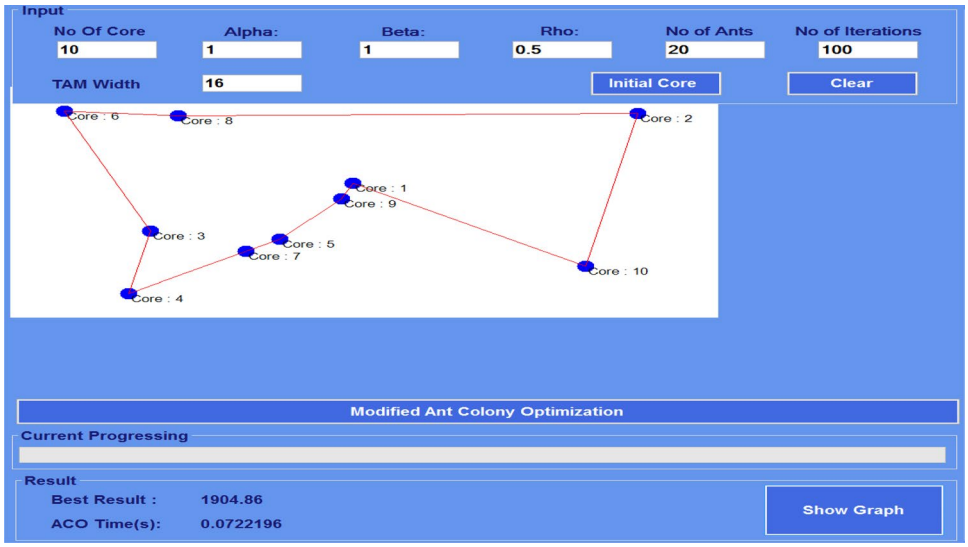

Fig. 5 Core initialization of modified ACO optimization—d695 SoC

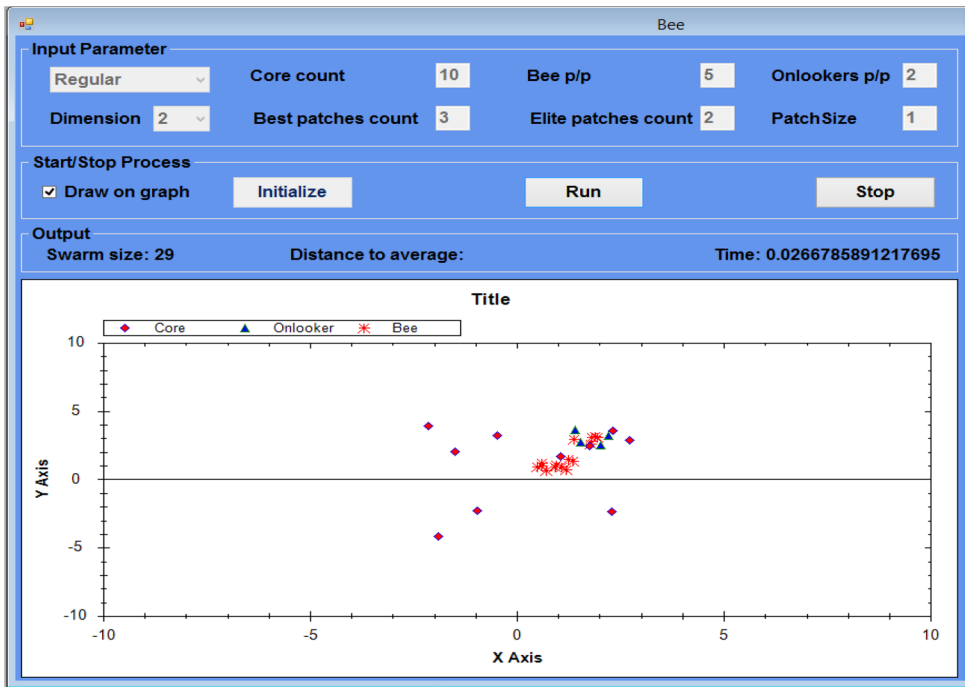

Fig. 6 Core initialization of ABC optimization—d695 SoC

Modified Artificial Bee Colony Algorithm and Firefly Algorithm respectively. Numerous iteration parameters are taken as input. The testing time and best result point are computed.

