An Enhanced Artificial Bee Colony: Naïve Bayes Technique for Optimizing Software Testing

Palak[1*], Preeti Gulia[2], Nasib Singh Gill[3]
Department of Computer Science and Applications
Maharshi Dayanand University, Rohtak, India

***Abstract*—Software driven technology has become a part of life and the quality of software largely depends on the extent of effective testing performed during various phases of development. A wide range of nature inspired searching techniques are employed over years to automate the testing process and provide promising solutions to elude the infeasibility of exhaustive testing. These techniques use metaheuristics and work by converting the problem space into search space. A subset of optimized solutions is searched that reduces overall time by shortening the testing time. Objective: An enhanced Artificial Bee Colony- Naïve Bayes optimizer for test case selection is proposed in this paper. This article also aims to provide brief insights into the emergence of hybrid swarm-inspired techniques over the last two decades. Method: The modified Artificial Bee colony is applied after component selection and further optimization is achieved using Naïve Bayes classifier. The proposed technique is implemented and evaluated taking three benchmark programs into consideration. The proposed technique is also compared to other competitive swarm intelligence-based techniques of its class. Results: The experimental results show that the proposed technique outperforms other swarm-inspired techniques in terms of execution time in a given scenario and capable of higher detection of faults with minimal test case selection. Conclusion: The proposed approach is an improvement over existing techniques and helps in huge time and cost saving. It will contribute to the testing society and enhance the overall quality of the software**.

Keywords*—Software testing; artificial bee colony; swarm intelligence; Naïve Bayes; test case selection

I. Introduction

Increasing demand in robust software can be seen as a consequence of rapidly developing hardware industry and outburst in evolution of technology. Smart devices have become our part of our surroundings. Software testing is a very important phase of development of robust software which involves finding possible faults and errors so that the final product meets the overall expectation of the customer without failing. Software testing has always been a hot topic of research for software industry practitioners and researchers. It is the procedure of the identification and authentication of the software services by selecting if it is fulfilling the user's needs. Despite checking correctness of input and output during testing there are many other concerns to deal with. Some of them include dead code, redundant code, faulty components, exceptions etc. These aspects principally affect the overall performance and customer satisfaction. Though, specifications are frequently ignored and there are numerous obstructions to the execution, including the inadequate design, phase limits, absence of automatic apparatuses, and so forth. The cost of testing increases with complexity of the system. More important aspects to be considered are the continuous updates in the system and addition of new functionalities. The configurable architecture of the software now- a- days makes the testing process more difficult due to the behavioral changes of components at each configuration (Myra B. Cohen et al. 2007). Poor testing ultimately leads to system failure and lowers down the faith of the customer. The significance of software testing is that it helps the software programmers for building error-free and acceptable software. A worthy and quality test suite can catch most of the errors without jumping time constraints. The process of testing is carried out at functional (Black Box) as well as structural level (White Box). Functional testing aims to check correctness of the input and output only whereas structural testing checks the deep insights at the root level considering the architectural aspects of the program.

Optimization of testing processes can be done at several levels of software testing life cycle. The field of automated testing is growing day by day due to various underlying benefits. With the increase in dependence of software and ever-growing system requirements, manual testing is not possible at every level of development. Manual testing is time consuming whereas automated testing is fast and repeatable. Smaller projects are feasible to be tested manually but this is not the case for large dynamic projects. The manual testing time rises exponentially with increase in the length of code but this is not the case in automated testing. All you need to do is to write an isolated code called a "Test Code" to test the main functional code. This code can be executed any number of times to find errors whenever required. One more important benefit to mention here is that automated testing permits you to refactor ("changing the structure without altering the actual functionality") the code without any hustle to manually test the code every time you refactor the code. From automated generation of effective test suites to test case selection and prioritization, research is going on which encompasses meta-heuristics and artificial intelligence. Automated testing helps you emphasize more on the quality of the software rather than memorizing what and how to test. Test automation can be done at various levels. Unit testing automation involves testing each independent functional unit without considering the external dependencies. Optimization of such activities utilizes various engineering domains like data mining, artificial intelligence, machine learning, swarm intelligence and many more. Over years, soft computing has emerged as a

*Corresponding Author

promising solution towards optimization problems that involves metaheuristics [1]. Various evolutionary algorithms are also preferred over random search techniques for test suite generation that attracts researchers in this field [2]. This paper also aims to utilize a swarm-based approach for optimization in the testing process.

