IOSR Journal of Engineering (IOSRJEN) **www.iosrjen.org**
ISSN (e): 2250-3021, ISSN (p): 2278-8719
Vol. 08, Issue 11 (November. 2018), ||V (II) || PP 51-59

Comparative Analysis of Regression Test Case Optimization Techniques

Sandeep Dalal[1], Sudhir[2]
1Assistant Professor, M.D.University, Rohtak
2Research Scholar, M.D.University, Rohtak
Corresponding Author: Sandeep Dalal1

Abstract: Viability of testing remains lower than desires, even after the extensive research to develop efficient testing techniques to enhance the quality of the product. Although testing is very costly and time -consuming activity, still there is no substitute for testing as it is the only way to achieve software quality. It is not feasible to completely and exhaustively test a software due to cost and time constraints. Therefore, even after usage of maximum efforts and efficient testing techniques, there is no guarantee of producing a bug-free software. Regression testing is a necessary evil associated with software testing as it ensures that modifications in software do not introduce fresh bugs in system. Optimization in regression is highly solicited as it is not feasible to completely run the entire test suite for regression. Many regression test case optimization techniques have been proposed by researchers in recent past. This paper makes a comparative analysis of Genetic Algorithm (GA), Bee Colony Optimization (BCO), Ant colony Optimization(ACO) and Bat Algorithm (BA) in terms of path coverage, iterations and execution time.

Keywords: - Regression Testing, Software Testing, Test Case Optimization, Test Case Prioritization

---

Date of Submission: 11-11-2018 Date of acceptance: 23-11-2018

I. INTRODUCTION

Software has become an integrated and in-separable part of human life. Every aspect of human life has become inherently dependent on software's and this dependency is constantly increasing from past few decades. In Modern society, one cannot imagine about living without software's as software has massively invaded into every facet of human life. Dissemination of software in all varied areas in the past decades has built pressure on software organizations to produce quality software's. The most significant prerequisite of a quality software product is to fulfil the customer's expectations [1][2][3][4]. In present ever-changing business atmosphere, delivery-time of a software is also a prime factor to measure the success of a software product. Consequently, a software which possess good quality and has been developed with minimum cost and minimum delivery time will be considered as successful. Software quality concerns are fairly wide including correctness, robustness, readability, reliability, changeability etc. [5][6]. Moreover, customer satisfaction is one of the major quality indicators as quality can be measured by the level up to which a software meets customer expectations. Actually, a software quality can be perceived from producer's as well as customer's perception. Actual software quality delivered may have "producer gap" and "customer gap" as shown in fig.1.1. A "producer gap" defines the deviation of delivered quality from the specifications of the producer. A "customer gap" defines the deviations of delivered quality from the customer's expectations [6]. Delivered software quality with minimized customer and producer gap is highly desirable. A potential usage of software testing lies in minimizing the gaps by effectively utilizing optimized software testing techniques**.** Software testing is an iterative and costly phase of software development that generally spans over from requirement gathering up to maintenance phase [6]. In-sufficient and in-effective testing is one of the major causes of software failures as shown in fig. 1.2.