It can be understood from Fig. 4 that the parameters alpha, beta and rho are started by the value 1, 1 and 0.5 respectively. The best result point is found as 1426.35 and the testing time is achieved as 0.120265. Similarly, for other algorithms the best result point and testing time are calculated by considering various input parameters.

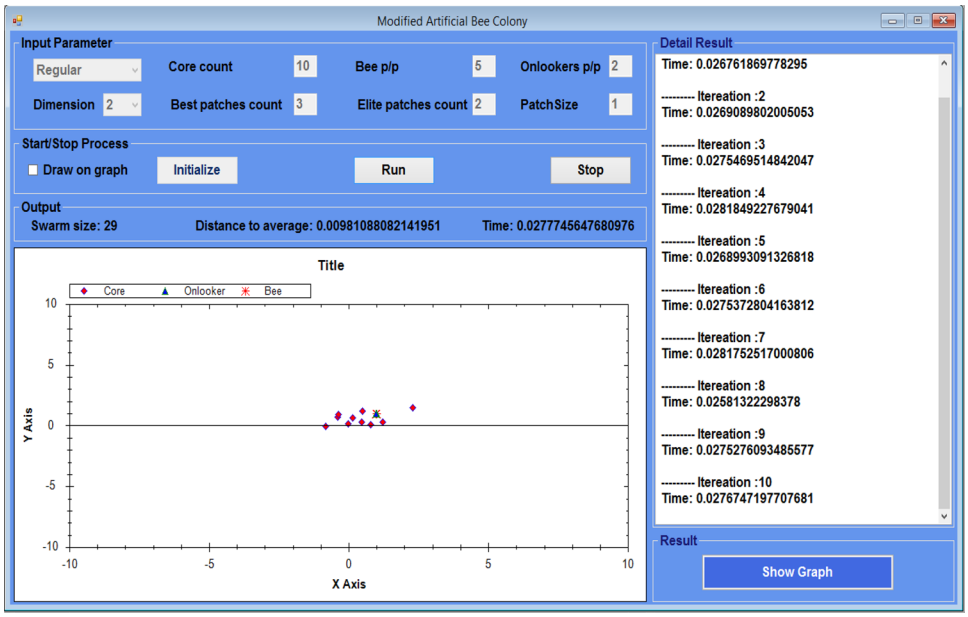

Fig. 7 Core initialization of modified ABC optimization—d695 SoC

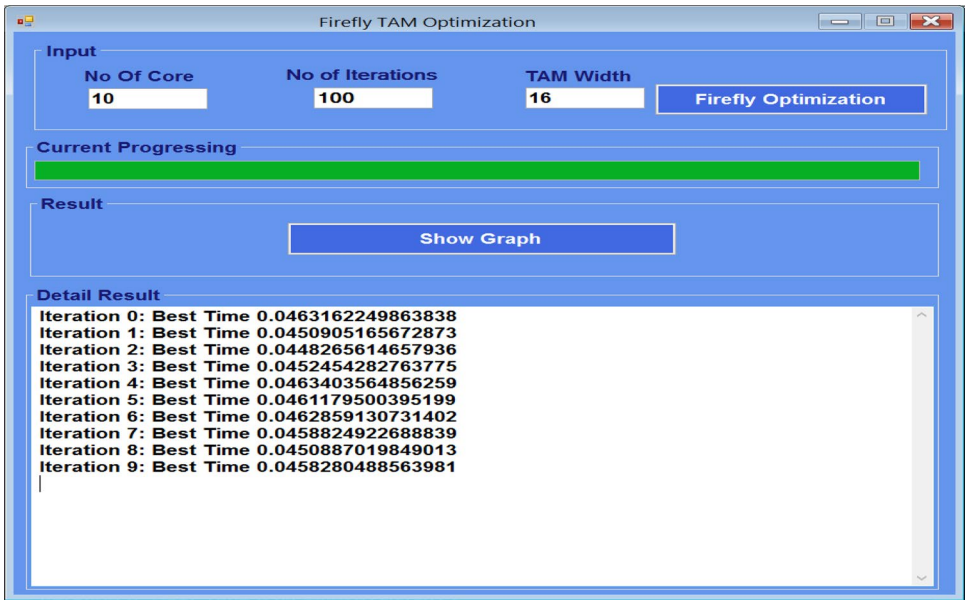

Fig. 8 Core initialization of firefly algorithm—d695 SoC

4.2 p22810 SoC Benchmark

Figures 9,10,11,12 and 13 show the initialization of cores for p22810 SoC benchmark circuit using ACO, Modified ACO, ABC, Modified ABC and Firefly algorithms respectively. Numerous iteration parameters are taken as input. The testing time and the best result point are computed.

Tables 1 and 2 show the comparison of testing time obtained from ACO, Modified ACO, ABC, Modified ABC, Firefly and Modified Firefly algorithms for d695 and

