International Journal of Engineering & Technology, 7 (4) (2018) 3217-3222

SPC

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET
doi: 10.14419/ijet.v7i4.19898
Research paper



"BA-TPF": A Novel Approach Towards Regression Test Case Optimization

Sandeep Dalal [1], Sudhir [2]*, Kamna Solanki[1]

[1] *Assistant Professor, Maharshi Dayanand University, Rohtak, India*
[2] *Research Scholar, Maharshi Dayanand University, Rohtak, India*
**Corresponding author E-mail: Sudhir.gehlawat1234@gmail.com*

Abstract

One of the most critical and expensive segment of software development is software testing. It is practically not feasible to conduct exhaustive testing on any software system due to cost and time constraints. Moreover, amendments and modifications in a software for the purpose of fixing failures is inevitable. Therefore, one can never guarantee that a software has been thoroughly tested. Testing the whole software system exhaustively by running the entire set of test cases after every modification is near to impossible. Hence, the software system is tested by running the optimized test suite for minimizing the cost and effort without compromising the quality of the test suite in terms of uncovering faults and providing better coverage based on many parameters. This paper presents a novel technique "BA-TPF" for multi-objective test suite optimization for enhancing path coverage with minimized repetition ratio and minimized test suite size.

Keywords: *Software Testing; Test Optimization; Test Case Minimization and Prioritization.*

1. Introduction

Now-a-days software is getting ubiquitous and there is an increased dependence of human society on the computing features and services provided by software. Testing consumes 30-50% of total cost of software development, resources and time [1-3]. It can be concluded that testing is one of the most crucial parts of any software development process. The objective of testing is not only to find software bugs, but also to throw light on the situations that could negatively affect the customer. Execution of large number of test cases does not guarantee the quality of a software. Moreover, effective testing process depend on the quality of test cases executed rather than quantity of test cases. "Effective test suite is referred to unique and non-redundant set of test cases which avoids the wastage of available limited resources" [4]. Redundancy of test cases in effective test suite is not a desirable property. So, the need of the hour is to propose more testing techniques, which are cost-effective in terms of minimized redundant test cases, effective coverage, higher fault detection capability and enhanced path coverage etc. [5-6].

Testing of a software is a continuous and multiphase process. Fundamental goal of software testing is to discover maximum number of errors and selection of the optimized test cases (or test suite) where optimization means prioritized and minimized set of test cases. Selection of test cases for execution for a particular program is a difficult task [7-8]. This section enlightens the test suite prioritization and test suite minimization techniques in brief.

Test suite prioritization is an important technique to identify a way to schedule and execute the test cases according to a specified priority order (e.g. Higher or lower priority order). In test suite prioritization technique, a rank is allotted to every test case in a test suite using some criteria.

The primary objective of the test suite minimization technique is to select the test cases based on pre-specified coverage criteria [9][10]. This technique removes the redundant or obsolete test cases from the original test suite. The pre-specified coverage criteria could be code, path, loop, block, control and data flow coverage.

2. Related work

Numerous researchers have applied nature inspired meta-heuristic techniques like ant colony optimization algorithm [11], [12], [21] genetic algorithm [22-30], bee colony optimization algorithm [31-36], bat algorithm [37-39] etc. for generation and optimization of test cases [43-45].

- Jeyamala and Mohan, (2009): The authors utilized ABC ("Artificial Bee Colony Optimization") algorithm for test suite optimization. Experimental analysis of their proposed ABC approach beats GA in test suite optimization [36].
- Krishnamoorthi and Mary, (2009): Sub-sequences of the original test suite are prioritized using GA in time-controlled environment to achieve superior fault detection rate as compared to randomly prioritized test suites. APFD metric has been used for performance evaluation of proposed technique [24].
- Rathore et. al., (2011): Tabu search and GA are collectively used for automatic test case generation. The performance of the proposed algorithm is better than individual GA based test case generation techniques [30].
- Srivastava et. al., (2014): This paper presented a technique for estimating the efforts for software testing using bat algorithm. The proposed model by the authors can be used to optimize the testing efforts by generating solutions iteratively [39].