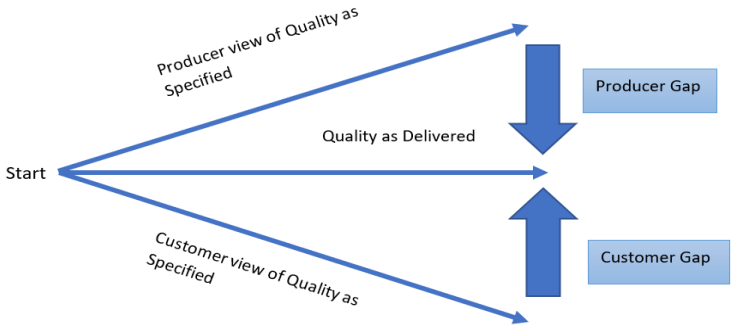


Figure 1.1: Software Quality Gaps

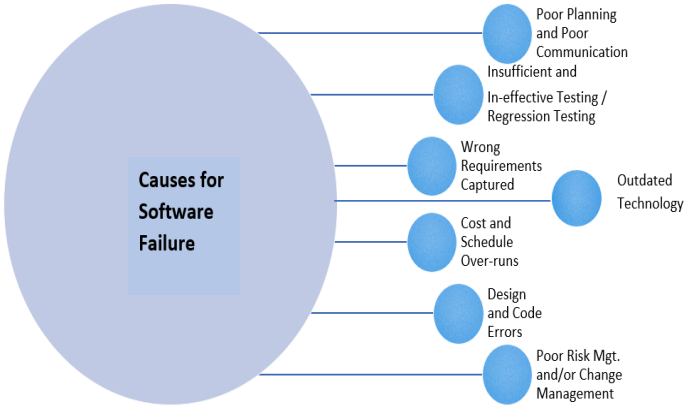


Figure 1.2: Causes for Software Failures.

II. SOFTWARE TESTING

The development of software is not only confined to just writing software code. Basically, software development is a long process having number of phases and each phase is very crucial in itself. Software requirements are captured by requirement analyst and are documented in the form of SRS (Software Requirement Specification). A software is designed based on the specifications of SRS document and finally code is written. After the software has been coded, it is tested thoroughly to find the failures (or bugs). A software failure is actually a fault that gets triggered during testing and a fault is propagated into the software system as a result of some human mistake or error as depicted in fig. 1.3.

Any human mistake is an Error. An effective testing method should be able to capture and detect the "Error" at the earliest. When an error gets propagated at the next level, it becomes a fault (or defect). "A fault is an observed deviation from the specified functionality of a software system" [7]. This fault is revealed when test cases are executed. The system will produce incorrect results revealing a failure or bug. Therefore, when a fault is materialized/ triggered or activated while testing; it is known as a software failure or bug. The significance of software testing can be perceived using Pareto Principle for software testing which describes that almost 80% of the software bugs (or failures) originate from 20% software modules [8][9][10]. This is known as 80:20 rule for software testing as described in fig. 1.4.

