DOI: http://dx.doi.org/10.26483/ijarcs.v9i1.5327

Volume 9, No. 1, January-February 2018

ISSN No. 0976-5697

International Journal of Advanced Research in Computer Science

REVIEW ARTICLE

Available Online at www.ijarcs.info

PRIORITIZING TECHNIQUE BASED ON GENETIC ALGORITHM- AN EXPERIMENTAL REVIEW

Garima
Computer Science Engineering
CT Institute of Engineering, Management and Technology
University: Punjab Technical University
Jalandhar,India

Sukhvir kaur
Computer Science Engineering
CT Institute of Engineering, Management and Technology
University: Punjab Technical University
Jalandhar,India

Abstract- Software testing is the system of validation and verification of the application product. Powerful program testing contributes to the supply of dependable, nice software product, leading to low application preservation rate and satisfied users. For this reason, it is an primary exercise of program progress approach. The importance of testing can be gauged from the indisputable fact that approximately 35% of the whole time required for setting up the program and over 50% of the whole development price is used for trying out. Mainly, exhaustive trying out requires colossal time and effort, making it pricey and infeasible. From the point of view of pleasant of checking out and discount of testing fee, automation of trying out process is the necessity of the hour. This paper implements improvised prioritizing technique based on Genetic algorithm.

Keywords – Software engineering, Software testing, Test techniques, Genetic Algorithm, Test case prioritization.

I. INTRODUCTION

Software engineering is the systematic strategy to development, operation, upkeep, and retirement of any software. Progress of any program follows a series of steps often called the program development existence cycle‘(SDLC). Application checking out is a foremost a part of SDLC. Program testing is the major nice control measure employed during software development. Its normal function is to observe error within the application [1]. Trying out consumes the best possible amount of time in the application development lifestyles cycle.

Whilst the essential factor in trying out is to test the performance of the application, it also ensures that the program meets the efficiency requirement, customer expectations, reliability, flexibility, correctness and many others. Application testing is most often utilized in organization with the terms verification ‘and validation ‘of whole SDLC. Software testing is an essential but incredibly laborious and high priced procedure [1]. Measurement of the effort in application checking out by itself is a complex quandary within the domain of software engineering.

Program testing is an empirical investigation carried out to furnish stakeholders with information concerning the quality of the product or service under experiment, with appreciate to the context where it's supposed to function. Software testing additionally supplies an purpose, impartial view of the software to permit the trade to appreciate and comprehend the hazards at implementation of the application. It will also be recounted because the system of validating and verifying that a program application/application/product meets the business and technical necessities.

Test case prioritization is a method to organize and plan test cases. The strategy is produced keeping in mind the end goal to run test instances of higher need to limit time, cost and exertion amid programming testing stage [12]. The motivation behind this prioritization is to improve the probability that if the experiments are utilized for relapse testing in the given request, they will more firmly meet some goal than they would in the event that they were executed in some unique request. Some organizations prefer to run “Smoke” or “Sanity” test every time they get a new build or version of the developing software [15]. In this case, test cases will be prioritize based on all the major modules of the software and sanity will be run on them to check the basic functionality for example, in a mobile testing, sanity test suite will have test cases like “restarting the device”, “turning off”,

“signing in”, “updating software” etc. Whether your organization runs regression or sanity or both, test Case

Prioritization techniques are applicable for all the cases [9] [14]. Organizing experiments should be possible based on necessities, expenses of bug settling, history of the parent gadget etc.

Since prioritizing the test cases helps in detecting the faults at an early stage, so substantial time can be saved which can further be utilized in early start-up of debugging activities.

In order to find out the best test case execution sequence in test case prioritization, all the possible permutations of the original test suite are considered as candidates [13]. Genetic algorithms are the search heuristics which are widely used in test case prioritization problems. The primary advantage of utilizing hereditary calculation is that it has a place with a more extensive class of transformative calculations which produce answers for enhancement issues by utilizing the systems propelled by characteristic advancement. These incorporate legacy, hybrid, transformation and choice.