Rest of the paper is organized as: Section II briefs the emergence of various swarm inspired techniques according to the timeline over the past two decades. Section III presents related work present in different literature over recent years in the similar domain of hybrid Artificial Bee Colony (ABC) optimization techniques. The proposed method in the form of a flowchart is presented in Section IV. Results and evaluation of the research are reported in Section V. Finally, the overall conclusion of the paper is given in Section VI along with the future scope of the article.

II. Related Work

Swarm Intelligence (SI) is a popular field of research that is motivated by the natural phenomenon of a population (group) of various living organisms in their natural habitat for search of food, shelter and security. Over the past few decades swarm-based optimization techniques are emerging at a very fast pace due to the inherent flexibility and robustness [3]. SI refers to collective intelligence that has attracted researchers in almost every area of industry. The community behavior of real living organisms dwelling in nature to protect and feed their community is the real inspiration behind SI. The individuals of a swarm interact mutually with each other and also locally with their surroundings in a decentralized way for survival forming a coherent system that can be modelled into a functional pattern [4]. SI laid its roots in the early 1990s and has become an ever-evolving field since then. This section gives brief glimpses of some important swarm-based optimization techniques that have emerged till date and applied in the field of software testing.

In 1999, “Ant colony optimization (ACO)” was proposed which is inspired by food searching behavior of colonies of real ants [5]. Ants communicate with each other by secreting a chemical substance on their path. The concentration of this chemical increases on the shorter path when the number of ants taking that path increases over time. This simple but robust behavior gave the inspiration to build a meta-heuristics model that can be used in various search optimization problems. ACO is well utilized in test case generation, selection and prioritization problems in past few years [6] [7] [8]. The problem of testing optimization is first converted to graphical search problem and then ACO is applied [9]. Various hybrid approaches with ACO have also been proposed with other techniques such as Genetic Algorithm (GA) which aims to select a minimal test suite for higher fault coverage [10].

An efficient algorithm inspired from the social behavior of bees in the search of food was given by Dervis Karaboga et al. in [11] named as “Artificial Bee Colony (ABC)” Optimization. They considered three types of bees and converted their behavior to a mathematical model. Initially half of the bees in the beehive are termed as “Employed Bees”. They search for the food randomly near the hive and come back to the hive. They dance in front of the second set of the bees called “Onlooker Bees”. This dance serves as the probability function for comparison and selection of the better food source. If the food source of any Employed Bee is exhausted then it becomes the scout and serves as the stopping criteria for the algorithm.

Due to its lightweight deployment with very small amounts of controller factors, numerous hard works have been done to discover ABC research. ABC has gained popularity since its origin and researchers are more interested in making hybrid algorithms that provide more diversification in searching the solution. ABC is inspired from natural behavior of honeybees in the search of nectar and their community behavior in maintaining the highest nectar collection. The success of ABC can be anticipated by vast literature available under reputed indexing that shows the interest of researchers in this approach. Originally the ABC technique employs three types of bees: Employed, Onlooker and Scout bees [11]. The employed bees are linked to a definite food source. Initially one employed bee is assigned to a food source. They transmit vital information such as navigation information, location and the profitability of the food source and carry the data with the rest of bees at the beehive. The onlooker bees are accountable for food source detection exploiting the information delivered by employed bees. The scout bees dispensed randomly to hunt the new food source whenever there is no further improved solution is found by either employed or onlooker bees [D. Karaboga, 2005]. The assumption is that the employed bees whose food source is exhausted are transformed into “scout bees" and commence a new exploration for the food source. The parallel conduct of these three bees speeds up the generation of feasible independent paths and software test suite optimization. ABC performs competitively to other conventional soft computing techniques and has gained popularity over last decade due to its easy implementation. Various hybrid and enhanced ABC techniques evolved over the past decade that are used for optimization problems especially in the field of software testing. Table I provides a brief insight into such hybrid ABC techniques:

TABLE I. EXISTING HYBRID ARTIFICIAL BEE COLONY BASED OPTIMIZATION TECHNIQUES

Author	Year	Technique used	Application Area
[Lakshminarayana P et al. [12]**]**	2021	Hybrid Cuckoo Search and Bee Colony Algorithm	Optimization of test cases and generation of path convergence within
[Hussain, Kashif et al. [13]**]**	2020	Scoutless ABC	Model-driven testing
[Saju Sankar S et al. [14]**]**	2020	Comprehensive Improved Ant Colony Optimization (ACIACO)	Automated test case generation
[Ammar K. Alazzawi et al. [15] [16] **]**	2019, 2020	Hybrid artificial bee colony algorithm and practical swarm optimization with constraint support	Generation of variable t-way test sets
[Snehlata Sheoran et al. [17]**]**	2019	Memory based ABC	Data flow testing to find out and prioritize the definition-use paths
[Hu Peng et al. [18]**]**	2019	Best Neighbor-guided artificial bee colony	Continuous optimization problems
[Sandeep Dalal et al. [19]**]**	2018	BCO-m-GA	Test case selection
[Faten Hamad [20]**]**	2018	Modified ABC	Software structural testing
[Sahoo, Rajesh et al. [21]**]**	2017	Hybrid PSO and BCA	Model-driven testing
Zohreh Karimi Aghdam et al. [22]	2017	Modified Fitness Function in ABC	Generate Test Data for Software Structural Testing
Xianneng Li et al. [23]	2016	Artificial bee colony algorithm with memory	Continuous optimization problems
D. Karaboga et al. [24]	2014	Quick ABC with different functions for employed and onlooker bees	Numerical Optimization Problems

III. PROPOSED APPROACH (ENHANCED ABC- NAÏVE BAYES OPTIMIZATION)

In this section, a novel “Enhanced ABC- Naïve Bayes Optimization (ABC-NB)” is proposed for software test case selection. Fig. 1 shows the flowchart of proposed methodology that is inspired from memory-based ABC [17], [19], [23] along with the Naïve Bayes Classifier to further enhance the results.

ABC is highly exploited in the field of software testing that shows the capability of the method. ABC also provides the inherent advantage of independent and parallel behavior of three types of honey bees. Also, this is a non- pheromone-based technique that decreases the computational complexity up to a great extent [25]. That’s why we prefer ABC over other swarm-based techniques for optimization of the testing process. The algorithm starts with the selection of the project. We are considering Components Based Software (CBS) development paradigm into account due to the inherent modularity and capability of handling complex projects.

The proposed approach works as follows: component-based projects are selected and uploaded to the repository and their individual components are extracted. Here components refer to each individual unit of work that has predefined interfaces and boundaries. Further each component is subdivided into modules. A component may consist of one of more modules and other components. “Enhanced ABC with memorizing capability” is applied for selecting a subset of test cases in the given fault matrix. The memory element is used to store the best solution found so far to maintain overall intensification as well as diversification. Originally ABC has three phases each related to three different types of bees in the beehive. This behavior is inspired from real beehives where nectar collection is a result of highly organized and collaborated team work. Fig. 1 shows the detailed flowchart of the proposed technique. The various phases and role of different type of bees is as follows:

A. Initialization

First of all, we need to initialize the population size i.e., no. of candidate solutions (initial number test cases in our case) that is denoted by TN. Each solution (test case) is related to D dimensional parameter vector that defines a particular solution based on fault matrix i.e.

Xi = $\{x_i^1, x_i^2, \ldots, x_i^D\}$, i = 1, 2, . . ., TN.

Initially the memory element is kept empty.

For a fault j in fault matrix for ith test case, the initial value x_i^j is generated by

$$x_i^j = x_{min}^j + \text{rand}(0, 1) \times (x_{max}^j - x_{min}^j) \quad (1)$$

where, i = 1, 2, . . ., TN and j = 1, 2, . . ., D. rand(0, 1) is a random number whose value belongs to [0, 1], max and min are the maximum and minimum value in case of each parameter respectively.

B. Employed Bees

Each employed bee maintains individual solutions so their number is equal to the total number of test cases, that is, TN. For each test case i, employed bees generate a new vector Y_i.

The neighbor search is performed by modifying jth parameter of Y_i where j ∈ {1, 2, . . ., D} is selected randomly. The following equation is used for updates done by employed bees (Karaboga and Basturk 2007):

$$y_i^j = x_i^j + \Phi_i^j \times (x_i^j - x_k^j). \quad (2)$$

Here k is randomly selected and i ≠ k. X_i will be replaced by Y_i; in the population if Y_i is better. Φ_i^j is for randomness ranging in [−1, 1].