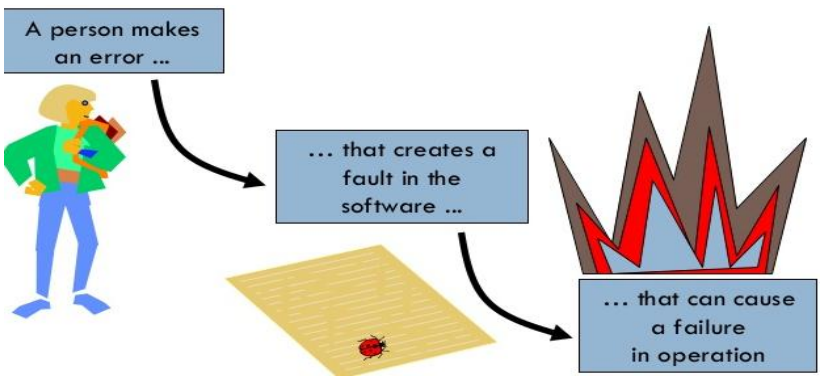


Figure 1.3: Error Propagation as Fault and Failure

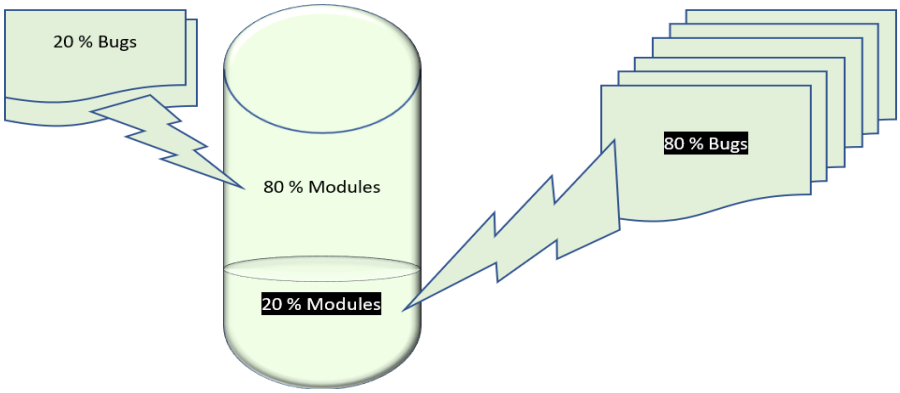


Figure 1.4: Pareto Rule for Software Testing

Therefore, all the software modules are not equally buggy as per the "Pareto Rule". It can be concluded that software bugs are not uniformly distributed among various software modules. Some modules have higher density of software bugs in comparison to other modules. Hence, rigorous testing/re-testing of the modules having higher density of bugs must be assured to achieve software quality. The software bugs captured during software testing phase are then fixed by the developers by altering the software code.

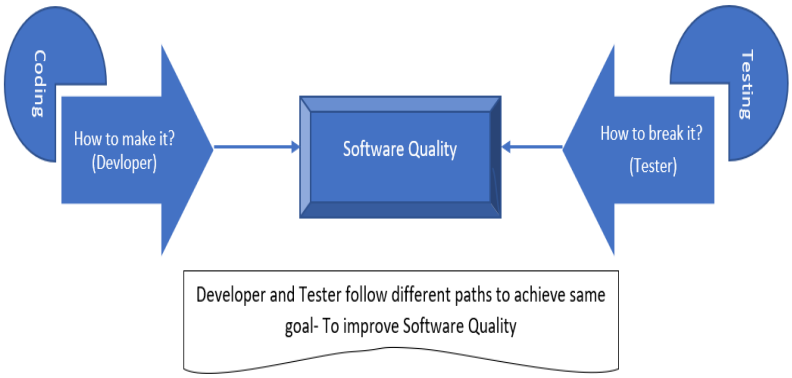


Figure 1.5: Different Paths for Software Quality

The main task of the tester is to find maximum number of bugs by demonstrating that system is not working as desired. Moreover, it is generally observed that a tester tries to break the system while a developer tries to make the system. However, the ultimate goal of both the tester as well as developer is to produce a quality software as shown in figure 1.5. The code for the software is constantly changed by the developer to fix the bugs reported by the testing team as shown in fig. 1.6.

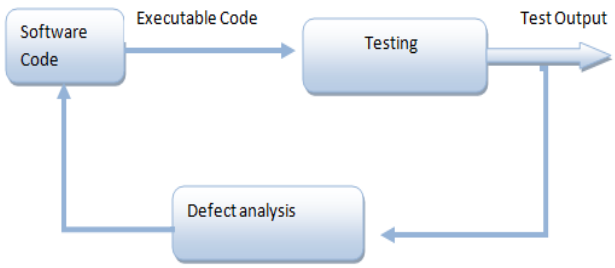


Figure 1.6: Software Testing Process

A tester designs and executes the test case to capture the software bugs. "A test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies user requirements and works as intended". A test case is a set of input condition that measures the deviation of expected behavior of the system with observed behavior. A test suite is a collection of test cases. A test case is

either marked as "passed" or "failed" after execution. The test case whose expected and observed output is same will be marked as "passed". The test cases which "fail" depicts the presence of bugs. Failed test cases are reported to the developers for fixing the defects (or bugs) as shown in fig. 1.7.