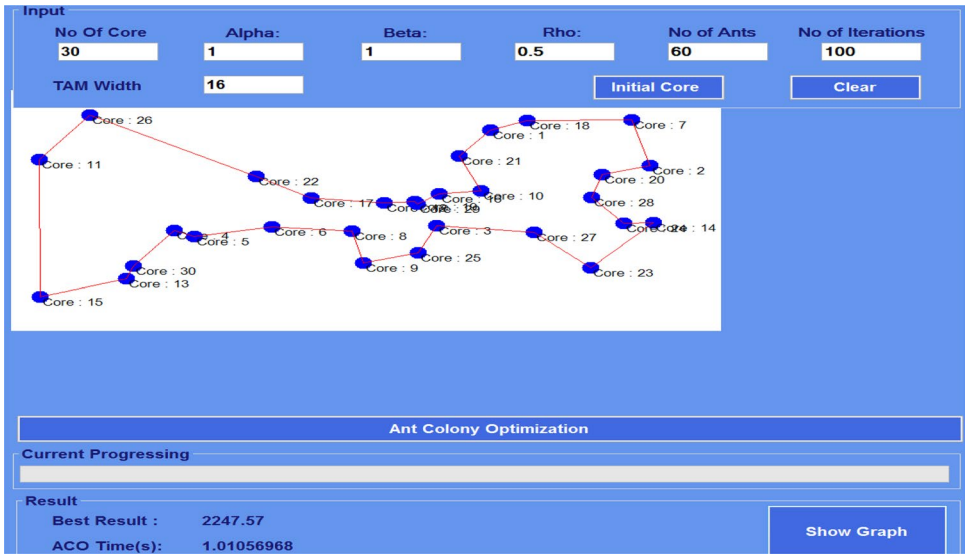

Fig. 9 Core initialization of ACO optimization—p22810 SoC

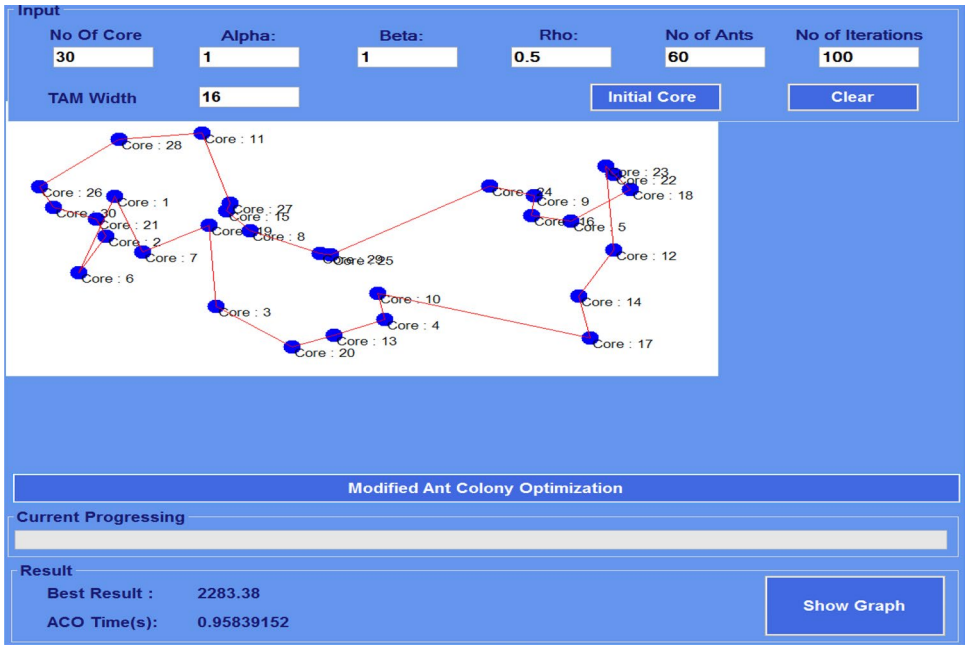

Fig. 10 Core initialization of modified ACO optimization—p22810 SoC

p22810 SoC benchmark circuits. It can be easily inferred that the Modified ABC Algorithm gives optimal test time when compared with the other proposed algorithms.

Figures 14 and 15 show the test time comparison for several algorithms proposed in a graphical format for d695 and p22810 benchmark circuits. The obtained graph explains that the various proposed algorithms show a specific quantity of reduction in the testing time compared with the ACO algorithm.