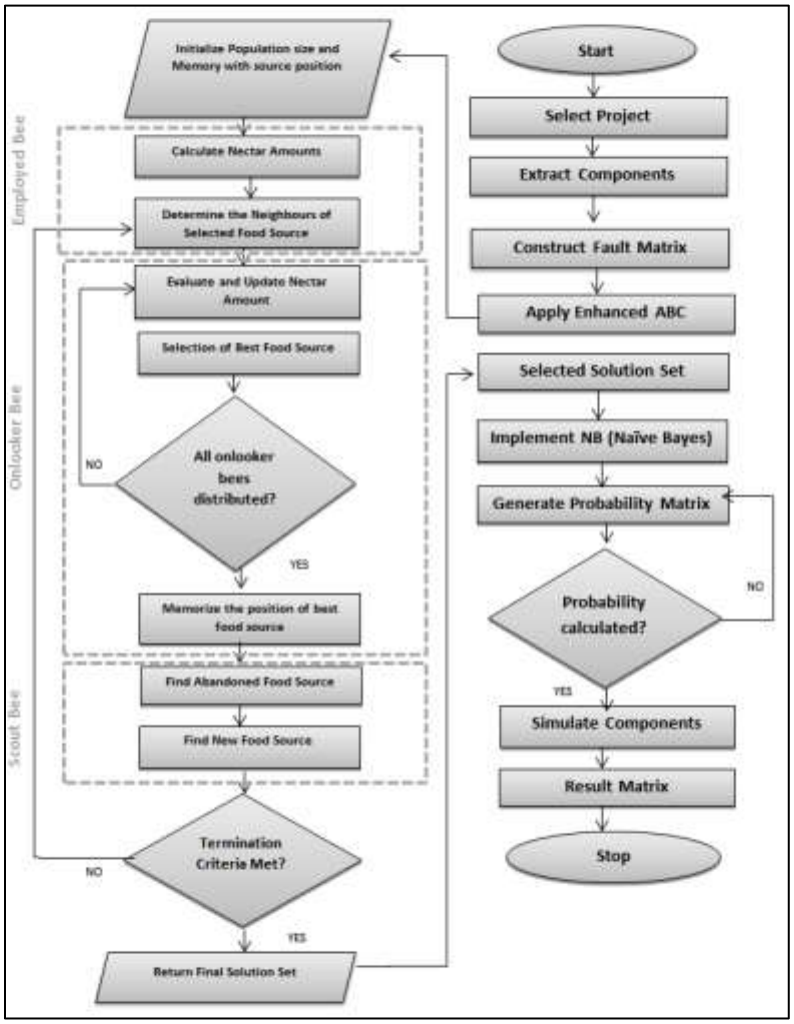
Initialize Population size and Memory with source position
Employed Bee
Calculate Nectar Amounts
Determine the Neighbours of Selected Food Source
Evaluate and Update Nectar Amount
Selection of Best Food Source
Onlooker Bee
All onlooker bees distributed?
NO
YES
Memorize the position of best food source
Scout Bee
Find Abandoned Food Source
Find New Food Source
Termination Criteria Met?
NO
YES
Return Final Solution Set
Start
Select Project
Extract Components
Construct Fault Matrix
Apply Enhanced ABC
Selected Solution Set
Implement NB (Naïve Bayes)
Generate Probability Matrix
Probability calculated?
NO
YES
Simulate Components
Result Matrix
Stop

Fig. 1. Flowchart of Proposed Methodology.

Each employed bee also has memory ME_i, that stores the best solution found so far. After every iteration memory is also updated. During next cycle, ME_i is searched first before randomly selecting neighbor solution.

C. Onlooker Bees

The probability of selecting a test case "i" by an onlooker bee is denoted by p_i, which is calculated by

$$p_i = \frac{fit_i}{\sum_{j=1}^{TN} fit_j} \tag{3}$$

where fit_i denotes the fitness value of ith test case, which is calculated on the basis of probability of a test cases to find given set of errors. The onlooker bee also generates a new solution Yi using equation (2) similar to the employed bee. Each onlooker bee also has memory MO_i, that stores the best solution found so far. After every iteration memory is also updated. During next cycle, MO_i is searched first before randomly selecting neighbor solution.

