ISSN:0975-9646

A Genetic Algorithm Based Average Percentage of Statement Coverage Technique for Test Case Ordering

Atul kumar Pal[1], Makul Mahajan[2]
[1] *Student, Lovely Professional University,*
[2] *Assistant Professor, Lovely Professional University*

Abstract: This paper address the research in the field of test case ordering in regression testing. The idea is to improve APSC by applying our proposed approach adaptive genetic algorithm hybrid approach for test case ordering in regression testing. In this research basically we focused on test-case ordering and statement coverage by Applying adaptive genetic algorithm hybrid approach and measure APSC (Average Percentage Statement Coverage) and GA (Genetic Algorithm). In this research we take hundred test-case of apache server and evaluate hundred test-cases. We used java eclipse environment for coding and run the test cases. First we apply APSC (Average Percentage of statement coverage) technique for ordering test-cases as well measure the APSC. We got good results but this technique not sufficient to cover maximum statement. So, we hybrid the adaptive and Genetic Algorithm approach to measure the APSC and run all test-cases until all statement not covered.. Our approach gives us better results than single APSC adaptive technique.

Keywords: ***Regression Testing, Test case prioritization/ordering, Genetic Algorithm , APSC(Average Percentage of statement coverage) ,Adaptive Approach ,APFD(Average Percentage of Fault Detection),Fitness, Parent Generation , Crossover , Mutation.***

INTRODUCTION

Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software .It is also defined as a systematic approach to the analysis, design, assessment, implementation, testing, maintenance and reengineering of software .Software testing is an important activity in software development as well in software maintenance and quality assurance. It recognizes defects and problems, and evaluates and improves product quality. Software testing has been a serious research topic since the late 1960s. Software testing may represent more than 40%-60% of a software development budget. Moreover, approximately 50% of the elapsed time is expended in testing software being developed. Software maintenance refers to the modifications of software after delivery. Other terms suggested for maintenance are software support, software renovation, continuation engineering and software evolution [9]. To discover blames and issues in the item outline as right on time as would be prudent, trying is done in numerous stages. Software Testing Play an important role in assuring the software quality of the system. However many research papers proved and state that more than half of cost in software is used in testing and maintenance of the software. So many researchers already had done a lot research in to reducing the cost of software testing. But as well we have need to take care of their will be no effect on the quality while we apply many approach in reducing the cost of testing ex: we can detect fault properly, we can cover overall statements of the code, we can provide ordering to each test case in which sequence we run test case that we cover all statements of the code.

In this research we focused on Adaptive approach and extend this approach by applying Genetic Algorithms through this approach we found that we got better result of APSC then existing approaches. The adaptive test-case ordering approach computes the fault-detection capability of each test case based on the faulty potential (which measures to what extent a statement is likely to contain faults) of its executed statements. During regression testing, as soon as a selected test case finishes running, the adaptive approach modifies the faulty potential of all the statements executed by this test case based on its output, and then modifies the fault detection capability of all unselected test cases. The adaptive approach selects a test case with the largest fault-detection capability and programmers run the selected test case. The preceding process repeats until all the test cases are selected and run. Generally speaking, the adaptive approach schedules test cases and executes test cases simultaneously. This is also the main difference between the adaptive approach and existing test-case prioritization approaches.

In this research we focus to improve the efficiency in average percentage of statement coverage technique. We improved the APSC of adaptive approach by applying our proposed approach Adaptive genetic algorithm hybrid approach.

In this paper we discuss our research work in five section. In Section I we discus on related work. In Section II we discus on our Problem Formulation . In Section III we discus on our Research Methodology . In Section IV we discuss on our Results . In Section V we discuss on Conclusion and Future Scope.

1. RELATED WORK

Dan Hao *et al*.(2013) says that prioritization of test-case is to arrange the execution order of test cases like that we can concentrate on some destinations like ahead of schedule flaw identification in the code before execute the experiments. They connected the versatile approach in existing experiment prioritization approach. The proposed methodology separate the procedure of experiment prioritization and the execution transform by giving the execution request to every single test case before run the experiments. As the implementation data of adjusted code is not available for existing experiment prioritization these methodologies rely on upon the past Program execution data

before changes in the Program. To conquer this issue, they show a multipurpose investigate prioritization approach, which chooses the implementation request of experiments at the same time amid the execution of experiments. The versatile methodology chooses experiments in light of their flaw identification ability, which is computed in view of the produce of chose experiments. When an experiment is chosen and runs, the deficiency recognition ability of every unselected experiment is changed by yield of the most recent chose experiment. To assess their proposed methodology they perform this methodology on eight C language Program and four java language Program. Their experimental results prove the Adaptive approach is significantly better than the existing test case prioritization. In figure 1: comparison of both approach is shown [3].