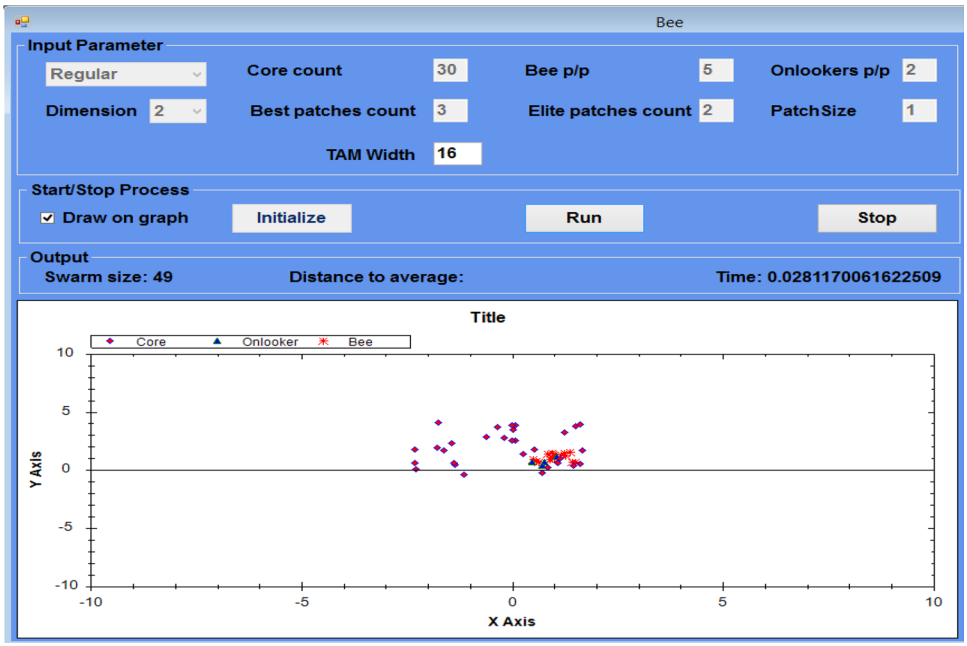

Fig. 11 Core initialization of ABC optimization—p22810 SoC

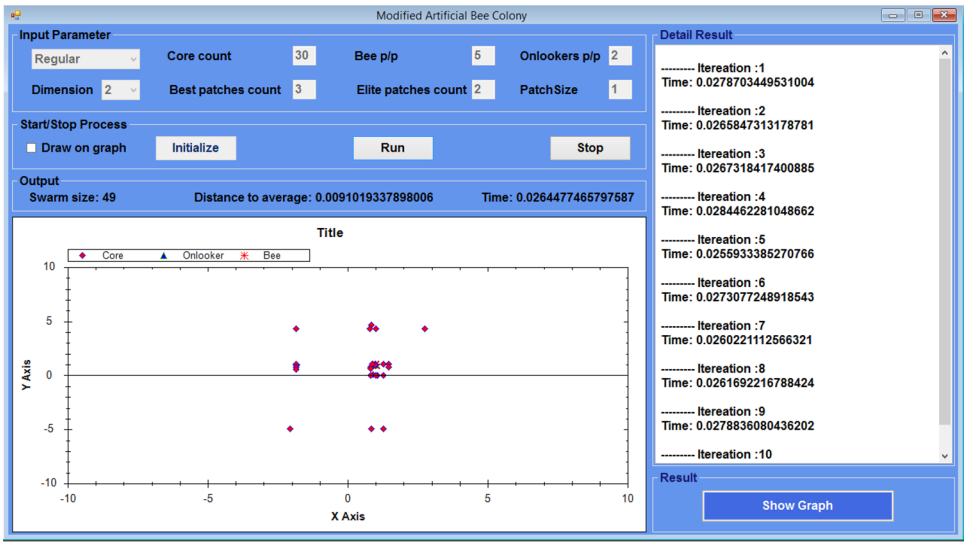

Fig. 12 Core initialization of modified ABC optimization—p22810 SoC

5 Conclusion and future work

In this paper, the optimization of SoC has been performed by reducing the testing time using various algorithms for d695 and p22810 benchmark circuits. The various TAM widths have been taken into consideration. The final obtained result has shown that the Modified ABC Algorithm is more efficient in reducing the test time for SoC benchmarks. When compared with ACO, MACO, ABC, Firefly and Modified Firefly Algorithms the testing time of Modified ABC Algorithm has been reduced to 82%, 69%, 25%, 43% and 48% for d695 SoC and 80%, 73%,

