[1]Vivek Kulkarni,

Dr. R. Madhan Mohan,

Dr. H. Venkateswara Reddy

Butterfly Algorithm Approach for Test Case Prioritization

Journal of Electrical Systems

Abstract: **-** Regression testing is the costly, time-consuming and critical testing activity that guarantees the improvements made in code have not changed features already in use. After any code update, running the entire test suite is just an infeasible task due to time and effort constraints. To achieve the early fault detection objective, the test case prioritization technique encourages deciding on the test case execution sequence. The optimization of software testing is still an important task, as the average percentage of detected failures (APFD), the average percentage of Branch Coverage detection (APBCD), and output of time are unsatisfactory in priority test cases. In this paper, we had proposed the Butterfly algorithm for test case optimization. We use Butterfly Algorithm with a fitness function specified with a similitude-distance model to optimise the ordering of test cases. Three testing suites selected from the software testing case repository were experimented with within 3 benchmarking programmes. Our Test Case Prioritization technique (TCP) was seen well as current works with the Butterfly APFD Algorithm as the output matrix. Overall APFD results show Butterfly Algorithm being a successful competitor in TCP applications.

.***Keywords:*** Test Case Prioritization, Regression Testing, APFD, APBCD, Butterfly algorithm

INTRODUCTION:

Software testing requires a long-running time and can be the costliest step of the software development process[1]. The checking of applications is understood as the least comprehensive aspect of the development process. Also, testing of applications is done over and over again, because of time limitations and resources it is often done in hurry. In light of this, the Test Case Prioritization application (TCP) has been stated to increase test viability in software tests [2][3][4][5]. First stated by [6] is the priority test case strategy. However, this work has only prioritised test cases that have been selected. The method was suggested and tested in a more widespread sense by two scholars, Rothermel and Harold. An example of a Test Suite is defined in Table 1[7].

[1] Department of Computer Science and Engineering
Annamalai University, India
vivekdk1971@gmail.com
[2]Department of Computer Science and Engineering
Annamalai University, India
madhanmohan_mithu@yahoo.com
[3]Vardhaman College of Engineering
Hyderabad, India
h.venkateswarareddy@vardhaman.org

Table 1: Test Suite Example

Fault → / TC ↓	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
T1	*							*	*	
T2	*					*	*			
T3			*	*	*				*	*
T4		*	*	*						
T5					*		*	*		*

TCP attempts to order several test cases based on preferred properties for early optimisation [2],[8]. It is also difficult in real-life challenges to assess which measurements potentially show defects. The priority measures for the test case, therefore, rely on various approaches and are expected to improve earlier fault exploration in the early intensification of a certain process. The test case priority method has multiple aspects. Having been defined by [9] as eight wide dimensions. The TCP techniques were rooted in selection procedures, input and output sort based on their commonalities. TCP can be accomplished by using string metrics that can differentiate between test cases in terms of character deviations and the priority of a pre-set fitness function. The present study in recent years seems to have prioritised test cases using only test case-generated information [5],[10],[11]. Software testers, for example, can prioritise test cases before device source code is accessible by using available information such as test case feedback. This lowers the evaluation time total. Right after the design process TCP can be started. There is space for progress in time compliance[4],[5], among the various TCP strategies suggested. Not only is this issue common for non-artificial intelligence applications in TCP, but also for most TCP applications for Artificial Intelligence (AI) that need improvement in particular for APFD and implementation time, as recent studies have pointed out [3], [5], [12],[13].

We can see from recent research that several issues exist and are worth looking at. An AI methodology is suggestions and recent analysis indicates that the optimization algorithm may have important results both for an increase in the outcome of APFD and for time running. The result is, "which optimization algorithm will be best for TCP background" as a macro research issue or concern for this article. Since Butterfly Algorithm was not explored in TCP, the algorithm was chosen by the authors.

Code coverage is a software testing measure, also known as Code Coverage Testing, that is used to determine how much code in a source is tested. This metric is useful in assessing the quality of a test suite and determining how thoroughly a programme is validated. The degree to which the software code source code was tested relates to the basic code coverage. This Code Coverage is one of the white box testing forms.

Knowing that quality software products are desired by each client after development, it is also understood that the developer team is likewise accountable for providing quality software products to the customer/client. The performance, functionality, and behaviour of a product are all referred to as quality. Other qualities include accuracy, dependability, effectiveness, security, and maintainability, among others. When it comes to software, the Code Coverage measure is useful in measuring the performance and quality of the product.