D. Scout Bee

When a solution cannot be further improved by either employed or onlooker bee that solution is considered as poor performing in the process of evolution as must be removed from the final solution set. In such a scenario, a scout bee is generated, it abandons the poor performing test cases and starts with a whole new random solution. Scout bees maintain randomness and diversification in the algorithm.

After the application of Enhanced ABC, a solution set of promising test cases is returned to the system. Here comes the role of Naïve- Bayes Classifier. It generates the probability matrix over the solution set that is returned in the previous stage and further classifies the solution set. Hence a reduced result set is generated. Naïve Bayes is a family of classification techniques that assumes all features into consideration as independent and of equal weight. The proposed technique is applied on three component-based student projects and implemented in ten iterations with fault matrix of size 50*50 in each project. Errors are induced using mutation to test the efficiency of the proposed method.

IV. Results and Discussion

The proposed approach is implemented in "Visual C# Express 2010" using three student's projects namely: Café Management (CM), Hospital Management (HM), and Payroll System (PS). All of them are implemented in C# using component-based paradigm. The reason for selecting Visual

C# projects is the intrinsic component-based approach that is offered by this platform. The details of these projects are given below in Table II:

TABLE II. DETAILS OF STUDENT'S PROJECTS

Project Name	**kLOC**	**Number of Components**	**Total Number of Modules**
Café Management (CM)	69	6	46
Hospital Management (HM)	78	5	53
Payroll System (PS)	43	3	34

A. No. of Selected Test Cases vs No. of Faults Detected

The experiment is conducted for ten iterations to rule out any chances of error and for averaging of the results with fault matrix of size 50*50 in each project. Initially it is assumed that each test case is capable of finding at least one error. Errors are induced using mutation to test the efficiency of the proposed method. Gradually as the algorithm converges, a smaller fault matrix with a lesser number of selected test cases, Table III shows the performance of the proposed technique in terms of percentage of test case selected and percentage of faults detected.

Fig. 2 shows the results in graphical form. It is depicted that the proposed ABC-NB technique selects less than 47 % of test cases to achieve near optimal fault coverage. The size of the test suite the faster the process is. The results of the proposed technique prove promising in selecting better and shorter test suite so that overall execution time can be reduced.

B. Comparison of Execution Time

Being a costly and time-consuming process, software test execution time plays a very important role. Cost can be greatly minimized by decreasing the execution time without compromising with the quality of test suite. On the basis of fault matrix, the execution time of selected test cases by the proposed approach is compared with the execution time of selected test cases by other swarm-based techniques namely PSO, ACO, ABC and the results are summarized in Table IV.

Fig. 3 shows the comparative graph for the same depicting the clear time saving that can be achieved using the Enhanced ABC- Naïve Bayes technique. It can be argued from the experimental results that the proposed hybrid technique is capable of providing time saving as compared to other competitive techniques. As it shortens the execution time in the given scenario, efforts and cost are automatically reduced.

TABLE III. PERFORMANCE OF ENHANCED ABC- NAIVE BAYES FOR SELECTION OF TEST CASES AND FAULTS COVERED

Project name	**Total No. of Test Cases**	**Total Number of faults**	**Number of test cases selected**	**% of test cases selected**	**No. of Faults Detected**	**% of faults detected**
CM	50	50	21	**42%**	49	**98%**
HM	50	50	19	**38%**	50	**100%**
PS	50	50	23	**46%**	50	**100%**

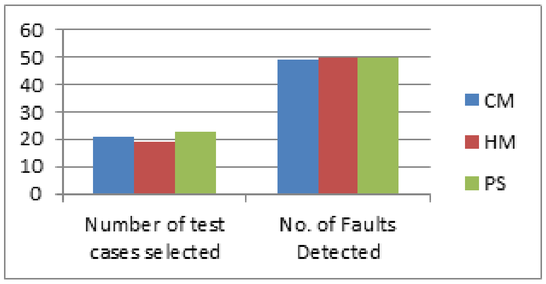

Fig. 2. Performance of Enhanced ABC- Naive Bayes.