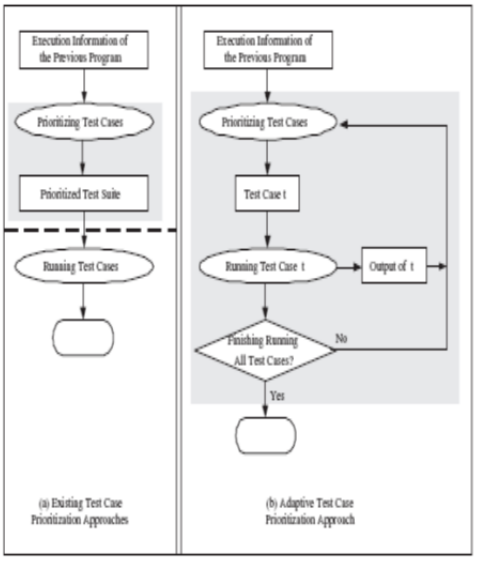

Figure 1: Comparison between Test case prioritization approach and adaptive approach.[3]

Yu-Chi Huang *et al.*(2011) give brief history detail on test-case prioritization technique for regression testing and applied Genetic Algorithms process to cover statement of the code . They perceived that during testing, the experiment is a couple of data and expected yield, and various experiments will be executed either successively or haphazardly. The procedures of experiment prioritization generally timetable experiments for relapse testing in a request that endeavors to expand the proficiency. In any case, the expense of experiments and the strictness of shortcomings are generally shifted. In their paper, they propose a method of expense aware experiment prioritization taking into account the utilization of past records. They accumulate the past records from the most recent relapse testing and afterward propose a hereditary calculation to choose the best request. Some very much requested analyses are performed to assess the viability of our proposed system. Assessment results show that their proposed methodology has enhanced the deficiency discovery adequacy. It can likewise been discovered that organizing experiments in light of their authentic data can give high test adequacy amid testing [27].

2. PROBLEM FORMULATION

The existing test-case ordering approaches Dan Hao et al.[23] present an adaptive test-case prioritization approach, which determines the implementation order of test cases concurrently during the execution of test cases. In particular, the adaptive approach selects test cases based on their fault detection capability, which is calculated based on the output of selected test cases. As soon as a test case is selected and runs, the fault-detection capability of each unselected test case is modified according to the output of the latest selected test case. The adaptive approach is better than the additional approach on some subjects (e.g, replace and schedule).

When we apply this approach by taking hundred apache server test cases in java. we found that only 31 test cases cover near about 98 percent statements coverage but what about left test cases how we provide them order that we can cover maximum statements, so to improve this problem we applied Genetic Algorithm approach with adaptive approach on left test cases only and we found that this Adaptive Genetic Algorithm is better than simple adaptive approach we cover 99.6 percentage approx. statements cover by our proposed approach .

Figure2 shows that how we reach towards this problem this flow diagram represent the our work process from beginning of the research .

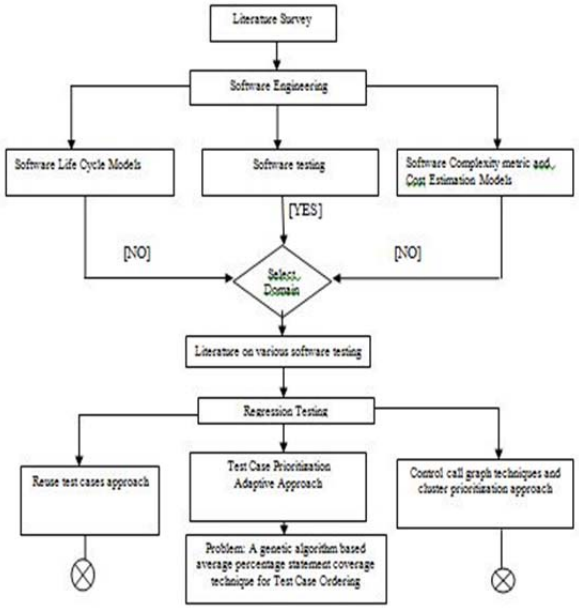

Figure 2: Flow chart to reach the problem in research.