A subset of Evolutionary Algorithms, Genetic Algorithm is used for generating optimal or near optimal solutions to complex problems by relying on techniques inspired by natural selection. Genetic algorithms are usually employed for generating optimal or near optimal solutions to complex problems by means of genetic operator’s viz., crossover, mutation and selection [5].

A genetic algorithm works by evolving a population of possible solutions to a complicated problem towards an optimal solution. Every possible solution has its own group of characteristics (referred to as chromosomes or genotype) that can be changed and mutated. Traditional representation of solutions involves binary strings of 0s and 1s. However, other representations are also viable [11].

To start with, a random pool of all possible solutions (represented as chromosomes) to a given problem is generated. This population is then modified time and again in order to reach an ideal solution. At each step, one or more chromosomes from the current population are picked up as per some fitness function of the problem under consideration. These best fit individuals are then used to create new off springs, which are further added to the next generation. The new generation of candidate solutions is then utilised in the next iteration of the algorithm. Usually, the algorithm ceases when largest number of generations has been reached, or an acceptable fitness value has been achieved.

II. RELATED WORK

A modified condition/decision coverage based approach is proposed by Jones. They presented 2 new calculations for test suite diminishment and prioritization that join parts of MC/DC viably like complexities of MC/DC and also the success of test suite decline technique is evaluated on a Space series which is implemented in C language. The proposed approach has been proven to be more effective in terms of fault detection. The paper additionally exhibits the aftereffects of experimental examinations that assess these calculations. The outcomes accomplished up to this point are empowering in that they demonstrate the potential for generous test-suite estimate lessening concerning MC/DC. Such methods can altogether lessen the cost of relapse testing for those clients of this effective testing measure. [1] Korel proposed the to start with demonstrate based experiment prioritization approach, which plans the request of experiments on the basis of collected execution data of the changed model together with the past framework display and the adjusted framework show. Execution of the demonstrate is reasonable when contrasted with execution of the framework, hence the overhead connected with test prioritization is comparatively little. This paper also presented an expository structure for assessment of test prioritization techniques. This system may decrease the cost of assessment when contrasted with the current assessment structure that depends on experimentation. The consequences of the exploratory investigation recommend that framework models may enhance the viability of test prioritization as for early fault discovery. [2] Walcott presented a relapse test prioritization system that uses a hereditary calculation to reorder test suites in light of testing time imperatives. Examination comes about show that this prioritization approach regularly yields higher normal level of flaws distinguished (APFD) values, for two contextual analysis applications, when essential square level scope is utilized rather than technique level scope. The investigations likewise uncover principal exchange offs in the execution of time-mindful prioritization. This paper demonstrates that the prioritization system is suitable for some, relapse testing situations and clarifies how the standard approach can be stretched out to work in extra time obliged testing conditions. [3] A technique for cost-conscious experiment prioritization in light of the utilization of verifiable records is proposed by Yu-Chi. They assemble the verifiable records from the most recent relapse testing and afterward propose a hereditary calculation to decide the best request. Some controlled examinations are performed to assess the viability of the proposed strategy. Assessment comes about demonstrate that the proposed technique has enhanced the blame recognition adequacy. It can likewise been discovered that organizing experiments in light of their verifiable data can give high test viability amid testing. [4]