The objective of generating test cases is distinctive from TCP, whereby the prioritization of test cases is started in the test case generation first and then the appropriate test cases are selected. In the meanwhile, TCP just covers prioritising test cases, based on the test case order by a predefined objective. The work suggests the implementation by optimising the most efficient test sequences of Butterfly Algorithm in applicable software test procedures. Motivated by the proposal, it is necessary to retrieve and rewrite the schematics of the Butterfly schematic algorithm that are applied in the work for TCP use. Consequently, this paper's thesis goals were as follows.

- To apply Test Case prioritization using the Butterfly algorithm which is not mentioned and used in all previous work.

- The Butterfly algorithm is to be used in four separate string metric benchmarking programs.
- Comparison of results with current work to APFD and APBCD and implementation time performance.

The remaining paper is broken up in the given section. Section 2 outlines work in TCP with various algorithms for optimization. Section 3, describes an overview of the Butterfly algorithm. Section 4 discusses the experimental application for a TCP problem using the Butterfly algorithm. Section 5 focuses on the analytical assessment, conducted with findings and discussion of the proposed study compared to the previous works. Finally, Section 6 concludes.

LITERATURE REVIEW:

The advancements in Regression Testing gained the attention of several researchers to focus more on Test Case Prioritization (TCP). The technique based on TCP appears to be a promising technique for prioritizing the test suite to effectively minimize time and early detection of faults without neglecting the reliability and quality of the application. TCP promotes fault aware engineering principles for designing effective prioritization of test cases of AUT. TCP discovers rearranging test cases that provide the maximum benefits to the application under test. TCP approach evaluates the priority of every test case based on some performance goal to confirm that the test cases with higher priorities are executed before the other test cases to early detection of maximum faults in the testing process. The fault detection rate entirely depends on how effective the test cases are sequences in their execution ord1er. The effectiveness of TCP techniques is quantified by the rate of fault detection per execution of test cases metric known as Average Percentage of Faults Detected (APFD).

Panwar D. et al [14] suggest in their paper to close the test case optimally in a confined setting, the Cuckoo Search (CS) algorithm, which is supported by the Modified Ant Colony Optimization (M-ACO) algorithm. The CS algorithm is inspired by the constraint in other hosts nests of brood lethargy and egg-laying. Since this algorithm depends on the parameters, cuckoo search is efficient and simpler to apply compared to other optimised algorithms. However, the hybrid Cuckoo-ACO optimisation approach is best for selecting the test case and setting priorities, according to the proposed empirical research. The inconvenience of this approach is the time and expense limitation.

In his article, M. Khatibsyarbini et. al. [15] proposed that priority test cases could be used for optimum use of the Firefly algorithm. They use the Firefly algorithm to optimise the ordering of test cases with a fitness equation with a similarity distance model. It was observed that after experimenting the algorithm with three test suites, the Test Case Priority (TCP) technic with a similarities-length model for the Fireflies algorithm demonstrated a superior, if not equivalent, APFD approach. Overall results of APFD suggest that in contrast with other TCP optimisation algorithms Firefly algorithms perform better.

Harikarthik S. K. et al. [16] (2019) investigated a hybrid artificial neural network (ANN) – whale optimization algorithm(WOA) approach for prioritizing test cases using the fault detection history of test cases. The algorithm starts with the generation of test cases that undergo clustering to classify them into related and unrelated test cases. This classification is carried out using a modified kernel fuzzy c means (MKFCM) algorithm. The related test cases further undergo TCP by the whale optimization algorithm approach. The objective of TCP is to generate the test case scheduling that enhanced the chances of discovering defects in source code early. MANN classification algorithm is used for prioritization. TCP is further optimized using WOA which calculates the weight of each test case. Calculated weight is used as a fitness value to prioritize the test cases. The performance of the presented hybrid approach is evaluated using the Average Percentage of Faults Detected (APFD), memory and execution time metrics. The result shows that ANN-WOA outperforms compared to traditional neural network algorithm.

Maheswari and Mala [17] (2015) presented a hybrid combination of genetic algorithm (GA) and Simulated Annealing (SA): GASA demonstrates the benefits of both algorithms. The test cases are assigned priority using the fault detection capability of the test case and its execution time. It uses rapid processing along with exploration over large data suite aspects of GA and the effectual local solution aspect of SA. In the first step, the solution is generated randomly which is then evaluated for fitness function on the rate of fault detection and execution time parameters.