TABLE IV. EXECUTION TIME OF SELECTED TEST CASES IN MILLISECOND (MS)

Algorithm/ Project	**CM**	**HM**	**PS**
PSO	165	133	124.3
ACO	160	145.5	120.4
ABC	**125.6**	143	111.9
Enhanced ABC- Naive Bayes	127.3	**129.4**	**105.4**

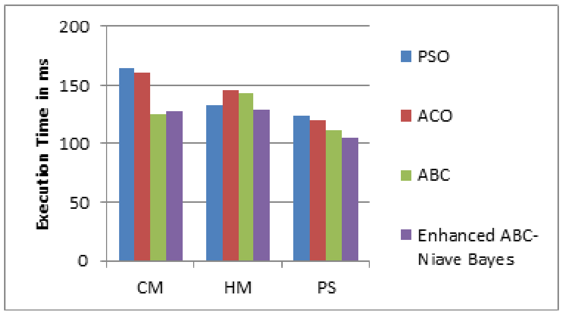

Fig. 3. Comparison of Execution Time.

V. CONCLUSION AND FUTURE SCOPE

Swarm intelligence always inspired researchers to optimize the search problems to save time and money. In this paper, a novel Enhanced ABC – Naïve Bayes algorithm is proposed that is inspired from the colony of honey bees for optimization of test case selection. Being a time consuming and important task, testing always requires optimization. The proposed technique is applied on three component-based student projects and implemented in ten iterations with fault matrix of size 50*50 in each project. Errors are induced using mutation to test the efficiency of the proposed method. The results show that the proposed method is able to find near optimal (i.e.,~ 100%) faults in less than 47 % of total test cases. Thus, a huge amount of time saving can be achieved. The proposed method ABC-NB is also compared with other swarm-based techniques of its class by taking execution time of the selected test cases as a parameter. The proposed technique outperforms PSO, ACO and original ABC as depicted by the results. In future, the proposed method will be compared and evaluated with other swarm-based techniques of its class using more parameters to assess the efficiency and accuracy of the proposed method.

REFERENCES

[1] P. Gulia and P. Palak, “Nature Inspired Soft Computing Based Software Testing Techniques For Reusable Software Components,” J. Theor. Appl. Inf. Technol., vol. 95, no. 24, pp. 6996–7004, 2017.

[2] J. Campos, Y. Ge, N. Albunian, G. Fraser, M. Eler, and A. Arcuri, “An empirical evaluation of evolutionary algorithms for unit test suite generation,” Inf. Softw. Technol., vol. 104, no. August, pp. 207–235, 2018, doi: 10.1016/j.infsof.2018.08.010.

[3] A. Abraham, H. Guo, and H. Liu, “Swarm Intelligence: Foundations, Perspectives,” Swarm Intell. Syst., vol. 25, pp. 3–25, 2006.

[4] I. Aydogdu, M. P. Saka, and E. Do, “Analysis of Swarm Intelligence À Based Algorithms for Constrained Optimization,” in Swarm Intelligence and Bio-Inspired Computation, Elsevier Inc., 2013, pp. 25–48.

[5] M. Dorigo and G. Di Caro, “Ant Colony Optimization: A New Meta-Heuristic,” in Proceedings of the 1999 congress on evolutionary computation-CEC 99, 1999, pp. 1470–1477.

[6] B. Suri and S. Singhal, “Analyzing test case selection & prioritization using ACO,” ACM SIGSOFT Softw. Eng. Notes, vol. 36, no. 6, p. 1, 2011, doi: 10.1145/2047414.2047431.

[7] U. M Diwekar and B. H Gebreslassie, “Efficient Ant Colony Optimization (EACO) Algorithm for Deterministic Optimization,” Int. J. Swarm Intell. Evol. Comput., vol. 05, no. 01, 2015, doi: 10.4172/2090-4908.1000131.

[8] P. Palak and P. Gulia, “Ant Colony Optimization Based Test Case Selection for Component Based Software,” Int. J. Eng. Technol., vol. 7, no. 4, pp. 2743–2745, 2018, doi: 10.14419/ijet.v7i4.17565.

[9] S. F. Ahmad, D. K. Singh, and P. Suman, “Prioritization for Regression Testing Using Ant Colony Optimization Based on Test Factors,” in Intelligent Communication, Control and Devices, 2018, pp. 1353–1360.

[10] P. Palak and P. Gulia, “Hybrid swarm and GA based approach for software test case selection,” Int. J. Electr. Comput. Eng., vol. 9, no. 6, pp. 4898–4903, 2019, doi: 10.11591/ijece.v9i6.pp49898-4903.