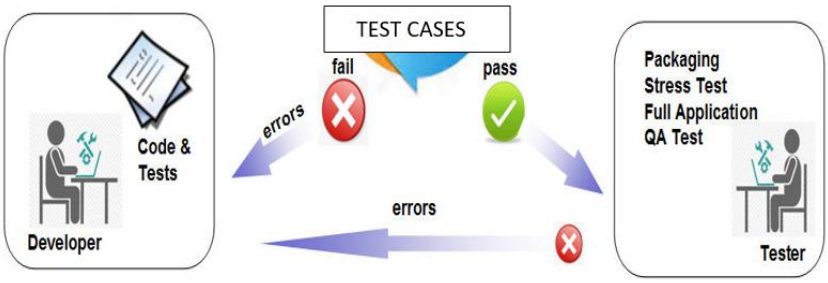


Figure 1.7: Detailed Software Testing Process

Among all the phases of software development, the maximum efforts and cost is incurred during software testing. Software testing is inherently iterative, expensive, time consuming, utmost complex and most significant activity of software development. These characteristics make testing an open research as in-effective and in-efficient testing may lead to production of poor quality and un-reliable software. An un-reliable software requires high maintenance cost and un-necessary repetitions of efforts. Software complexity can be simply defined by the statement "it is nearly impossible to completely test software i.e. exhaustive testing of software is not possible". DeLay in finding a defect or bug (or failure) enhances the price of rectifying a software defect many-fold. Therefore, it is cost effective to catch a defect as soon as it enters in the system. Cost of finding and fixing a defect is minimum, in case a defect is captured as soon as possible. A quality software cannot be produced without adequate and efficient software testing methods. Henceforth, the key objective of the software organization is to pay attention towards using optimized and most efficient software testing techniques to build quality software within stipulated time and cost. A successful software product is the one that is developed with minimum expenditure. Further, a successful software product is delivered on time and fulfills the customers' expectations**.** Therefore, it is evident that quality of a software highly depends on the effectiveness and efficiency of software testing methods and techniques. "Software testing carry out a significant role for quality assurance and make sure the reliability of software"

III. REGRESSION TESTING

Software code keeps on changing as the defects are fixed by the developers. Every time a defect gets fixed, the software code is altered and updated. Moreover, the requirements of the end-user cannot be considered as fixed in most of the cases and these ever-changing user requirements also contribute towards continuous changes in code. Regression Testing is also an essential testing that need to be performed after the amendments are incorporated in the software code to ensure that these amendments do not adversely affect rest of the software system. One can conclude that changes/ amendments in the software code are inevitable. Hence, Regression testing is a necessary evil because regression testing is extremely expensive, time intensive and iterative phase of software testing that must be carried out after every change in software code[9][10][11][12].

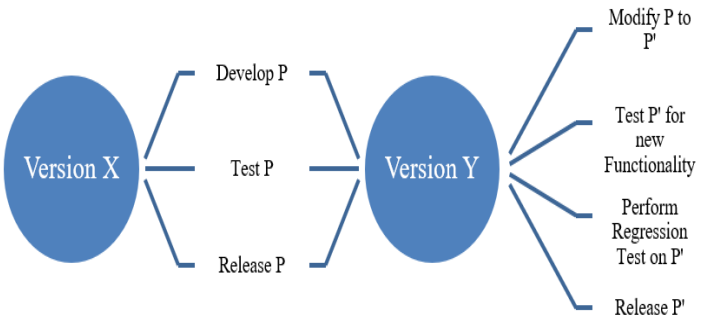


Figure 1.8: Regression Testing Process

Fig. 1.8 depicts the regression testing process by taking an example of any software program P, which has been released as version X. After P gets released, any kind of corrective, adaptive or perfective maintenance

will change the software code leading to the newer version of software program P' (release version Y). Thereafter, this modified software program P' undergoes rigorous testing again to as a safety measure so that the accurate functioning of newly added functionality can be assured. Moreover, this re-testing of software program P' ensures that the change in code has not affected rest of the system and the newer defects have not been introduced in system. Therefore, running the regression tests cross verifies that all kinds of malfunctioning of the updated software program P' gets detected and fixed before P' is released as version Y.