Copyright © 2018 Sandeep Dalal, Sudhir. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

3. Novel approach "BA-TPF" for test suite optimization

This paper proposes a novel "BA-TPF" technique for test case optimization. "BA-TPF" technique is based on application of "Test Path Fitness (TPF)" to the test sequences generated using Bat Algorithm (BA). The proposed technique works by applying the bat algorithm on the UML based state-chart diagrams to generate the favourable test sequences. Later on, fitness function is applied to these test sequences to calculate "TPF" values. Prioritized and minimized test sequences are thus obtained by arranging the test cases in descending order of TPF values.

3.1. Bat algorithm and its applications

Bat algorithm is a nature inspired algorithm developed by Xin-She Yang. Bat algorithm has found its applicability in solving many optimization problems. The following points completely define a bat behavioural characteristic [42]:

1) "All bats use echolocation to sense distance, and they also 'know' the difference between food/prey and background barriers in some magical way".
2) "Bats fly randomly with velocity VI at position xi with a frequency f_{min}, varying wavelength λ and loudness A_0 to search for prey. They can automatically adjust the wavelength (or frequency) of their emitted pulses and adjust the rate of pulse emission $r \in [0, 1]$, depending on the proximity of their target."
3) "Although loudness can vary in many ways, we assume that the loudness varies from a large (positive) A_0 to a minimum constant value A_{min}."

Bat algorithm is the newest nature motivated algorithm among the population based and meta-heuristic algorithms that basically works by following the magical "echolocation" process used by bats to find it's pray [42]. Bat algorithm has proved its efficiency and efficacy in software testing in comparison to the contemporary nature inspired algorithms. Only few researchers have worked towards finding the applicability of bat algorithm in testing.

- Muhammed Maruf Ozturk, (2017): The author proposed a bat inspired test case prioritization algorithm (BITCP). The pray distance and loudness were adapted as execution time of test case and number of defects detected. The proposed algorithm yielded higher value of APFD metric in comparison to other contemporary techniques. The authors also emphasized that LBS (Local Beam Search) is also a promising search-based algorithm which yields better results than ACO, PSO (Particle Swarm Optimization) and greedy algorithms [37].
- Praveen Ranjan Srivastava, (2017): The author proposed a testing technique for path generation using a hybrid methodology of cuckoo search and bat algorithm. The proposed technique generates all the independent paths of a software code through its control flow graph [38].
- Srivastava et. al., (2014): This paper offered a technique for estimating the efforts for software testing using bat algorithm. The proposed model by the authors can be used to optimize the testing efforts by generating solutions iteratively. The results were analysed by the authors to prove that the estimated testing efforts were close to the actual efforts and were better than other current techniques for effort estimation [39].

Many variants of Bat algorithm have been developed in recent years like "Fuzzy Logic Bat Algorithm (FLBA)", "Multi-objective bat algorithm (MOBA)", "K-Means Bat Algorithm (KMBA)", "Chaotic Bat Algorithm (CBA)", "Binary bat algorithm (BBA)", "Differential Operator and Levy flights Bat Algorithm (DLBA)", "Improved bat algorithm (IBA)"[43]. Active research must be conducted to make utilization of these enhanced bat algorithms in software testing.

3.2. Architecture/Model for the Proposed "BA-TPF" Technique

The proposed model for approach "BA-TBF" applies the bat algorithm on UML state-chart diagrams representing user requirements and converting them into an intermediate form for test sequence generation [40]. Then, the generated test sequences are optimized using the test path fitness algorithm. The five -layered model/ architecture of the proposed approach can be conveniently understood using the diagrams figure1 and figure2. Outcome from every layer is passed on to the next upcoming layer as input.