[11] D. Karaboga and B. Basturk, “A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm,” J. Glob. Optim. 39, pp. 459–471, 2007, doi: 10.1007/s10898-007-9149-x.

[12] L. P and T. V Suresh Kumar, “Automatic Generation and Optimization of Test case using Hybrid Cuckoo Search and Bee Colony Algorithm,” J. Intell. Syst., vol. 30(1), pp. 59–72, 2021.

[13] K. Hussain, M. Najib, M. Salleh, S. Cheng, Y. Shi, and R. Naseem, “Artificial bee colony algorithm: A component-wise analysis using diversity measurement,” J. King Saud Univ. - Comput. Inf. Sci., vol. 32, no. 7, pp. 794–808, 2020, doi: 10.1016/j.jksuci.2018.09.017.

[14] S. S. S. B and V. C. S. S. B, “An Ant Colony Optimization Algorithm Based Automated Generation of Software Test Cases,” in the International Conference on Swarm Intelligence. (ICSI 2020), 2020, vol. 1, pp. 231–239, doi: 10.1007/978-3-030-53956-6.

[15] A. K. Alazzawi, H. Rais, and S. Basri, “HABC : Hybrid Artificial Bee Colony For Generating Variable T-Way Test Sets,” J. Eng. Sci. Technol., vol. 15, no. 2, pp. 746–767, 2020.

[16] A. K. Alazzawi, H. Rais, S. Basri, and Y. A. Alsariera, “PhABC : A Hybrid Artificial Bee Colony Strategy for Pairwise test suite Generation with Constraints Support,” in IEEE Student Conference on Research and Development (SCOReD), 2019, no. October, pp. 106–111, doi: 10.1109/SCORED.2019.8896324.

[17] S. Sheoran, N. Mittal, and A. Gelbukh, “Artificial bee colony algorithm in data flow testing for optimal test suite generation,” Int. J. Syst. Assur. Eng. Manag., vol. 11, pp. 340–349, 2019, doi: 10.1007/s13198-019-00862-1.

[18] H. Peng, C. Deng, and Z. Wu, “Best neighbor-guided artificial bee colony algorithm for continuous optimization problems,” Soft Comput., vol. 1, no. 23, pp. 8723–8740, 2019, doi: 10.1007/s00500-018-3473-6.

[19] S. Dalal, “Performance Analysis of BCO-m-GA Technique for Test Case Selection,” Indian J. Sci. Technol., vol. 11(9), no. March, 2018, doi: 10.17485/ijst/2018/v11i.

[20] F. Hamad, “Using Artificial Bee Colony Algorithm for Test Data Generation and Path Testing Coverage,” Mod. Appl. Sci., vol. 12, no. 7, pp. 99–112, 2018, doi: 10.5539/mas.v12n7p99.

[21] R. Sahoo, S. Nanda, and D. P. Mohapatra, “Model Driven Test Case Optimization of UML Combinational Diagrams Using Hybrid Bee Colony Algorithm,” Int. J. Intell. Syst. Appl., no. July, 2017, doi: 10.5815/ijisa.2017.06.05.

[22] Z. K. Aghdam and B. Arasteh, “An Efficient Method to Generate Test Data for Software Structural Testing Using Artificial Bee Colony Optimization Algorithm,” Int. J. Softw. Eng. Knowl. Eng., vol. 27, no. 6, pp. 951–966, 2017, doi: 10.1142/S0218194017500358.

[23] X. Li and G. Yang, “Artificial bee colony algorithm with memory,” Appl. Soft Comput., vol. 41, pp. 362–372, 2016, doi: 10.1016/j.asoc.2015.12.046.

[24] D. Karaboga and B. Gorkemli, “A quick artificial bee colony (qABC) algorithm and its performance on optimization problems,” Appl. Soft Comput. J., vol. 23, pp. 227–238, 2014, doi: 10.1016/j.asoc.2014.06.035.

[25] S. Sekhara, B. Lam, M. L. H. Prasad, U. K. M, and S. Ch, “Automated Generation of Independent Paths and Test Suite Optimization Using Artificial Bee Colony,” in International Conference on Communication Technology and System Design, 2011, vol. 30, no. 2011, pp. 191–200, doi: 10.1016/j.proeng.2012.01.851.