After calculating the fitness score, each chromosome is refined locally by the SA approach and a new population is generated. The proposed GASA technique is evaluated on a manual test case fault matrix containing seven test cases and 11 faults along with the execution time of each test case. APFD is calculated to evaluate the performance of GASA compared to GA and SA algorithms. Results depict that GASA outperforms compared to GA and SA.

Reddy V. and Reddy R. M. [18] (2016) explored a single objective-based hill-climbing optimization algorithm for solving the TCP problem. Implemented approach reorders the test cases using knowledge of code coverage capability and execution time of each test case. For calculating code coverage data by each test case, the Java-based Emma software tool is used. Emma captures the underline statements after the execution of each test case and stored them in a dataset. Test cases covering the maximum number of code statements and utilizing minimum test execution time are assigned higher priority and are ranked before lesser priority test cases. The author introduces a mutation factor in the hill-climbing algorithm which randomly muted the bits of the test case execution sequence.

Gao Det al. [12] (2015) demonstrated a multi objective-based TCP approach using an ant colony optimization (ACO) algorithm. Factors considered for prioritization are the Number of faults detected by a test case, test case execution time and Fault severity. Quantification index is introduced for analysing the fault impact on test case prioritization in the form of fault severity for each test case. Fault severity is classified into levels ranging from trivial level to critical level, and different severity values are assigned to different levels. The objective of prioritization is to order test cases covering maximum severe faults before other non-severe faults covering test cases. ACO algorithm optimizes the test case execution sequence using the pheromone updating rule and test case node selection rule. The efficacy of ACO is calculated using the APFD metric and is compared with the random, original and optimal ordering of TC execution. The experiment is computed on a manual test case- fault matrix comprising of 8 test cases and 10 faults. Result clearly shows the improvement of ACO over other approaches.

Ashraf E. et al. [19] (2017) implemented a novel TCP approach using particle swarm optimization (PSO) with the goal of early fault detection by minimum execution of test suite. The approach uses six parameters for priority calculation. Customer assigned priority, code implementation complexity and requirement instability parameters used for calculating the score of new test cases. Requirement traceability, fault impact of requirement and execution time parameters are used to calculate the score of existing test cases. The performance aim is to set a priority of the test cases to the new best positions to reveal maximum defects earlier in regression testing. Average percentage of fault detection (APFD) metric used for validation of the effectiveness of the presented PSO based prioritization.

OVERVIEW OF BUTTERFLY ALGORITHM:

A new optimization approach to mimic the feed behaviour of butterflies is discussed in this article. Any biological details are discussed in the following subsections to explain this latest algorithm and how they can be modelled in BOA.

Butterflies have a very specific sense of the scent source based on laboratory experiments [20]. You can discern and sense the strength of different fragrances [21]. The quest agents of BOA are butterflies for optimisation. If a butterfly flies from place to place, it can vary depending on its health. The butterfly's health is connected with exercise. The fragrance is transmitted over time, and it can be felt by butterflies so that they can share their information with other butterflies and form a shared social knowledge network. If a butterfly detects the smell from a certain butterfly, it is moved and named a global search in the suggested algorithm. When a butterfly in another case doesn't detect the fragrance of its surroundings, it spontaneously shifts and this step is called a local search in the algorithm.

Per fragrance has its distinctive smell and personal feel in the Butterfly Optimization Algorithm (BOA). This separates BOA from other metaheuristics. It is assumed in BOA that each butterfly produces fragrance with some intensity, which indicates the fitness of butterflies. During the movement of the butterfly is from one place to another, its fitness varies accordingly. The fragrance is a sense by other butterflies in the same region and this is the way that butterflies can share their personal information and create a network of social information.

There are three important terms stimulus intensity (SI), sensory modality (Sc) and power exponent (P) where SI represents the magnitude of the actual stimulus. In BOA, SI is mapped with the fitness of the solution. So if a

butterfly emits more fragrance then others will more attractive to her. Butterflies have a natural phenomenon that is based on two important factors: first is variation in SI and accordingly formulation of bf. The stimulus intensity SI of a butterfly is associated with the objective function. Here, bfr is a butterfly relative parameter and other butterflies sensed the same. In BOA, the butterfly fragrance [22] is as follows

$$bf_{ri} = Sc(SI)^a \qquad (1)$$

where bf_{ri} is the magnitude of fragrance, SI, Sc, and a parameter are dependent on modality. This algorithm combines two main steps: (1) the Global search step and (2) the Local search step. Under global search step, the butterfly move towards the fittest butterfly bg* as follows:

$$S_i^{t+1} = S_i^t + (r^2 * bg^* - S_i^t) * bf_{ri} \qquad (2)$$

where S_i^t is the solution vector si for ith butterfly in iteration number t and bg^* shows the best-known solution for the current state found among all the solutions in the current stage. The fragrance of ith butterfly is represented by bf_{ri} and r is a random number in between 0 and1. The second step, the Local search step is represented by:

$$S_i^{t+1} = S_i^t + (r^2 * S_k^t - S_i^t) * bf_{ri} \qquad (3)$$

where s_i^t and $s_k^t$are j^{th} and k^{th} butterflies taken randomly from the solution space. If x_j^t and x_k^t belongs to the same crowd and r is a random number in between 0 and 1, so the equation (3) performs a local random walk. Here is a probability p is utilized to main the balance between local and global search in BOA. The above-mentioned steps are used to form Butterfly Optimization Algorithm and its stepwise pseudocode is given in as follows.

Butterfly Optimization Algorithm

1) Formulation the Objective function f(s), s= (s1, s2, sdim) where dim is the no. of dimensions.
2) Generation of initial population of n butterflies si= (i=1,2,3,4, … n)
3) Define a, p and c parameters.
4) While stopping criteria not met do
5) For every butterfly bfy
6) Compute fragrance using equ. (1)
7) end of for loop
8) Select the best bfy
9) For every bfy in the population do
a. Generate a random probability s between 0 to1.
10) Check if s<p, then
11) Take step towards best bfy for global search by using equ. (2)
12) else
13) Move randomly, and perform local search equ. (3)
14) end if
15) Evaluate the newly generated solution & update the better solution.
16) end for
17) Update the value of a parameter.
18) end while
19) The best solution found.

System Architecture:

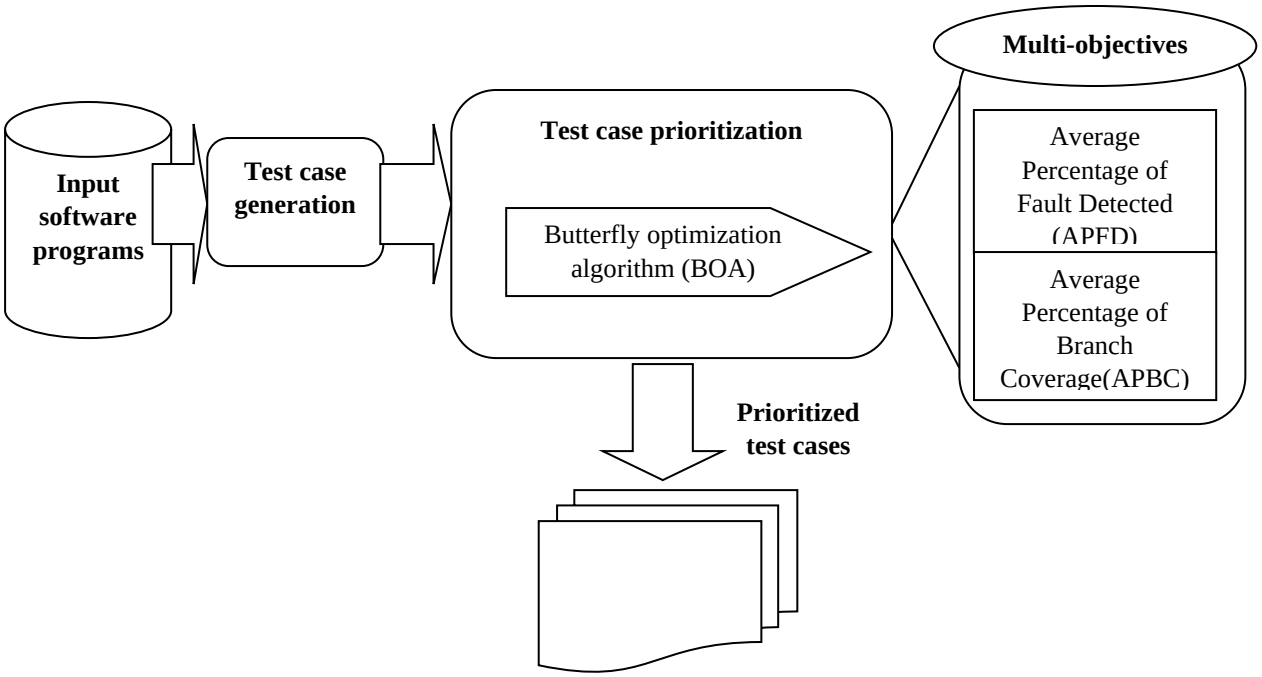

Figure 1: System Architecture

Figure 1 presents the architectural design of the test case prioritization model using the proposed Butterfly optimization test case prioritization algorithm. We had designed and built a priority test case technique through the proposed methodologies by using a Butterfly optimization algorithm. The software programmes are initially supplied for the test case generation phase, in which the test case from the given programming is produced. The test case to be chosen is decided based on a sequentially weighted coverage criterion or fitness. Therefore, to determine optimal test cases based on constraints an effective search criterion will be formulated. The search criterion is formulated as an optimization problem, solved through an algorithm. The test case prioritisation is therefore carried out by the tester using the proposed Butterfly optimization algorithm. Besides, two objectives such as an average percentage of detected failures (APFD) and an average branch coverage (APBC) will be considered in the proposed model for prioritising the test case. Here, the performance of the proposal is assessed using performance measurements, such as APFD and APBC.

Experiment Analysis and Results:

The test was conducted in a stable laboratory setting with a computer running on a 4 GB RAM and Core i5 Intel CPU. The different test case suites are used for performing the TCP optimization scenario. Butterfly algorithms and other evaluation algorithms programming are developed using Python 3.7 version and necessary supported libraries. The test case fault matrix was created in Comma Separated Value(CSV) file and then converted into an equivalent txt file with CSV. We measured the prioritisation of experiment performance to run the permutation of test cases and analysed the results with APFD and execution time parameters. Table 1 summarizes the details of all three projects on which we have executed the experiments.

Table 2: Project Description for comparative Analysis

S.No.	Project Name	No. of test cases	No. of faults
Project 1	Real Estate Management System	237	52
Project 2	Online Examination System	103	23
Project 3	Project Management System	138	27

APFD metric is considered a fitness function and is applied to compute the score of fault detection rate [28]. APFD measures the total number of faults detected by the prioritized test case order. No. of unique TCs is also considered as a parameter for evaluation of proposed work with existing. We have also evaluated our approach for execution time metrics. Table 3,4,5 show the comparison of the APFD results for Project 1, Project 2, Project 3 respectively.

Table 3: APFD Results for Real Estate Management System

Algorithm Name	APFD Values(in %)
GA	80.03
HC External	66.65
HC Internal	67.39
Random Search	68.61
Whale Optimization	69.31
FireFly	90.14
Butterfly	94.55

Table 4: APFD Results for Online Examination System

Algorithm Name	APFD Values(in %)
GA	78.25
HC External	64.93
HC Internal	65.23
Random Search	67.98
Whale Optimization	68.12
FireFly	88.43
Butterfly	92.47

Table 5: APFD Results for Project Management System

Algorithm Name	APFD Values(in %)
GA	79.63
HC External	65.96
HC Internal	66.15
Random Search	68.05
Whale Optimization	70.25
FireFly	87.12
Butterfly	91.47

From table 3,4,5 it is observed that the proposed butterfly algorithm had outperformed all other previous optimization algorithms for test case prioritization and was given a good APFD value.

Figure 2 represent the graphical representation of APFD result comparison for all 3 three projects mentioned as per table 2. It clearly shows that the Butterfly algorithm had outperformed compared to all other algorithms in terms of APFD calculations.

Tables 6 represent the average time taken to complete the search for solution paths for all three benchmark programming for any priority algorithm in terms of algorithm performance. Butterfly Algorithm offers the lesser