The proposed BA-TBF technique for test suite optimization has five layers:

1) User Requirements from SRS
2) UML Representation
3) Intermediate Graph Generation (IGG)
4) Bat algorithm Core Layer ("BA")
5) Test Path Fitness Measurement ("TPF")

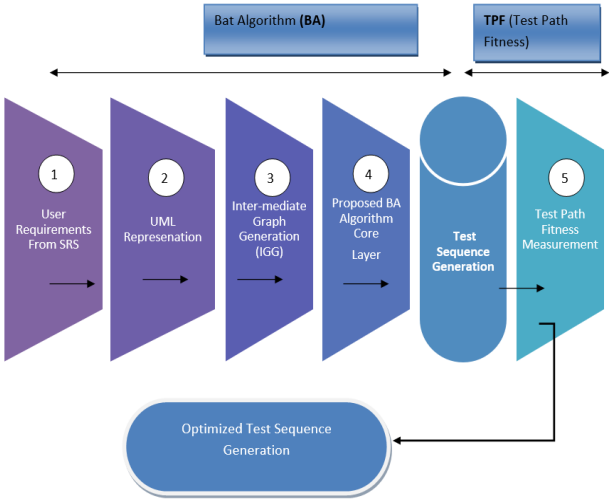


Fig. 1: Model for Proposed "BA-TPF" Technique.

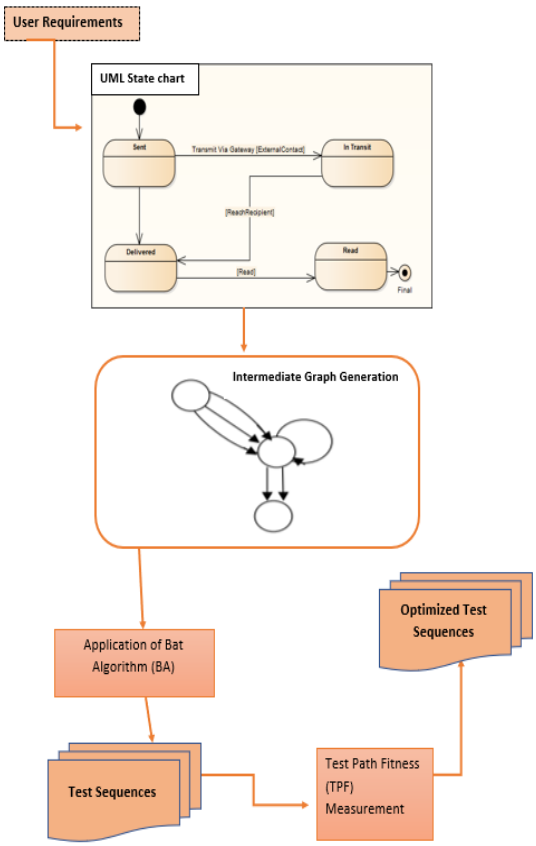


Fig. 2: Architecture for the Proposed Approach.