P.R. Srivastava developed a variable length Genetic Algorithm for detecting the most critical path clusters for optimizing software testing performance. To implement this changeable measurement lengthwise Genetic Algorithms is developed that enhance and select the product way groups which are weighted as per the criticality of the way. Thorough programming testing is once in a while conceivable on the grounds that it ends up noticeably recalcitrant for even medium estimated programming. The proposed method outperformed the local and exhaustive search techniques. By investigating the most critical paths first, it led to an extra effective manner to technique testing which in turn, helped to perform effort and cost estimations in a better way during testing phase. [5] Conrad introduced a wide assortment of change, hybrid, choice and change administrator that were utilized to reorder the test suite. An exploratory examination was executed on 8 contextual analysis applications, utilizing APFD as scope viability metric and their JUnit test cases at framework level. The outcomes are investigated with the assistance of bean plots. On correlation of the proposed procedure with irregular inquiry and slope climbing methods, GA yields better outcomes. Likewise, GA is found to have comparative execution times as that of arbitrary inquiry and slope climbing. All things considered, GA demonstrates a more prominent inconstancy and is likewise a forthcoming territory of research in the field. [6] A technique of cost-cognizant test case prioritization which was on the basis of usage of historic records is proposed by Y.Huang. The chronological data was gathered from the most recent regression testing after which a genetic algorithm was proposed to decide the most effectual order. Results proved that the proposed technique led to an upgrading in the fault detection effectiveness. [7] Yu-Chi proposed a cost-aware prioritization method that requested experiments as indicated by their history data by utilizing hereditary calculation. The system organized experiments based on their test expenses and blame severities, without investigating the source code. It additionally enhanced the prioritization execution by maintaining a strategic distance from specific situations where the experiments with comparable capacity in the past relapse testing were given a similar rank. The productivity of the same was assessed by utilizing a UNIX utility program and the outcomes affirmed the value of the proposed procedure. [8] An algorithm for system level TCP from software requirement specification is proposed by R.Kavitha. This was done with an aim to enhance client fulfillment with quality programming and furthermore to enhance the rate of serious blame identification. The proposed calculation organized the experiments based on three factors to be specific, client need, changes in necessity and execution many-sided quality. The proposed procedure was then approved with two distinct arrangements of mechanical tasks and the outcomes showed

that it improved the rate of fault detection. [9] Another Genetic calculation for organizing the relapse test suite, which organized the experiments based on add up to code scope by A. Kaur. Here, various prioritization approaches have been broke down, in particular: add up to blame scope with in time obliged condition and measure of code scope on various illustrations and their limited arrangement got, separately. The proposed approach helped in automating the test case prioritization process. The results denoting its efficiency were evaluated by means of Average percentage of Code Coverage (APCC) metric. [10]

III. EXISTING METHODS

A subset of Evolutionary Algorithms, Genetic Algorithm is used for generating optimal or near optimal solutions to complex problems by relying on techniques inspired by natural selection. Genetic algorithms are usually employed for generating optimal or near optimal solutions to complex problems by means of genetic operator's viz., crossover, mutation and selection.

A genetic algorithm works by evolving a population of possible solutions to a complicated problem towards an optimal solution. Every possible solution has its own group of characteristics (referred to as chromosomes or genotype) that can be changed and mutated. Traditional representation of solutions involves binary strings of 0s and 1s. However, other representations are also viable.

To start with, a random pool of all possible solutions (represented as chromosomes) to a given problem is generated. This population is then modified time and again in order to reach an ideal solution. At each step, one or more chromosomes from the current population are picked up as per some fitness function of the problem under consideration. These best fit individuals are then used to create new off springs, which are further added to the next generation. The new generation of candidate solutions is then utilised in the next iteration of the algorithm. Usually, the algorithm ceases when largest number of generations has been reached, or an acceptable fitness value has been achieved.

- In GAs, we have a pool or a population of possible solutions to the given problem.
- These solutions then undergo recombination and mutation (like in natural genetics), producing new children, and the process is repeated over various generations. Each individual (or candidate solution) is assigned a fitness value (based on its objective function value) and the fitter individuals are given a higher chance to mate and yield more "fitter" individuals. This is in line with the Darwinian Theory of "Survival of the Fittest".
- In this way we keep "evolving" better individuals or solutions over generations, till we reach a stopping criterion.