3. RESEARCH METHODOLOGY

Our Research Methodology is basically the extension of adaptive approach in which we merge the genetic algorithm and we form new algorithm and we give that algorithm name is Adaptive Genetic Hybrid algorithm. Figure 3 our Proposed algorithm from step 1 to step 18 the adaptive approach work done from step 19 we apply genetic algorithm. Step 21 we take left test cases as Input those are left after adaptive approach and on left test cases we apply genetic algorithm. From step 31 we apply the process to measure the APSC(Average Percentage of Statement Coverage).

Algorithm of Adaptive Genetic Hybrid proposed Approach

Input: Test Suite T

Output: $T_{greatest}$(A test case which has largest fitness value in population of final generation).

APSC (Measure Adaptive Percentage of Statement Coverage)

Declaration:

Ts: represent the latest selected test case .
N : number of test cases
M : statements
P: population size .
G: number of generation.
Cp: Crossover Point.
Mp : Mutation Point.
Ltc : Left Test cases after Adaptive Approach ordering .

Adaptive Process :

1. **Begin**
2. for each test case t in T.
3. calculate initial priority(t).
4. **End** for .
5. Select the test case (ts) with the largest priority in T.
6. Add ts to T'
7. T ← T-{ts}.
8. Run ts.
9. **While** T is not empty **do**.
10. For each test case t in T.
11. Change priority(t) based on the output of ts.
12. End for
13. Select the test case(ts) with largest priority which cover statement.
14. Add ts to T'.
15. T ← T-{ts}.
16. Run ts.
17. End while
18. Return Ltc : left test case from T those not cover statement.
19. **Genetic Algorithm Process :**
20. **Begin :**
21. **Input:** Ltc
22. P_1 ← generate population (Ltc,P,fl,fsl).
23. For i=1 to g.
24. F_1 ← evaluateFitness (P_i, t_c,fl,fsl)
25. P_{i+1} ← addTwoBest(F_i, P_i)
26. For j=3 to P.
27. $Parent_1$ ← RandomSelectParent(P_i)
28. $Parent_2$ ← RandomSelectParent(P_i)
29. $Child_1$,$child_2$ ← CrossOver($Parent_1$, $Parent_2$,C_p)
30. $Child_1$ ← Mutation($Child_1$, mp)
31. $Child_2$ ← Mutation($Child_2$, mp)
32. P_{i+1}← addChildren($Child_1$,$child_2$)
33. F_{g+1} ← EvaluateFitness(P_{g+1},tc,fl,fsl)
34. $T_{greatest}$ ← SelectBest Child(F_{g+1},P_{g+1})
35. **Return** $T_{greatest.}$
36. **Measure APSC :**
37. C←n*m (n←Lts)
38. N_2←2*n
39. S_1←sum/c (sum=0)
40. S_2←1/(2*n)
41. Apsc←1-(S_1+S_2)
42. Apsc←Apsc*100
43. **Return** Apsc

Figure 3: Algorithm1 (Adaptive Genetic Hybrid Algorithm)

In figure 4 represent the flow chart of our prosed algorithm the gray shaded area in flow chart represent the adaptive approach and rest part applied by us that is genetic algorithm .