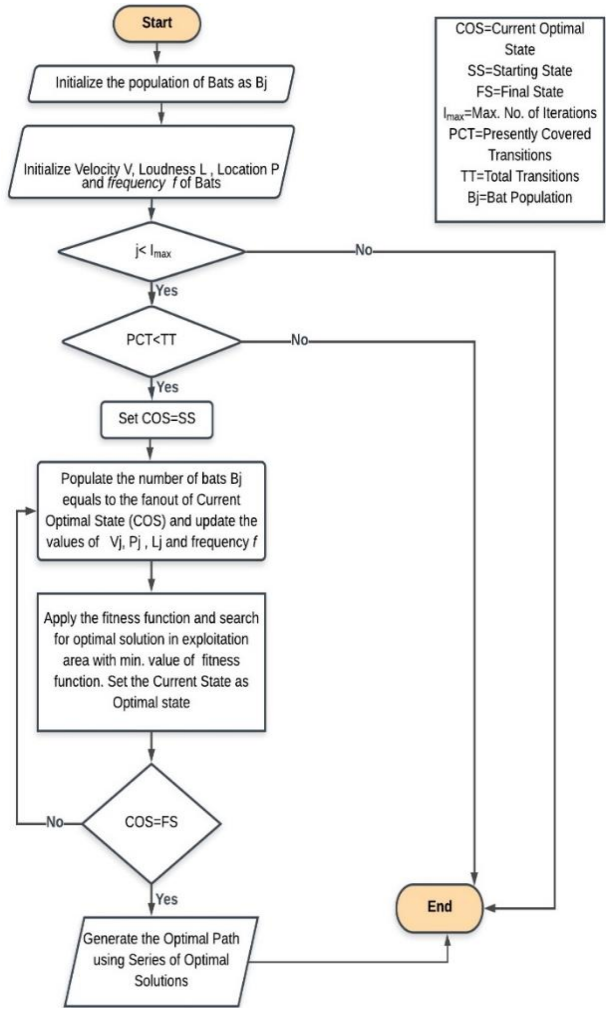


Fig. 3: Flowchart for the Proposed "BA-TPF" Approach.

1) User Requirements from SRS:

The first and foremost layer captures and frames the end-user requirements. It is one of the most vital step in the proposed approach as incorrect specifications may spoil the usefulness of the approach.

2) UML Representation:

UML (Unified Modelling Language) state-chart diagrams represents end-user (or clients) requirements. UML representation of functional user requirements assists the stakeholders, developers and testers to make a convenient understanding of the workflow of the system. These diagrams extend the end-user specification from a varied perspective, which is capable of being interpreted from various viewpoints. The current research work utilizes UML state-chart diagrams for test sequence generation and optimization.

3) Inter-mediate Graph Generation (IGG):

UML diagram is converted into an easily understandable format known as intermediate graph in third layer. Therefore, third layer deals with transformation of the UML state chart diagrams into an intermediate form.

4) Bat Algorithm Core Layer:

Bat algorithm is applied to the intermediate graph to generate favorable test sequences. This layer emphasizes to generate the favourable test sequences that traverse most desirable transitions in terms of achieving higher path coverage. The purpose has been reached out by altering the "fitness function" applicable to produce optimal state and sequence by bats. The bat algorithm utilizes certain parameters out of which, some parameters assist in computing the traversal of bats, while the remaining parameters serve the objective of discovering an optimum local solution amid of the observed solutions in neighbourhood. Following section illustrates various parameters used in bat algorithm along with their basic characteristics. These parameters were selected because some parameters of the state chart diagram are required to achieve accuracy in results.

- Inclination Factor (IF): The significance of each state is defined by this factor. Highest significance of a state is denoted as 0 and lowest significance of a state is denoted as 1. Inclination Factor designates the significance of a state and can be measured in terms of quantity of fan-in and fan-out of a state.
- Chance Factor (CF): The possibility or probability of a bat to reach its final state is defined by chance factor. The value of this parameter relies on the number of final states. Higher quantity of end states diminishes the value of chance factor. If number of final states in a state-chart diagram is z, then CF is defined as:

$$CF = (1/z) \quad \ldots\ldots (1)$$

- Velocity (V): Velocity of the bats increases as the bats approaches near to its pray.
- Location (P): The location of the bat keeps changing. Location is regularly updated depending upon the local optimal solution and loudness of the bat.
- Frequency (f): Number of end states influences the frequency. Larger number of end states require higher frequencies by the bat.
- Loudness (L): Value of this parameter of the bat algorithm keeps on decreasing as the bats approaches near to its pray.
- The bat algorithm core layer operates to produce a series of desirable test sequences to assist in achieving better path coverage for optimization. Some related parameters like loudness, frequency, velocity etc. are assigned and initialized. The bats traverse randomly during each iteration to attain an optimum local solution. These produced solutions are thereafter passed through a fitness measurement scale using a particular fitness function. This process is repeated iteratively to reach to best sequences of solution nodes. The fitness function has been framed to measure the optimality of generated local solution. Fitness function can be calculated using some basic parameters defined below:

$$\text{Fitness Function } \phi = [1/ (1\text{-IF}) *(v_j/ f_j)] * \mu \quad (2)$$

Where μ=CF×IF … (μ denotes the probability of reaching to end state using a given path)

CF (Chance Factor) = (1/no. of final states)

IF (Inclination Factor) = (fan-in×.75) + (fan-out×.25) in range [0-1]

$$Fj=f_{lowest} + (f_{highest}-f_{lowest}) *\mu \quad (3)$$

Fj denotes the bat frequency to search it is pray.

$$V_j^t = V_j^{t-1} + (p_j^t-p^*) f \quad (4)$$

Where V_j^t is velocity of bat at time t and it increases as the bat moves near the pray and p* is the local optimal solution selected from current optimal solutions.

$$P_{new}= (€ \times Lt) + P_{old} \quad (5)$$

Where p_{new} is the new position for random path of each bat and € is random integer between 0 and 1.