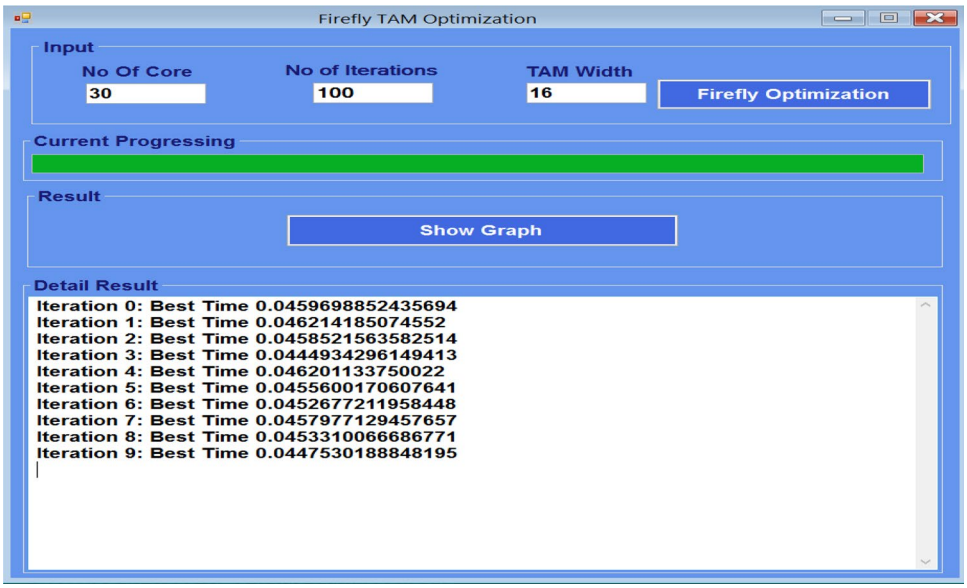

Fig. 13 Core initialization of firefly algorithm—p22810 SoC

Table 1 Test time comparison of different TAM width for d695 SoC

TAM Width "W"	ACO	Modified ACO	ABC	Modified ABC	Firefly	Modified firefly
64	0.115898931	0.035986249	0.026773415	0.021173415	0.0330091	0.0284903
56	0.115730567	0.03665028	0.026808083	0.021208182	0.0330489	0.0288502
48	0.116411767	0.036791811	0.026919859	0.021319759	0.0331303	0.0287198
40	0.116405873	0.038201633	0.026976328	0.021376328	0.0331698	0.0290803
32	0.116222444	0.03609952	0.027006077	0.021416077	0.0332729	0.0287055
24	0.116115549	0.037655745	0.027072338	0.021472338	0.0333701	0.0285497
16	0.124666684	0.037780764	0.027182794	0.021582793	0.0337692	0.0297982

Table 2 Test time comparison of different TAM width for p22810 SoC

TAM width "W"	ACO	Modified ACO	ABC	Modified ABC	Firefly	Modified firefly
64	0.435908149	0.36743764	0.026297126	0.020897126	0.0450991	0.0412982