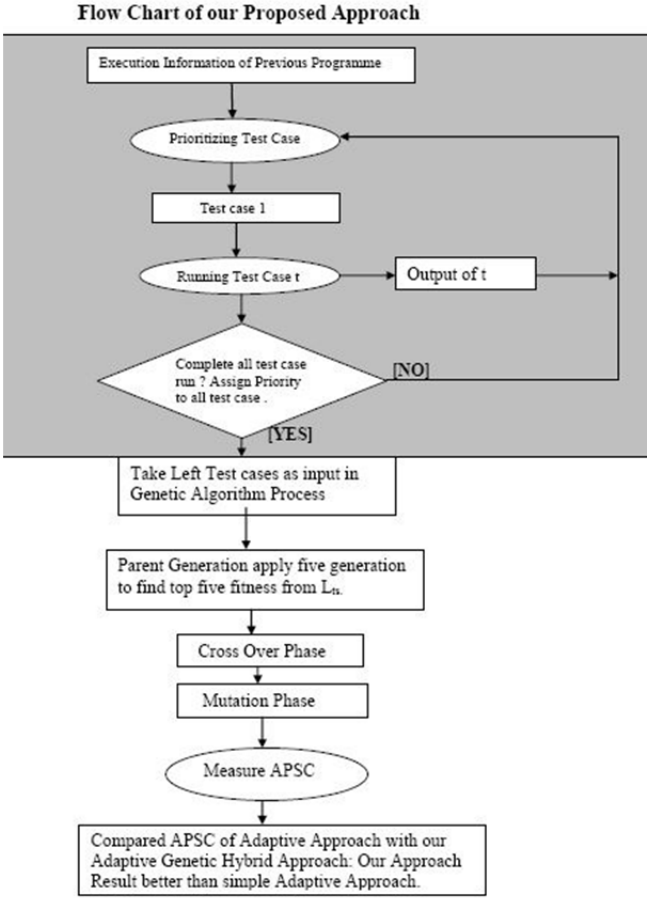

Figure 4 : Flow Chart of our Proposed Approach(Research Methodology)

Adaptive Genetic Algorithm Hybrid proposed test-case prioritization approach in this approach we ordering the test case and find the average percentage of statement coverage for hundred test cases in java . First we measure the APSC of adaptive approach and ordering the test case. In adaptive approach we order the test case like until our statement not cover if test cases left or we can say failure test cases those are unable to cover any statement its means the statement coverage is not done perfectly . We take that Left test cases after applying adaptive approach and perform genetic algorithm on these test case. In Genetic algorithm we apply three main techniques to order the test case like this our APSC improved as compared to adaptive approach. We apply these techniques in genetic algorithm to giving the order to each test case

3.1 Adaptive Approach
3.2 Parent Generation
3.3 Cross Over
3.4 Mutation
3.5 Measure APSC
3.6. Execution time

3.1 Adaptive Approach

In this Research Methodology, we first present the adaptive process of the existing test-case prioritization approach by howing its basic difference with our proposed approach adaptive genetic hybrid approach and then give the details of the adaptive genetic hybrid approach in below sections. For ease of exhibition, we present the adaptive genetic hybrid test case ordering approach in terms of statement coverage, which can also be implemented on other adaptive approach also. In figure 8: the dark area of flow chart represent the adaptive approach methodology the rest for flow chart is further methodology of Genetic algorithm. The overall flowchart figure 3. Represent the our adaptive genetic hybrid approach methodology.

We take hundred apache server test cases Antloader package of test cases in java IDE Eclipse.

First we set each test case priority 1.

Priority (t) = Σ Potential(S) ---------- (1)

Where potential(s) represent how likely statement covered by the existing selected test case. Potential(S) of any statement S in which scope [0,1].

$$\text{Potential(S)} = \begin{cases} \textbf{If test case(t') passed then,} \\ \text{Potential(s) , s is not executed by t'.} \\ \text{Potential(s)*q , s is executed by t'} \\ \textbf{If test case(t') failed then,} \\ \text{Potential(s)*P , s is executed by t'} \end{cases}$$

P and q are two non-negative constants whose values are between 0 and 1. In our implementation process while all test case priority set 1 in initial than, we run all test case those test case cover statement we provide "G" to that test case. Those test case gain maximum number of G their priority must be high. so according to this process we found that in this approach by running hundred test cases few test cases cover the statements on that bases we calculate APSC of this approach.

The effect of passed/failed output on the $Potent(s)$ of any statement s is measured by q/p in the earlier equation. Moreover, when p=q=0, the adaptive approach becomes the additional statement-coverage based test-case prioritization approach, whereas when p=q=1, the adaptive approach becomes the total statement-coverage based test case ordering approach. That is, the total or additional statement-coverage based test-case ordering approach can be viewed as an instance of the adaptive approach. The existing research on test-case prioritization has fully evaluated the effectiveness of the total approach and the additional approach. Although p and q in the preceding equation are two independent variables, to facilitate evaluation of the proposed adaptive approach, currently we assume $p+q = 1$ in this research and evaluate the effectiveness of the adaptive approach by setting q=0, 0.2, 0.4, 0.6, 0.8, or 1,

Than we calculate APSC for adaptive approach by applying APSC formula .

$$APSC = 1 - \frac{Ts_1+Ts_2+\ldots\ldots\ldots\ldots Ts_{sm} + 1/2*n}{n*m}$$

3.2 Parent Generation : This is the first step of genetic algorithm of parent selection we apply this process only on the remaining test cases after adaptive approach for the selection of five top parents we set priority to each test case according to the statement coverage we calculate fitness In our proposed algorithm 1 Pg + 1, is produced, the fitness value of each chromosome is determined on line 33. In Algorithm1 , and the chromosome whose fitness value is the greatest is selected to be the test order. We select parent randomly Algorithm 2 show that how parent selection process going on.