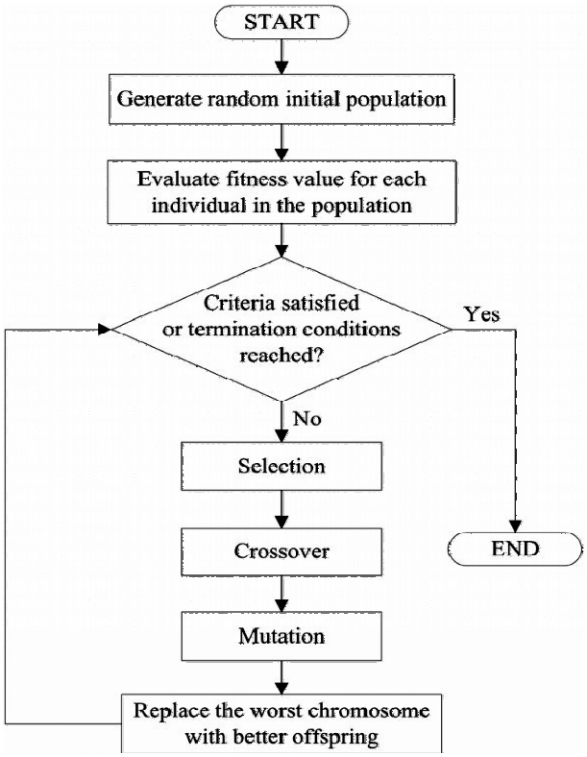

Figure 3.1 Genetic Algorithm

IV. EXPERIMENTAL RESULTS

To implement genetic algorithm, we take different number of generations and measure execution time, APSC (average percentage suite coverage) measure and APFD (average percentage fault detection) measure based on the number of generations.

Table 4.1 Number of iterations for APSC

Sr. No.	No. of Iterations	Genetic Algorithm
1	2nd generation	95.72
2	3rd generation	96.53
3	4th generation	96.28
4	5th generation	96.35

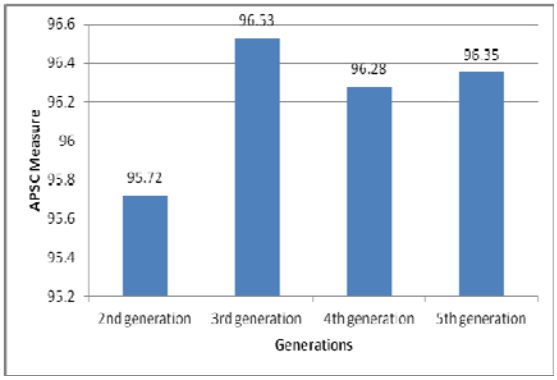

Figure 4.1 Graph showing APSC for number of iterations

Table 4.1 Number of iterations for APFD

Sr. No.	No. of Iterations	Existing Technique
1	2nd generation	97.1
2	3rd generation	96.7
3	4th generation	94.8
4	5th generation	98.9