56	0.436873964	0.36687257	0.026399611	0.021009611	0.0451194	0.0410196
48	0.439020897	0.36792353	0.026530394	0.021110394	0.0452393	0.0409398
40	0.442404672	0.36914785	0.026634811	0.021264811	0.0452993	0.0408891
32	0.44337699	0.36939797	0.026776916	0.021426916	0.0455389	0.0418761
24	0.45058744	0.37184387	0.026886905	0.021566905	0.0458695	0.0415116
16	0.47002504	0.39647125	0.026969701	0.021539701	0.0460583	0.0420491

20%, 41% and 47% for p22810 SoC benchmark circuit respectively. Hence the objective of reducing the testing cost is achieved by minimizing the SoC testing time using the proposed algorithm.

In future, this work can be extended to mixed-signal VLSI chips contains digital circuits, and analog circuits. As more complex, mixed signal SoC designs continue to stress

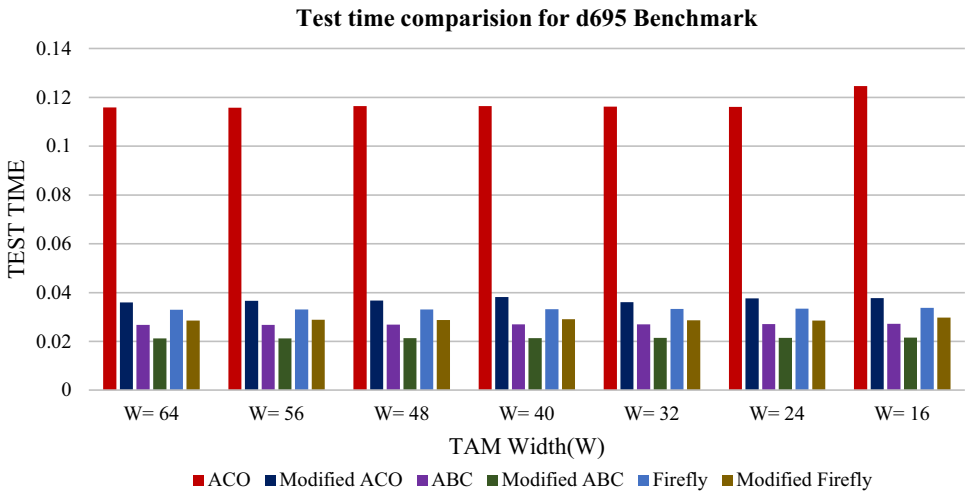

Fig. 14 Graphical depiction of testing time using ACO, MACO, ABC, MABC, firefly and modified firefly for d695 SoC