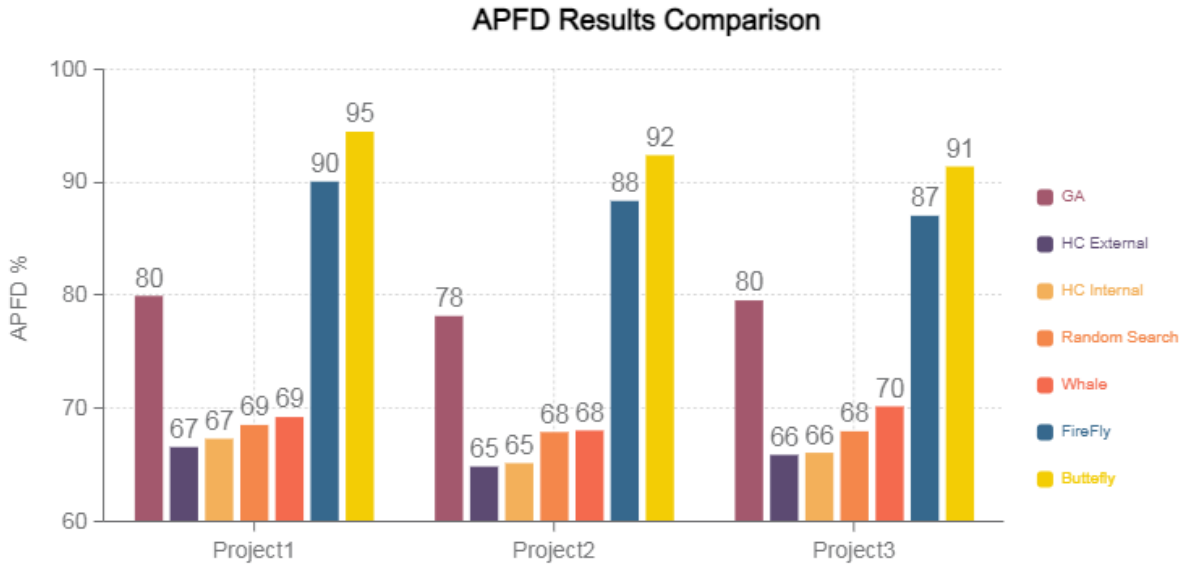

Figure 2: APFD Result Comparison

average time taken to prioritise test cases in all three Benchmark programmes compared to Firefly, Whale and GA optimization algorithm, whereas Hill Climbing and Random Search show reduced time complexities in comparison with another algorithm. But being the main focus of algorithm comparison of Butterfly was with GA, Whale and Firefly, in that scenario, Butterfly had executed the TCP problem faster. The butterfly technique, which is focused exclusively on the next brightest butterfly agent in the research, should attribute this outcome to a reduced time to hunt. The number of iterations has also been set based on the scale of the benchmarks.

Table 6: Execution Time Analysis

Name of Algorithm	**Project 1**	**Project 2**	**Project 3**
GA	9.044	8.124	8.845
HC External	0.159	0.162	0.179
HC Internal	0.126	0.132	0.139
Random Search	0.559	0.570	0.615
Whale Optimization	3.809	3.256	3.321
FireFly	7.574	7.129	7.478
Butterfly	3.73	3.08	3.68

In figure 3 we had analysed the execution time for each algorithm. The main comparison is between Firefly and

Butterfly algorithms. Figure 3 and respective table 6, shows that the Butterfly algorithm takes lesser time in execution and gives an output of APFD in comparison with the Firefly algorithm. The GA algorithm also has a higher time of execution whereas HC and RS although shows lesser time but there is no iteration we had performed as compared to GA, Butterfly and Firefly.

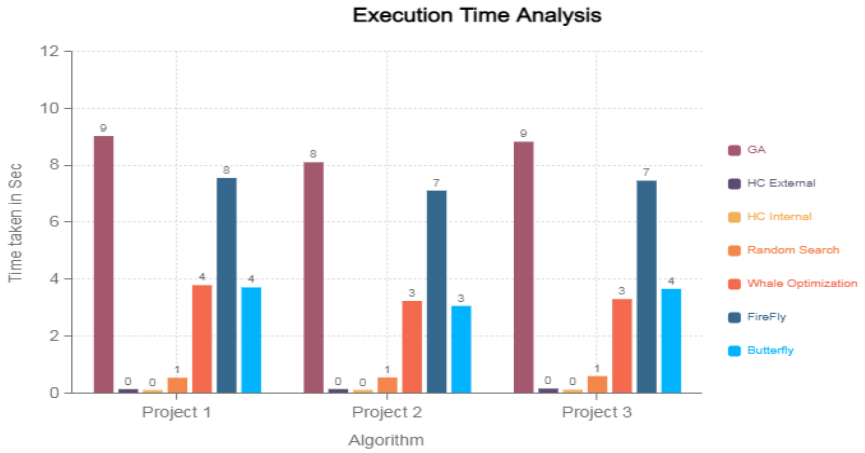

Figure 3: Execution Time Analysis

Another parameter on which we had tried to compare these algorithms is on basis of the Average percentage of Branch Coverage(APBC). The APBC is considered one of the best and proven white box testing mechanisms. The formula for calculating APBC is as follows.