L_t is the loudness of the bat which decreases as the bat moves near its pray

$L_j^{t+1} = L_j^t \times ĉ$ where ĉ is a constant (taken as 0.85)

$f_{min} = 1$, $f_{max}= 10$.

The loudness L_j needs to be updated invariantly as the iterations progresses. Generally, the value of loudness can be taken as per one's convenience [42].
As the bats reaches near to their pray, the velocity of the bat keeps on increasing. Hence, velocity of the bat is directly proportional to the nearness of the bat to its final state. The frequency of bat varies depending on the number of end (final) states in the state chart diagram. Therefore, fitness function is taken as directly proportional to velocity and inversely proportional to frequency. As for as the Inclination Factor of each state is concerned, minimum value is highly desirable. So, fitness function is chosen as inversely proportional to inclination factor. The specified range lies from 0 to 1. Therefore, IF factor value was subtracted from 1 for optimization. The flowchart for the proposed test suite optimization technique is described in figure 3. The pseudocode for the proposed "BA-TPF" technique for test optimization has been described as:

START
a) Initialize the population of bats as $B_{j=}$ {j=1, 2,3,……..j}
b) Initialize Velocity V_j as 1, Loudness L_j as 100, Location p_j, and frequency [$f_{least=1}$, $f_{highest=10}$] range from 1 to 10
c) While ($j< I_{max}$) where

I_{max}= Maximum no. of Iterations

While (PCT<TT) where TT = Total Transitions, PCT= Presently Covered Transitions by the Bat

C.1 Set COS = SS
Where

COS= Current Optimal State, SS= Starting State

C.2 Calculate out-bounds of the COS where out-bounds=fan-out of State
Populate the number of Bats B_j= Out-bounds of the COS and update the values of V_j, P_j and frequency
C.3 Apply the fitness function and search for the best possible solution in exploitation area with minimum value of the fitness function:
CS= Bat [NS]. PS where PS (Present State) defines the target state of each bat and NS (Next state) defines the count of selected state and CS (Current state) defines the optimal state among every present state of the bats

CS [L_j]= NS [L_j] CS[V_j]= NS [V_j]

Update Frequency of CS as the frequency of NS
C.4 Go to c.2 and repeat the process until COS is the FS (Final State)
C.5 Generate the complete path as a sequence of optimal states.
C.6 end Repeat
END

5) Test Coverage and Path Fitness Measurement:
This Layer deals with measurement and calculations of TPF (Test Path Fitness) values for test suite optimization. Higher the value of TPF, higher is the priority of the test case in a prioritized test sequence. The steps for calculations of TPF is described below:

a) The test sequences generated after application of Bat algorithm are taken as input and value of ASCP (All State Coverage Percentage) Metric and ATCP (All Transition Coverage Percentage) metric is calculated for each generated test sequence or test path [41].
All-State Coverage Percentage (ASCP) Metric: All-state coverage metric can be applied to the test model and full coverage can be achieved when every state of the UML state chart diagram is visited at least once.

ASCP= (No. of States Visited/Total No. of States) *100

All-Transition Coverage Percentage (ATCP) Metric: All-transition coverage metric can be applied to the test model and full coverage is achieved when the test cases visit every transition of the UML state chart diagram at least once.