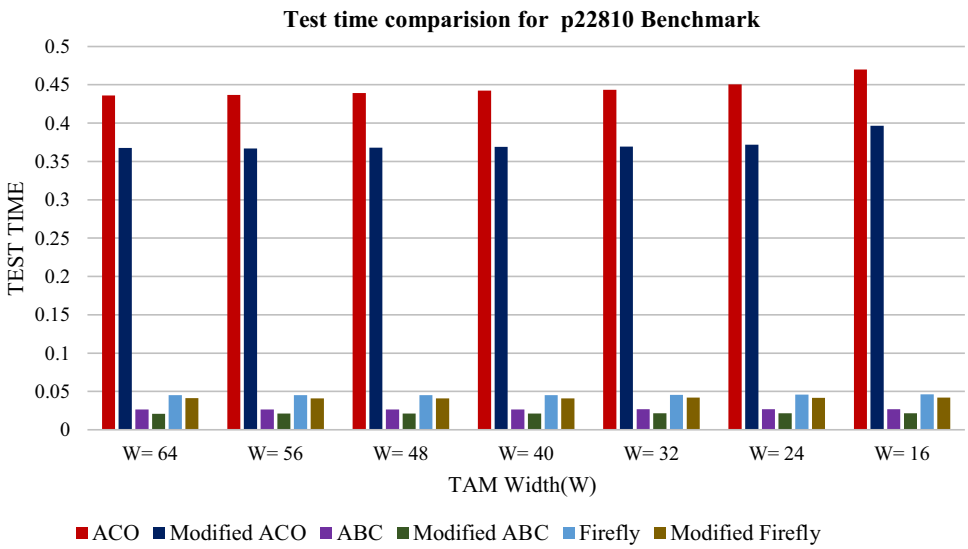

Fig. 15 Graphical depiction of testing time using ACO, MACO, ABC, MABC, firefly and modified firefly for p22810 SoC

on verification methodologies and schedules, designers need new approaches for solving today's test challenges.

Acknowledgements The authors extend their sincere thanks to Centre for Research of Anna University for providing us a platform to take over our research work.

Compliance with ethical standards

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

References

1. Iyengar V, Chakrabarty K, Marinissen EJ (2003) Efficient test access mechanism optimization for system-on-chip. IEEE Trans Comput Aided Des Integr Circuits Syst 22(5):635–643. https://doi.org/10.1109/TCAD.2003.810737
2. Larsson E, Fujiwara H (2006) System-on-chip test scheduling with reconfigurable core wrappers. IEEE Trans VLSI Syst 14(3):305–309
3. Koranne S (2003) Design of reconfigurable access wrappers for embedded core based SoC test. IEEE Trans VLSI Syst 11(5):955–960
4. Larsson E, Peng Z, Chakrabarty K (2002) An integrated framework for the design and optimization of SOC test solutions. J Electron Test 21:21–36. https://doi.org/10.1007/978-1-4757-6527-4_2
5. Marrouche W, Farah R, Harmanani HM (2018) A strength pareto evolutionary algorithm for optimizing system-on-chip test schedules. Int J Comput Int Syst 17(02):1850010
6. Iyengar V, Chakrabarty K, Marinissen EJ (2002) Test wrapper and test access mechanism co-optimization for system-on-chip. J Electron Test 18(2):213–230
7. Başak ME, Kuntman A, Kuntman HH (2014) MOSFET Spice parameter extraction by modified genetic algorithm. Inf MIDEM 44(2):142–151
8. Novak F (2001) Testability issues of system-on-chip design. Inf MIDEM 2:84–87