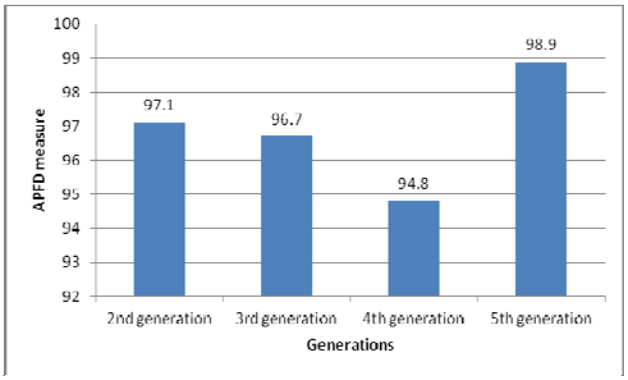

Figure 4.2 Graph showing APFD for number of iterations

V. CONCLUSION

The major purpose of this paper is to gain knowledge of about software testing, and scan case prioritization. On this paper, Genetic Algorithm and repair-and-Reschedule adaptive method is combined to perform scan case prioritization in program trying out. Scan Case Prioritization pursuits to organize the test cases so as to maximize the announcement coverage or fault detection cost. In contrary to this, a repair-and-reschedule adaptive procedure preserves time by way of carrying out these two duties concurrently. Nevertheless, it most effective schedules the going for walks order of some chosen test cases that have attained some amount of announcement insurance policy or fault detection previously. Consequently, it does not consider the entire experiment cases for prioritization, which implies that full declaration coverage or fault detection has now not yet been accomplished. In future, we can combine the genetic algorithm with fix-and-reschedule algorithm.

REFERENCES

[1] J. A. Jones and M. J. Harrold, "Test-suite reduction and prioritization for modified condition/decision coverage," IEEE Transactions on Software Engineering, vol. 29, no. 3, pp. 195– 209, 2003.

[2] B. Korel, L. H. Tahat, and M. Harman, "Test prioritization using system models," in the 21st IEEE International Conference on Software Maintenance, 2005, pp. 559–568.

[3] K.R.Walcott, M.L.Soffa, G.M.Kapfhammer and R.S. Roos. "Time aware test suite Prioritization", International Symposium on software Testing and Analysis, pp. 1-12, July 2006.

[4] H. Park, H. Ryu, and J. Baik, "Historical value-based approach for cost-cognizant test case prioritization to improve the effectiveness of regression testing," The Journal of Systems and Software, pp. 39–46, 2008.

[5] P R Srivastava, T. Kim. Application of Genetic Algorithm in Software Testing. In: Proceedings of International Journal of Software Engineering and its applications, 87-96, October 2009.

[6] A. P.Conrad,R. S.Roos, "Empirically Studying the role of selection operators during search based test suite prioritization", In the Proceedings of the ACM SIGEVO Genetic and Evolutionary Computation Conference, Portland, Oregon, 2010.

[7] YC Huang, CY Huang, JR Chang. Design and Analysis of Cost-Cognizant Test Case Prioritization Using Genetic Algorithm with Test History. In: Proceedings of 34th IEEE Annual Computer Software and Applications Conference, 413-418, July 2010.

[8] Y. Huang, C. Huang, J. Chang, T. Chen: "Design and Analysis of Cost-Cognizant Test Case Prioritization Using Genetic Algorithm with Test History." In: Proceedings of 34th IEEE Annual Computer Software and Applications Conference, (2010)

[9] R. Kavitha, V.R. Kavitha, Dr. N. S. Kumar: "Requirement Based Test Case Prioritization." In: Proceedings of IEEE International Conference on Communication Control and Computing Technologies (ICCCCT), (2010)

[10] A. Kaur, S. Goyal. A genetic algorithm for regression test case prioritization using code coverage. In: Proceedings of International Journal on Computer Science and Engineering, 1839-1847, May 2011.

[11] S. Sabharwal, R. Sibal, C. Sharma. A Genetic Algorithm based Approach for Prioritization of Test Case Scenarios in static testing. In: Proceedings of International Conference on Computers and Communication Technology (ICCCT), 304-309, September 2011.

[12] YC Huang, KL Peng, CY Huang. A history-based cost cognizant test case prioritization technique in regression testing. The Journal of Systems and Software, 85(3): 626-637, March 2012.

[13] G. Fraser, A Arcuri. Whole Test Suite Generation. IEEE Transactions on Software Engineering. 39(2), 276-291, February 2013.

[14] D. Hao, X. Zhao, L. Zhang. Adaptive Test-Case Prioritization Guided by Output Inspection. In: Proceedings of 37th IEEE Annual Computer Software and Applications Conference (COMPSAC), 169-179, July 2013.

[15] T. B. Noor, H. Hemmati: "A similarity-based approach for test case prioritization using historical failure data." In: Proceedings of 26th IEEE International Symposium on Software Reliability Engineering (ISSRE), (2015)