Regression testing can be implemented in various ways. A software developer can re-execute the test cases of the original test suite which was executed on the earlier version of the software system; this method is often denoted as "retest-all approach"[10][11][12]**.** However, the limitation of this approach is that it is impractical to re-execute all test cases again and leads to useless efforts as modification affect only a portion of a system. Some test suites are extremely large, therefore cost and time constraints do not permit re-execution of all test cases in a test suite to carry out regression tests. Therefore, developers may use any of the following options:

- Test Case Selection: "Run only a subset of the test suite, which can be chosen by using regression test selection techniques".
- Test Case Minimization: "Permanently reduce the number of test cases by eliminating redundant test cases from the test suite".
- Test Case Prioritization: "Re-schedule the execution sequences of the test cases within the test suite in a particular order to maximize some goal"

All these techniques for selective re-testing are known as regression test case optimization techniques. Running entire set of regression test suite after every change makes it a cumbersome take that requires cost, time and efforts repetitively. Moreover, time and cost constraints always create barrier in running the entire test set repeatedly to carry out regression testing. Keeping in view the complexity and iterative nature of regression testing, there arises a need of minimized and optimized set of set cases that can perform the task of regression testing without compromising the efficiency and efficacy of regression testing. To efficiently handle the quickly changing user requirements of modern-day large scale and complex software systems, there is an urgent need of efficient and effective regression testing paradigms that can reduce overall cost, time, efforts and assists in enhancing software quality.In this regard, this paper makes a comparative analysis of GA, ACO, BCO and BA based test case prioritization.

IV. COMPARATIVE ANALYSIS OF TEST CASE OPTIMIZATION TECHNIQUES

This section makes a comparative analysis of various GA, BCO, ACO and BA based test case optimization techniques on various parameters like path coverage, iterations and execution time. The experiment has been conducted on six programs written in java language for automated test data generation and optimization. Summary of six programs can be described as:

P1: Calendar Problem- "This program takes date, month and year as input and returns the day as output."

P2: Triangle Problem- "The program checks whether the sides of triangle given as inputs form a triangle or not. If they form a triangle, then it classifies the triangle as isosceles, equilateral or scalene."

P3: Quadratic equation Problem- "The program accepts three inputs and checks whether they can form a quadratic equation or not. If they form a quadratic equation, then it also finds the roots of equation."

P4: Currency Converter Problem- "The program takes U.S. dollars as input and convert them into various other currencies."

P5: Point Location Calculation Program: "This program accepts the coordinate of a point in x-y Cartesian plane and also the radius of circle whose center is at origin. It then checks whether the point lies inside, outside or on the boundary of the circle".

P6: Palindrome Problem – "This program takes a word as input and checks whether the entered input is same from reverse direction or not."

Many researchers have applied ACO [13][14][15][16][17][18][19][20][21][22][23], GA [24][25][26][27][28][29][30][31][32], BCO [33][34][35][36][37][38][39], BA [40][41][42][43][44][45][46] for test case optimization. This section makes a comparative analysis of these optimization approaches.

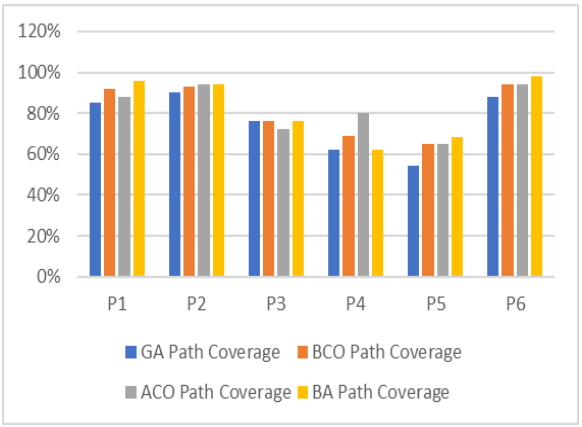
Table 1.1: Comparative Table of GA, BCO, ACO and BA Algorithm