9. Iyengar V, Chakrabarty K (2002) System-on-a-chip test scheduling with precedence relationships, preemption, and power constraints. IEEE Trans Comput Aided Des Integr Circuits Syst 21(9):1088–1094. https://doi.org/10.1109/TCAD.2002.801102
10. Olenšek J, Puhan J, Bűrmen Á, Tomažič S, Tuma T (2006) Optimization of integrated circuits by means of simulated annealing. Informacije MIDEM 36:79–84. http://www.midem-drustvo.si/Journal%20papers/MIDEM_36(2006)2p79.pdf
11. Srinivas N, Deb K (1994) Muiltiobjective optimization using nondominated sorting in genetic algorithms. Evol Comput 2(3):221–248
12. Kennedy J (2010) Particle swarm optimization. Encyclopedia of machine learning. Springer, Heidelberg, pp 760–766
13. Mahi M, Baykan ÖK, Kodaz H (2015) A new hybrid method based on particle swarm optimization, ant colony optimization and 3-opt algorithms for traveling salesman problem. Appl Soft Comput 30:484–490
14. Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern B 26(1):29–41
15. Wang Z, Xing H, Li T, Yang Y, Qu R, Pan Y (2015) A modified ant colony optimization algorithm for network coding resource minimization. IEEE Trans Evol Comput 20(3):325–342
16. Liu J, Yang J, Liu H, Tian X, Gao M (2017) An improved ant colony algorithm for robot path planning. Soft Comput 21(19):5829–5839
17. Vanchinathan K, Valluvan KR (2018) A metaheuristic optimization approach for tuning of fractional-order PID controller for speed control of sensorless BLDC motor. J Circuit Syst Comput 27(08):1850123
18. Basturk B (2006) An artificial bee colony (ABC) algorithm for numeric function optimization. IEEE Swarm Intell Symp, USA
19. Karaboğa N, Cetinkaya MB (2011) A novel and efficient algorithm for adaptive filtering: artificial bee colony algorithm. Turk J Electr Eng Comput 19(1):175–190
20. Xue Y, Jiang J, Zhao B, Ma T (2018) A self-adaptive artificial bee colony algorithm based on global best for global optimization. Soft Comput 22(9):2935–2952
21. Karaboga D, Gorkemli B, Ozturk C, Karaboga N (2014) A comprehensive survey: artificial bee colony (ABC) algorithm and applications. Artif Intell Rev 42(1):21–57
22. Gao W, Liu S, Huang L (2012) A global best artificial bee colony algorithm for global optimization. J Comput Appl Math 236(11):2741–2753
23. Liang Y, Wan Z, Fang D (2017) An improved artificial bee colony algorithm for solving constrained optimization problems. Int J Mach Learn Cybern 8(3):739–754
24. Yang XS, Hosseini SSS, Gandomi AH (2012) Firefly algorithm for solving non-convex economic dispatch problems with valve loading effect. Appl Soft Comput 12(3):1180–1186
25. Fister I, Fister I Jr, Yang XS, Brest J (2013) A comprehensive review of firefly algorithms. Swarm Evol Comput 13:34–46
26. Niknam T, Azizipanah-Abarghooee R, Roosta A (2012) Reserve constrained dynamic economic dispatch: a new fast self-adaptive modified firefly algorithm. IEEE Syst J 6(4):635–646
27. Kavousi-Fard A, Samet H, Marzbani F (2014) A new hybrid modified firefly algorithm and support vector regression model for accurate short term load forecasting. Expert Syst Appl 41(13):6047–6056
28. Wahid F, Alsaedi AKZ, Ghazali R (2019) Using improved firefly algorithm based on genetic algorithm crossover operator for solving optimization problems. J Intell Fuzzy Syst 36(2):1547–1562