ATCP=
(No. of Transitions Visited/Total No. of Transitions) *100

b) Path Complexity ($C_{p)}$ for each test sequence is calculated using adding the node complexity of each node in a path using the formulae:

Node Complexity=Node Weight + (Fan-In *Fan-Out)

c) Test Path Fitness (TPF) for each test path is calculated using the following:

Test Path Fitness (TPF) $=[\frac{ASCP \times ATCP}{Cp}] \times \frac{r}{t}$

Where

ASCP= All-State Coverage Percentage (ASCP) Metric Value

ATCP= All-Transition Coverage Percentage (ATCP) Metric

C_p *=Path Complexity*

r=Unique Transition Coverage Constant

t=Repeated Transition Coverage Constant

R is the unique transition coverage constant and its value is decided based upon the number of unique paths (transitions) covered by the test case. Default value of r is set as 1. If a test case covers at least one transition (path) uniquely (i.e. the path is not covered by any other test case), then value of r is taken as 2. t is the repeated transition coverage constant and its value is decided based upon the number of repeated paths (transitions) covered by the test case. Default value of t is set as 1. If a test case covers only those transitions that has already been covered by the prior test cases, then value of t is taken as 2.

d) A prioritized test sequence is generated by arranging the test sequences in descending order of the TPF values. The test sequence possessing the highest value is assigned the highest priority.
e) The performance of the Prioritized test sequence is evaluated using the three metrices namely: TCP (Test Suite Percentage for Complete Path Coverage), PRT (Percentage of Repeated Transitions), APPC (Average Percentage of Path Covered).

4. Conclusion and Future work

This paper proposes a novel "BA-TPF" technique for test suite optimization in regression testing. The architecture/model for the proposed technique contains five layers. The approach has been described in detail using the pseudocode and flowchart. In five-layer proposed model, first four layers collectively produces the test sequences using bat algorithm and fifth layer prioritize these generated test sequences by arranging the test sequences in descending order of TPF values to measure the fitness of the path. Further research will focus on experimental evaluation of the proposed approach against un-prioritized and random order prioritization based on the three metrices namely TCP, PRT, APPC. Future research in this regard will be to comparatively evaluate the performance of the proposed approach against contemporary techniques.

References

[1] B. Beizer. "Software Testing Techniques". Van Nostrand Reinhold, New York, NY, 1990.

[2] G. Myers. "The Art of Software Testing", NY,USA: John Wiley, 1979.

[3] P. Mathur. "Foundations of software testing". China Machine Press, 2008.

[4] Kaner, J. Bach, and B. Pettichord,. "Lessons Learned in Software Testing". John Wiley & Sons, 2008. Paul C. Jorgensen." Software Testing:A Craftsman's Approach", CRC Press, 4th Edition..

[5] W Wong, J. Horgan, S. London and H. Agrawal. "A study of effective regression testing in practice". Proceedings of IEEE Eighth International Symposium on Software Reliability Engineering. pp. 264-274, 1997. https://doi.org/10.1109/ISSRE.1997.630875.

[6] S. Yoo and M. Harman. "Regression Testing Minimisation, Selection and Prioritization : A survey". Journal of software testing , Verification and Reliability, Vol. 22, No. 2, pp. 67-120, 2012. https://doi.org/10.1002/stv.430.

[7] Z. Li, M. Harman and R. M. Hierons. "Search algorithms for regression test case". IEEE Transactions on Software Engineering, San Francisco, CA, USA, pp. 225-237, 2007.

[8] S. Elbaum, A. Malishevsky and G.Rothermel. "Test case prioritization: A family of empirical studies". IEEE Transactions on Software Engineering, Vol. 28, No. 2, pp 159-182, 2002 https://doi.org/10.1109/32.988497.

[9] Ansari, A., Khan, A., Khan, A., & Mukadam, K. (2016). "Optimized Regression Test Using Test Case Prioritization". Procedia Computer Science, vol. 79, pp 152-160. https://doi.org/10.1016/j.procs.2016.03.020.

[10] Suri, B. and Singhal, S. (2015). "Understanding the Effect of Time-Constraint Bounded Novel Technique for Regression Test Selection and Prioritization" .International Journal of System Assurance Engineering and Management, vol. 6, no. 1, pp. 71-77. https://doi.org/10.1007/s13198-014-0244-3.

[11] K. Solanki, Y. Singh, S. Dalal."Test Case Prioritization: An approach based on modified ant colony optimization". Proceedings of IEEE International Conference on Computer, Communication and Control. 2015 Sept; Indore: India .Available at IEEE-xplore Digital Library and SCOPUS.