$$\text{Branch Coverage } \frac{Number\ of\ decision/branch\ outcomes\ exercised}{Total\ number\ of\ decision\ outcomes\ in\ the\ source\ code)} *100 \qquad (4)$$

For performing the calculation of APBC, there are so many tools available readily. We had used python library naming covergy.py(https://coverage.readthedocs.io/en/coverage-5.5/index.html). Coverage.py is a Python utility for determining the code coverage of Python applications and scripts. It runs your programme in the background, recording which sections of code have been executed, and then examines the source code to find sections of code that might have been executed but were not. Branch coverage measurement is also supported by the coverage.py package. Whenever a line in your programme has the potential to go to more than one next line, coverage.py keeps track of which of those destinations is visited, and flags lines that haven't visited all of their potential future lines.

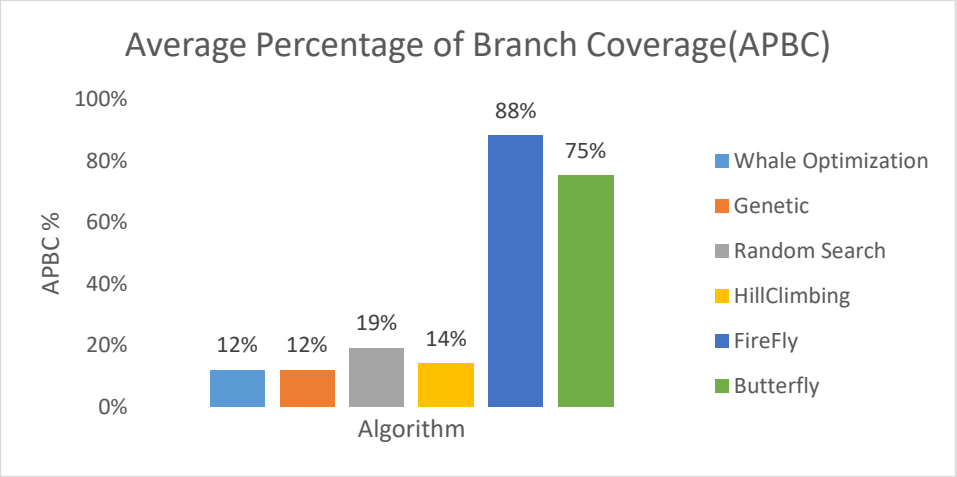

Figure 4: APBC Comparison

The APBC of Firefly is 85% in comparison with 75% of Butterfly. Here firefly has shown good results.

CONCLUSION AND FUTURE SCOPE

In prioritising test cases, this paper used a Butterfly approach to obtain higher APFD performance. In conjunction with other priority algorithms, 3 benchmarking programmes have been used to verify Butterfly Algorithm's reliability and performance in priority test suites. Butterfly achieved higher APFD values compared to other prioritisation algorithms in the experimental results. The experiment shows also that Butterfly Algorithm was above the time of execution of GA and Firefly algorithms. In summary, the findings of the APFD reveal that Butterfly Algorithm is a major competitor in the field of TCP. The findings from the APFD show that butterfly algorithms can be successful in identifying problems of defect proneness, which are important in safety-critical systems. In future work on this particular swarm intelligence algorithm focussing on coverage quality, it will be useful to study the potential improvement.

REFERENCES:

[1] S. Reid, "The Art of Software Testing, Second edition. Glenford J. Myers. Revised and updated by Tom Badgett and Todd M. Thomas, with Corey Sandler. John Wiley and Sons, New Jersey, U.S.A., 2004. ISBN: 0-471-46912-2, pp 234," *Softw. Testing, Verif. Reliab.*, vol. 15, no. 2, pp. 136–137, Jun. 2005, doi: 10.1002/stvr.322.

[2] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test Case Prioritization: An Empirical Study," 1999.

[3] M. Khatibsyarbini, M. Adham Isa, and D. Norhayati Abang Jawawi, "A HYBRID WEIGHT-BASED AND STRING DISTANCES USING PARTICLE SWARM OPTIMIZATION FOR PRIORITIZING TEST CASES," *J. Theor. Appl. Inf. Technol.*, vol. 30, no. 12, 2017, Accessed: Feb. 17, 2021. [Online]. Available: www.jatit.org.