GA			
Sr. No.	Path Coverage	Iterations	ET
P1	85%	4	1.35
P2	90%	6	2.34
P3	76%	3	0.97
P4	62%	9	3.4
P5	54%	4	4.8
P6	88%	7	2.76

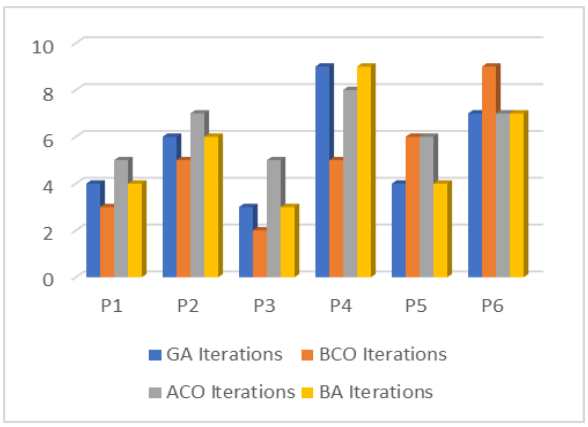
BCO			
Sr. No.	Path Coverage	Iterations	ET
P1	92%	3	1.29
P2	93%	5	1.9
P3	76%	2	1.2
P4	69%	5	3.4
P5	65%	6	4.3
P6	94%	9	2.5

ACO			
Sr. No.	Path Coverage	Iterations	ET
P1	88%	5	1.32
P2	94%	7	2.20
P3	72%	5	0.95
P4	80%	8	3.7
P5	65%	6	5.2
P6	94%	7	2.6

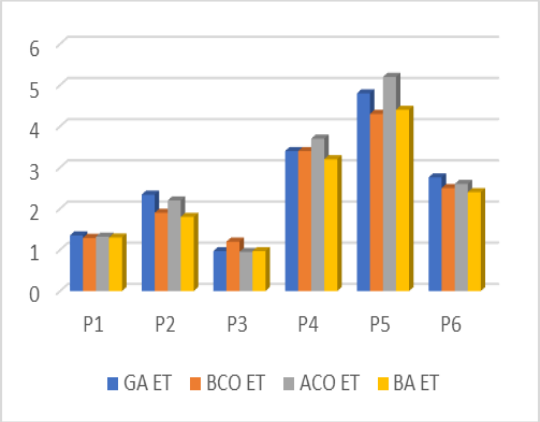
BA			
Sr. No.	Path Coverage	Iterations	ET
P1	96%	4	1.30
P2	94%	6	1.8
P3	76%	3	0.97
P4	62%	9	3.2
P5	68%	4	4.40
P6	94%	7	2.4



Graph 1.1: Comparative Path Coverage for Optimization Algorithms



Graph 1.2: Comparative Iterations for Optimization Algorithms



Graph 1.3: Comparative Execution Time for Optimization Algorithms

From table 1.1 and graph 1.1,1.2 and 1.3, it can be concluded that overall performance of Bat Algorithm is better in terms of path coverage, number of iterations needed for execution of algorithm and time taken for automated optimized test data generation as compared to BCO, ACO and GA. After that, BCO outperforms GA and ACO.

V. CONCLUSION

This paper makes a comparative analysis of various regression test case optimization techniques based upon nature inspired algorithms like GA, ACO, BCO and BA etc. The various techniques for regression test optimization were compared based on three parameters namely execution time, no. of iterations and percentage of path covered in terms of path coverage using 6 programs written in java language. The results of analysis proved that performance of bat algorithm either outperforms other techniques or works equally well.