[12] K. Solanki, Y. Singh, S. Dalal."Experimental Analysis of m-ACO Technique for Regression Testing". Indian Journal of Science and Technology, vol. 9, no. 30, https://doi.org/10.17485/ijst/2016/v9i30/86588.

[13] Mao, C., and Chen, J. (2012). "Generating Test Data for Structural Testing based on Ant Colony Optimization." IEEE International Conference on Quality Software (QSIC), pp. 98-101.

[14] Sharma, B., Girdhar, I., Taneja, M., Basia, P., Vadla, S., & Srivastava, P. R. (2011). "Software coverage: a testing approach through ant colony optimization". International Conference on Swarm, Evolutionary, and Memetic Computing, pp. 618-625. https://doi.org/10.1007/978-3-642-27172-4_73.

[15] Suri, B., and Singhal, S. (2011). "Implementing Ant Colony Optimization for Test Case Selection And Prioritization". International Journal on Computer Science and Engineering, vol. 3, no. 5, pp. 1924-1932.

[16] Srivastava, P., R., and Baby K. (2010). "Automated Software Testing Using Meta-Heuristic Technique Based On An Ant Colony Optimization". International Symposium on Electronic System Design (ISED), pp. 235-240. https://doi.org/10.1109/ISED.2010.52.

[17] Srivastava, P. R., Baby, K. M. and Raghurama, G. (2009). "An approach of optimal path generation using ant colony optimization". IEEE Conference TENCON 2009, pp. 1-6.

[18] Singh, Y., Kaur, A. (2010). "Test Case Prioritization using Ant Colony Optimization". ACM Software Engineering Notes, vol. 35, no. 4, pp. 1-7. https://doi.org/10.1145/1811226.1811238.

[19] Li, K., Zhang, Z., & Liu, W. (2009). "Automatic test data generation based on ant colony optimization" Fifth International Conference on Natural Computation.

[20] Chen, X., Gu, Q., Zhang, X., & Chen, D. (2009). "Building prioritized pairwise interaction test suites with ant colony optimization". Ninth International Conference on Quality Software, pp. 347-352.

[21] Li, H., and Lam, C., P. (2005). "An Ant Colony Optimization Approach to Test Sequence Generation for State Based Software Testing". IEEE International Conference on Quality Software, pp. 255-262.

[22] Berndt, D., and Watkins, A. (2005). "High Volume Software Testing using Genetic Algorithm". 38th IEEE International Conference on System Sciences, pp. 318-330.

[23] Rajappa, V., Biradar, A., and Panda, S. (2008). "Efficient Software Test Case Generation Using Genetic Algorithm Based Graph Theory". IEEE International Conference on Emerging Trends in Engineering and Technology, pp. 298-303.

[24] Krishnamoorthi, R., and Mary, A. (2009). "Regression Test Suite Prioritization using Genetic Algorithms". International Journal of Hybrid Information Technology, vol.2, no. 3, pp. 35-52.

[25] Srivastava, P., R., and Kim, T., H. (2009). "Application of Genetic Algorithms in Software Testing". International Journal of Software Engineering and it's Application, Science & Engineering Research Support Society, Republic of Korea, ISSN: 1738-9984, vol. 3, no. 4, pp. 87-96.

[26] McCaffrey, J., D. (2009). "Generation of Pair-wise Test Sets using Genetic Algorithm". 33rd IEEE International Computer Software and Applications Conference, pp. 626-631.

[27] Jayamala and V. Mohan. "Quality Improvement and Optimization of Test cases- A hybrid genetic algorithm based approach". ACM SIGSOFT Software Engg. Notes, Vol. 35, No. 3, pp. 1-14, 2010 https://doi.org/10.1145/1764810.1764824.

[28] Kaur, A., and Goyal, S. (2011). "A Genetic Algorithm for Regression Test Case Prioritization using Code Coverage". International Journal on Computer Science and Engineering, vol. 3, no. 5, pp. 1839-1847.