Algorithm 2: Random Parent Selection Algorithm

Input : Pi the population of the ith generation.
output : Parent chromosome selection
1. FitnessSum← calculate fitnessSum of chromosome(Pi)
2. r← generate random number(FitnessSum)
3. for K=1 to P
4. r← r-fitof Chromosomek
5. if r < 0
6. Break
7. Parent← chromosomek
8. Return Parent

Figure 5: Random Parent Selection Algorithm.

As the above Figure 5 Algorithm 2 states that first we input the population of the rest of test cases after adaptive approach in1st generation we apply five generation in our experiment. According to above algorithm first we take randomly chromosomes. We take two highest priority test cases from previous adaptive approach as Parent1 and Parent 2. While we calculate the fitness and the highest fitness test case become the next parents of nest generation. Like this process we got five highest parents with high fitness value. After performing first generation we not consider that highest parent fitness in second generation. Same like this after getting second highest fitness value we don't consider that test case in third so on until we not complete all process for each test case.

$$Fitness = 1 - \frac{Ts_1+Ts_2+\ldots\ldots\ldots\ldots Ts_{sm} + 1/2*n}{n*m}$$

3.3 Cross Over : After completion of first step of Genetic algorithm we get two Parents of high fitness value now we will perform cross over operation in our proposed approach. Crossover is ordinarily a recombination transform that consolidates the portions of one chromosome with the sections of another. The new chromosomes framed by hybrid acquire a few qualities from both folks. The calculation of the hybrid administrator is given in Fig. 3.5. The calculation is the single point hybrid. In the first place, an arbitrary number, r, which extends from 0 to 100, is created on line 1. On the off chance that r is not exactly the hybrid likelihood, cp, the recombination procedure will start on line 3. Something else, the kid is the copy of the guardian. At the point when hybrid is connected, the calculation chooses hybrid focuses, p1 and p2, for parent1 and parent2, separately, on lines 3 and 4. On lines 5 and 6, the subsequences before the hybrid point are then duplicated

from both folks. The joined capacity on lines 7 and 8 creates a tyke by consolidating the duplicated subsequence of one guardian with the qualities of another guardian that are not in the replicated subsequence.

Input: $Parent_1$: selective Chromosome from population
$Parent_2$: Another Selective chromosome from population
C_p : CrossOver Point
Output: $Children_1$, $Children_2$ (two new chromosomes produced by algorithm)
1. n← generateRandomNumber(100)
2. if n < C_p
3. P_1 ← select Crosspoint($Parent_1$)
4. P_2← select CrossPoint($Parent_2$)
5. $Segment_1$ ← fragment(P_1,$Parent_1$)
6. $Segment_2$ ← fragment(P_2,$Parent_2$)
7. $children_1$← join($Segment_1$, $Parent_2$)
8. $children_2$← join($Segment_2$, $Parent_1$)
9. **else**
10. $children_1$← $Parent_1$
11. $children_2$← $Parent_2$
12. **Return** $children_1$ $children_2$

Figure 6: Algorithm 3 Cross over

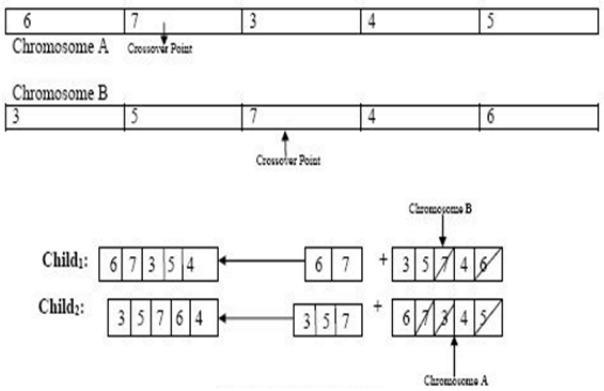

Figure 7: Example of Crossover

For example, considering the chromosomes in Fig 7 demonstrates their hybrid process. The hybrid purposes of An and B are at positions 2 and 3, individually. Child1 gets the subsequence before the hybrid point from An, and the rest from B. Since 6 and 7, which are qualities of B, are additionally in the subsequence duplicated from A, they are not added to the child1. So also, child2 acquires the subsequence before the hybrid point from B, and the qualities that are not in that subsequence from A.