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).
[5]. Paul C. Jorgensen. Software Testing:A Craftsman's Approach (CRC Press, 4th Edition)..
[6]. Dalal, S., Chhillar, R. (2013) "Empirical study of root cause analysis of software failure". ACM SIGSOFT Software Engineering Notes. 38(4):1–7
[7]. 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.1997 , 264-274.
[8]. S. Yoo and M. Harman. "Regression Testing Minimisation, Selection and Prioritization : A survey". Journal of software testing , Verification and Reliability,22 (2),2012, 67-120.
[9]. Z. Li, M. Harman and R. M. Hierons. "Search algorithms for regression test case". IEEE Transactions on Software Engineering, San Francisco, CA, USA, 225-237.
[10]. S. Elbaum, A. Malishevsky and G.Rothermel. "Test case prioritization: A family of empirical studies". IEEE Transactions on Software Engineering, 28(2),2002, 159-182.
[11]. Ansari, A., Khan, A., Khan, A., and Mukadam, K. (2016). "Optimized Regression Test Using Test Case Prioritization". Procedia Computer Science, 79, 152-160.
[12]. 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, 6(1), 71-77.
[13]. 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
[14]. K. Solanki, Y. Singh, S. Dalal."Experimental Analysis of m-ACO Technique for Regression Testing". Indian Journal of Science and Technology, 9(30).
[15]. Mao, C., and Chen, J. Generating Test Data for Structural Testing based on Ant Colony Optimization. IEEE International Conference on Quality Software (QSIC),2012, 98-101.

[16]. Sharma, B., Girdhar, I., Taneja, M., Basia, P., Vadla, S., & Srivastava, P. R. Software coverage: a testing approach through ant colony optimization. International Conference on Swarm, Evolutionary, and Memetic Computing,2011, 618-625.
[17]. Suri, B., and Singhal, S. "Implementing Ant Colony Optimization for Test Case Selection And Prioritization". International Journal on Computer Science and Engineering, 3(5), 2011, 1924-1932.
[18]. 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.
[19]. 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.
[20]. Singh, Y., Kaur, A. (2010). "Test Case Prioritization using Ant Colony Optimization". ACM Software Engineering Notes, vol. 35, no. 4, pp. 1-7.
[21]. Li, K., Zhang, Z., & Liu, W. (2009). "Automatic test data generation based on ant colony optimization" Fifth International Conference on Natural Computation.
[22]. 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
[23]. 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.
[24]. Berndt, D., and Watkins, A. (2005). "High Volume Software Testing using Genetic Algorithm". 38th IEEE International Conference on System Sciences, pp. 318-330.
[25]. 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.
[26]. 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.
[27]. 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.
[28]. McCaffrey, J., D. (2009). "Generation of Pair-wise Test Sets using Genetic Algorithm". 33rd IEEE International Computer Software and Applications Conference, pp. 626-631.
[29]. 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
[30]. 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.
[31]. 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.
[32]. 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.
[33]. 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.
[34]. 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.
[35]. 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.
[36]. 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.
[37]. Dahiya, S., S., and Chhabra, J., K. (2010) "Application of Artificial Bee Colony Algorithm to Software Testing". Australian Software Engineering Conference, pp. 149-154.
[38]. 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.
[39]. 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.
[40]. Öztürk, M. M. (2017). "A bat-inspired algorithm for prioritizing test cases". Vietnam Journal of Computer Science, 1-13.

[41]. 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.
[42]. 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.
[43]. 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.
[44]. Oluwagbemi, O. and H. Asmuni (2015), Automatic Generation of Test Cases from Activity Diagrams for UML Based Testing. Jurnal Teknologi,77(13)
[45]. Yang, X.S. (2010) A New Metaheuristic Bat-Inspired Algorithm, Nature Inspired Cooperative Strategies for Optimization, Spain.
[46]. 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

Sandeep Dalal. " Comparative Analysis of Regression Test Case Optimization Techniques." IOSR Journal of Engineering (IOSRJEN), vol. 08, no. 11, 2018, pp. 51-59.