[29] Kaur, A., and Goyal, S. (2011). "A Genetic Algorithm for Fault Based Regression Test Case Prioritization". International Journal of Computer Applications, vol. 32, no. 8, pp. 975-987.

[30] Rathore, A., Bohara, A., Prashil, R. G., Prashanth, T. S., & Srivastava, P. R. (2011). "Application of genetic algorithm and tabu search in software testing". Proceedings of the Fourth Annual ACM Bangalore Conference, pp 23. https://doi.org/10.1145/1980422.1980445.

[31] Singhal, S., Gupta, S., Suri, B., & Panda, S. (2016). "Multi-deterministic Prioritization of Regression Test Suite Compared: ACO and BCO". Advanced Computing and Communication Technologies, pp. 187-194.

[32] Dalal, S., and Chillar, R. (2013). "A Novel Technique for Generation of Test Cases Based on Bee Colony Optimization and Modified Genetic Algorithm". International Journal of Computer Applications, vol. 68, no. 19, pp. 12-17. https://doi.org/10.5120/11687-7359.

[33] Kaur, A., and Goyal, S. (2011). "A Bee Colony Optimization Algorithm for Code Coverage based Test Suite Prioritization". International Journal of Engineering Science and Technology, vol. 3, no. 4 pp. 2786-2795.

[34] Kulkarni, N., Singh, P., and Srivastava, P., R. (2011). "Test Case Optimization using Artificial Bee Colony Algorithm". Advances in Computing and Communications, Springer Berlin Heidelberg, pp. 570-579.

[35] Dahiya, S., S., and Chhabra, J., K. (2010) "Application of Artificial Bee Colony Algorithm to Software Testing". Australian Software Engineering Conference, pp. 149-154.

[36] Jeyamala, D., and Mohan, V. (2009). "ABC-Artificial Bee Colony Optimization based Test Suite Optimization Technique". International Journal of Software Engineering, vol. 2, no. 2, pp. 1-33.

[37] Öztürk, M. M. (2017). "A bat-inspired algorithm for prioritizing test cases". Vietnam Journal of Computer Science, 1-13.

[38] Srivastava, P. R. (2017). "Path Generation for Software Testing: A Hybrid Approach Using Cuckoo Search and Bat Algorithm". In Nature-Inspired Computing and Optimization , 409-424. Springer International Publishing. https://doi.org/10.1007/978-3-319-50920-4_16.

[39] Srivastava, P. R., Bidwai, A., Khan, A., Rathore, K., Sharma, R., & Yang, X. S. (2014). An empirical study of test effort estimation based on bat algorithm. International Journal of Bio-Inspired Computation, 6(1), 57-70. https://doi.org/10.1504/IJBIC.2014.059966.

[40] Srivastava, P. R., Pradyot, k., Sharma, D., Gouthami, K. P. (2015). Favourable test sequence generation in state base testing using bat algorithm. International Journal of Computer Applications in Technology, 51(4), 334-343. https://doi.org/10.1504/IJCAT.2015.070495.

[41] Oluwagbemi, O. and H. Asmuni (2015), Automatic Generation of Test Cases from Activity Diagrams for UML Based Testing. Jurnal Teknologi,77(13).

[42] Yang, X.S. (2010) A New Metaheuristic Bat-Inspired Algorithm, Nature Inspired Cooperative Strategies for Optimization, Spain.

[43] Satapathy, S. C., Raja, N. S. M., Rajinikanth, V., Ashour, A. S., & Dey, N. (2016). "Multi-level image thresholding using Otsu and chaotic bat algorithm". Neural Computing and Applications, 1-2

[44] Dalal, S., & Solanki, K. (2018). "Performance Analysis of BCO-m-GA Technique for Test Case Selection". Indian Journal of Science and Technology, 8(1), pp1-10. https://doi.org/10.17485/ijst/2018/v11i9/110048.

[45] Dalal, S., Chhillar, R. (2013) "Empirical study of root cause analysis of software failure". ACM SIGSOFT Software Engineering Notes. 38(4):1–7. https://doi.org/10.1145/2492248.2492263.