3.3.4 Mutation: Mutation is performed on the chromosomes got by the hybrid process First, the transformation (childrenc, mp) creates a number, n, which goes from 0 to 100 on line 1. On the off chance that n is not exactly the change likelihood, mp, the calculation chooses two qualities of childrenc arbitrarily and swap their positions, as demonstrated in Fig.8. Something else, the transformation administrator would not be connected.

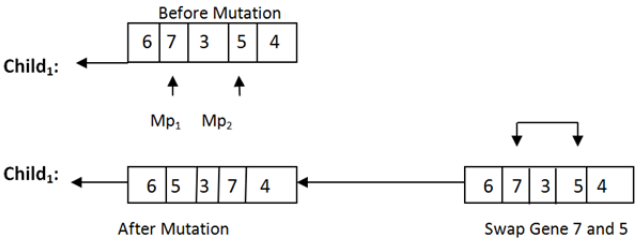

Figure 8 : Example of Mutation

Algorithm 4: Mutation
Input: $Children_c$ chromosome produced by crossover .
M_p Mutation Point
Output: $Children_m$, chromosome produced by algorithm
1. N← GenerateRandomNumber(100)
2. if N<M_p
3. Mp_1 , Mp_2 ←select Mutation Points
4.$Children_m$ ← SwapPosition(Mp_1 , Mp_2, $Children_c$)
5. else
6.$Children_m$← $Children_c$
7. **Return** $Children_m$

Figure 9: Algorithm 4 Mutation Algorithm

In Mutation phase of genetic algorithm paper we take both children chromosome generated by the crossover operator. We show in figure 3.8, how the children change after applying mutation operator. First we take $children_1$ and randomly generate number for two different mutation points . as in example Mp_1 and Mp_2 indicate gene 7 and 5 in above example. We simply swap these genes and got $children_{m1}$, and $Children_{m2}$. same process going on for each chromosomes we received after cross over operator/phase . The mutation ($child_c$, m_p) also gives those test cases a chance to get a higher priority for test case ordering .

3.5 Measure APSC: The fifth step of our methodology is measuring the average percentage of statement coverage which will show our experimental work, the result of APSC represent how our approach is better than adaptive approach. The general formula to measure APSC.

$$APSC = 1 - \frac{Ts_1+Ts_2+\ldots\ldots\ldots..\ T_{sm} \quad + \quad 1/2*n}{n*m}$$

but in our experimental coding we apply this formula like .

$$APSC = 1 - sum/c + 1/2*n$$

Where , n= number of test case (L_{tc} left test cases after adaptive approach)
M= statements
C= n*m
S_1 = sum/n*m , sum/C .
S_2 = 1/(2*n)
APSC= 1-S_1 +S_2 .
We take all high order test cases to evaluate the apsc for our proposed approach, we tak summation of each test-case priority calculate by our algorithm 1. On the basis of that prority reading we measure APSC and our results shows that our proposed approach is better than the previous adaptive approach. We are able to increase the efficiency of average percentage of statement coverage. Our results show in graphical form in the chapter Result and Analysis.

3.6 Execution time

The last parameter measures in this research is execution time .we calculate how much execution time should be taken by existing approach and proposed approach .and we found that execution time is high in our proposed approach because this is generally clear as well it should take more time than adaptive approach because we execute adaptive approach as well genetic algorithm process in which five parent generation , crossover and mutation operators processing in our proposed approach .we also found that execution time depend on the system configuration also . while we process this approach on high configuration system it take less time while we process on law configuration system it take a lot of time . so we conclude this parameter in our future scope we can improve APSC as well time execution if we apply any other approach/technique further.

4. RESULT AND GRAPHS

In this chapter we will discuss about the result obtained by us for both existing approach as well our proposed approach. In existing approach of adaptive test-case prioritization we calculate APSC (average percentage of statement coverage) and execution time also by vary the q and p value .

The existing research on test-case prioritization has fully evaluated the effectiveness of the total approach and the additional approach. Although p and q in the preceding equation are two independent variables, to facilitate evaluation of the proposed adaptive approach, currently we assume $p+q = 1$ in this research we evaluate the effectiveness of the adaptive approach and proposed approach by setting q=0, 0.2, 0.4, 0.6, 0.8, and 1 . We focus on Q factor value just because the q factor value multiply only when test case is pass. Same like that we calculate the execution time for the adaptive approach and proposed approach by setting q=0, 0.2, 0.4, 0.6, 0.8, and 1 .

Table 4.1 Adaptive Approach by different p, q factor value.