[4] M. Khatibsyarbini, M. A. Isa, D. N. A. Jawawi, and R. Tumeng, "Test case prioritization approaches in regression testing: A systematic literature review," *Information and Software Technology*, vol. 93. Elsevier B.V., pp. 74–93, Jan. 01, 2018, doi: 10.1016/j.infsof.2017.08.014.

[5] B. Jiang and W. K. Chan, "Input-based adaptive randomized test case prioritization: A local beam search approach," *J. Syst. Softw.*, vol. 105, pp. 91–106, Jul. 2015, doi: 10.1016/j.jss.2015.03.066.

[6] W. E. Wong, J. R. Horgan, S. London, and H. Agrawal, "Study of effective regression testing in practice," in *Proceedings of the International Symposium on Software Reliability Engineering, ISSRE*, 1997, pp. 264–274, doi: 10.1109/issre.1997.630875.

[7] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Prioritizing Test Cases for Regression Testing."

[8] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Trans. Softw. Eng.*, vol. 28, no. 2, pp. 159–182, Feb. 2002, doi: 10.1109/32.988497.

[9] S. Singhal, B. Suri, A. Kaur, and Y. Singh, "Systematic literature review on regression test prioritization techniques," *Informatica*, vol. 36, pp. 379–408, Dec. 2012.

[10] Y. Ledru, A. Petrenko, S. Boroday, and N. Mandran, "Prioritizing test cases with string distances," in *Automated Software Engineering*, Mar. 2012, vol. 19, no. 1, pp. 65–95, doi: 10.1007/s10515-011-0093-0.

[11] L. Mei, W. K. Chan, T. H. Tse, B. Jiang, and K. Zhai, "Preemptive Regression Testing of Workflow-Based Web Services," *IEEE Trans. Serv. Comput.*, 2014, doi: 10.1109/TSC.2014.2322621.

[12] D. Gao, X. Guo, and L. Zhao, "Test case prioritization for regression testing based on ant colony optimization," in *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, Nov. 2015, vol. 2015-November, pp. 275–279, doi: 10.1109/ICSESS.2015.7339054.

[13] R. M. Parizi, A. Kasem, and A. Abdullah, "Towards Gamification in Software Traceability: Between Test and Code Artifacts," doi: 10.5220/0005555503930400.

[14] D. Panwar, P. Tomar, and V. Singh, "Hybridization of Cuckoo-ACO algorithm for test case prioritization," *J. Stat. Manag. Syst.*, vol. 21, no. 4, pp. 539–546, Jul. 2018, doi: 10.1080/09720510.2018.1466962.

[15] M. Khatibsyarbini, M. A. Isa, D. N. A. Jawawi, H. N. A. Hamed, and M. D. Mohamed Suffian, "Test Case

Prioritization Using Firefly Algorithm for Software Testing," *IEEE Access*, vol. 7, pp. 132360–132373, 2019, doi: 10.1109/ACCESS.2019.2940620.

[16] S. K. Harikarthik, V. Palanisamy, and P. Ramanathan, "Optimal test suite selection in regression testing with testcase prioritization using modified Ann and Whale optimization algorithm," *Cluster Comput.*, vol. 22, no. 5, pp. 11425–11434, Sep. 2019, doi: 10.1007/s10586-017-1401-7.

[17] R. U. Maheswari and D. Jeya Mala, "Combined Genetic and simulated annealing approach for test case prioritization," *Indian J. Sci. Technol.*, vol. 8, no. 35, Dec. 2015, doi: 10.17485/ijst/2015/v8i35/81102.

[18] V. Reddy, "Regression Test Suite Prioritization Using Hill Climbing Algorithm," *IOSR*, vol. Volume 18, p. PP 130-141, Jun. 2016.

[19] E. Ashraf, K. Mahmood, T. Ahmed, and S. Ahmed, "Value based PSO Test Case Prioritization Algorithm," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 1, 2017, doi: 10.14569/ijacsa.2017.080149.

[20] R. Raguso, "Wake Up and Smell the Roses: The Ecology and Evolution of Floral Scent," *Annu. Rev. Ecol. Evol. Syst.*, vol. 39, pp. 549–569, Oct. 2008, doi: 10.1146/annurev.ecolsys.38.091206.095601.

[21] T. Wyatt, "Pheromones and animal behaviour: communication by smell and taste (book)," vol. 41, Jan. 2003, doi: 10.1017/CBO9780511615061.002.

[22] T. K. Sharma and C. Wook, "Synergetic Butterfly Optimization Algorithm*."