Q	p	APSC	Execution time
0	1	97.6097052	1343 ms
0.2	0.8	**98.6940925**	1047 ms
0.4	0.6	98.6645584	**859 ms**
0.6	0.4	98.6645584	969 ms
0.8	0.2	98.6645584	1214 ms
1	0	98.61392425	1191 ms

As it shown in table number 4.1 while we take different p, q factor values we get different average percentage of statement coverage and execution time. We found that the highest APSC is at q=0.2, p=0.8 value while the minimum time taken at q=0.4, p=0.6. so from this table we analyses we can change the factor value according to our need in which we have need to focus. If our focus on to statement coverage we take best value of p, q in which we get highest APSC. While we have needed to focus on execution time we will select p, q value according to the low executions time value.

Figure 10, represent the value of APSC on Y-axis while there is different Q factor value on x-axis. While in figure 11 the graph resent the execution time take by adaptive approach while we measure APSC.

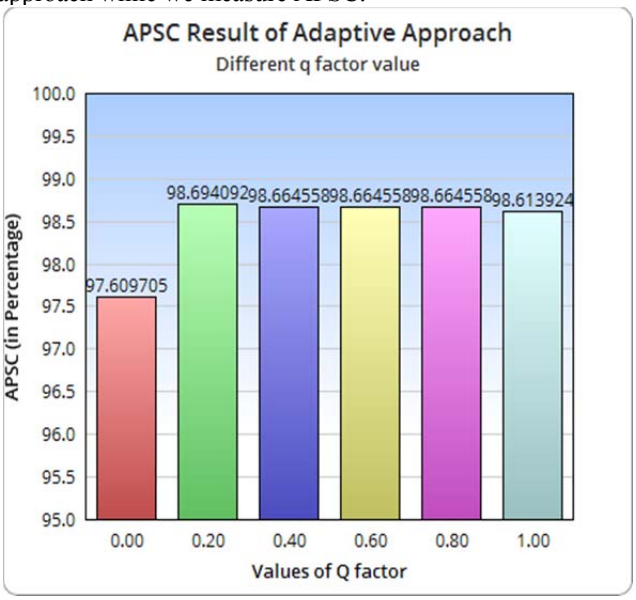

Figure 10: Graph of APSC according to Different Q values in Adaptive Approach.

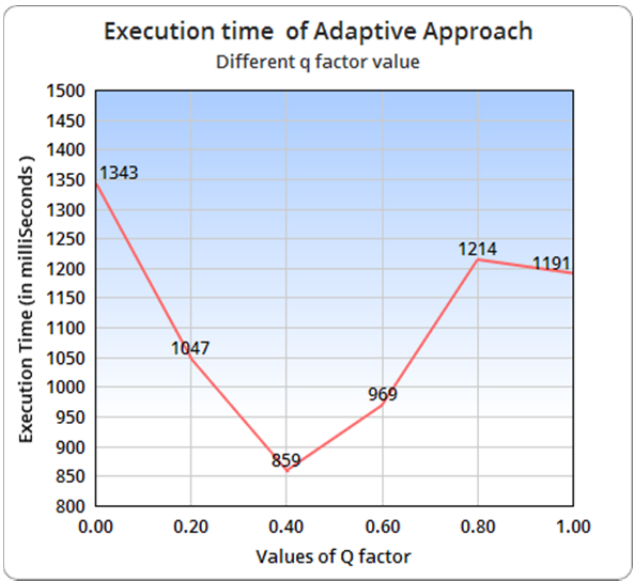

Figure 11: Graph of Execution Time according to Different Q values in Adaptive Approach

After measuring the APSC and execution time for adaptive approach. Than we measure the APSC and execution time for our proposed approach adaptive genetic hybrid approach. In Table number 4.2 the values of APSC and Execution time date at different Q and P factors. On the basis of this data set figure 12 represent the graph of APSC values on y axis and different Q factor values on x-axis. Similarly figure 13 represent the execution time value on y-axis and different q factor value on x-axis.

Table 4.2 Adaptive Genetic hybrid Approach by different p, q factor value.

Q	p	APSC	Execution time
0	1	99.34755322	41691 ms
0.2	0.8	99.54693411	52706 ms
0.4	0.6	99.67835639	50404 ms
0.6	0.4	99.65760801	48956 ms
0.8	0.2	99.67144225	39178 ms
1	0	99.58897335	51171 ms

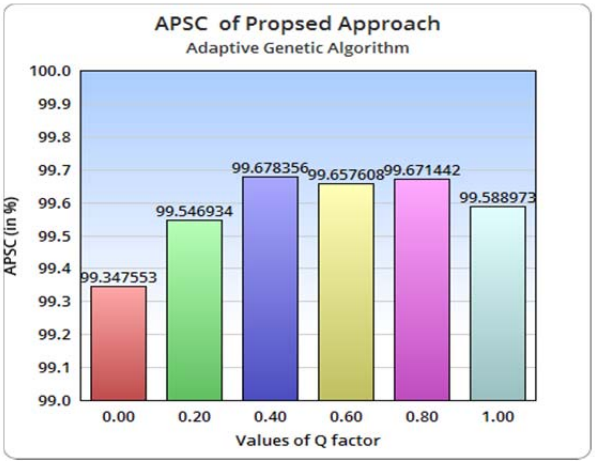

Figure 12: Graph of APSC according to Different Q values in proposed Approach

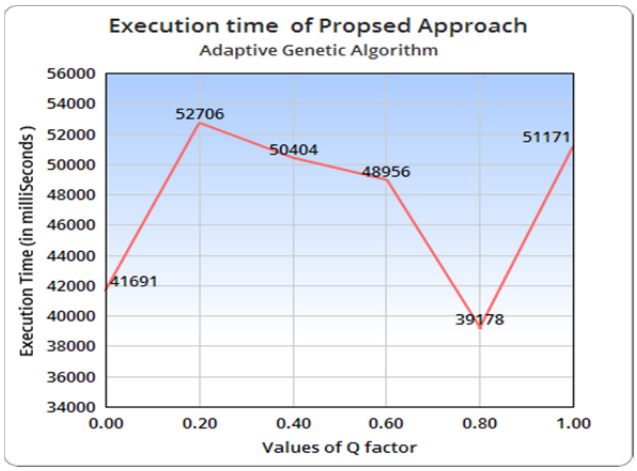

Figure 13: Graph of Execution Time according to Different Q values in Proposed Approach.

In the above figure and tables we represent the separate outcomes from both approach the existing adaptive approach as well our prosed approach. After that we compare the both approach on the basis of APSC as well time execution . In Figure 14 it shows that the comparison between the adaptive and our proposed approach. The red color bar indicate the adaptive approach while green bar represent our proposed approach and it is clear represented by graph our approach is better than existing adaptive technique in this research basically we focus only on APSC rather than execution time.While in figure 15 the line graph represent the time taken comparison between the existing approach as well our proposed approach red line in graph represent the time taken by adaptive approach and green line represent the time taken by our proposed approach.

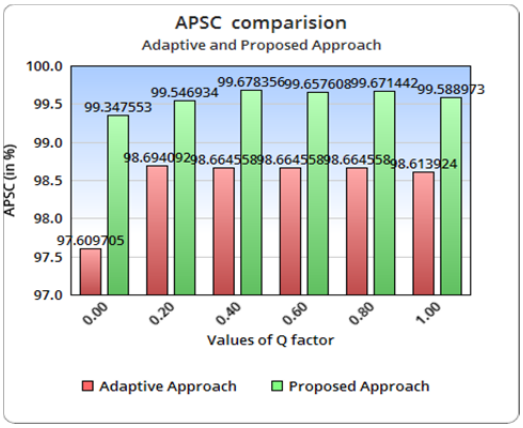

Figure 14: APSC Comparison of Adaptive and proposed Approach.

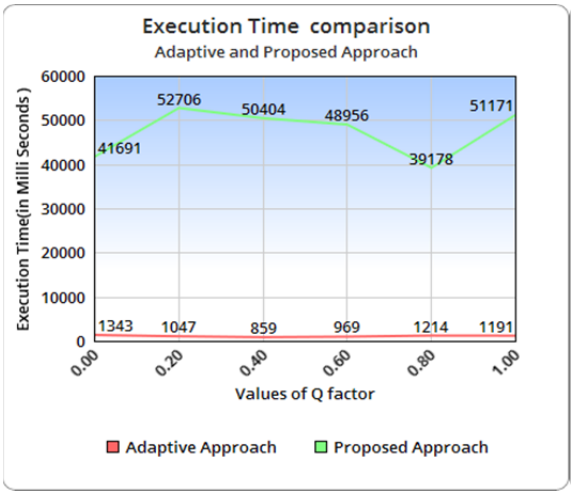

Figure15 : Comparison of Execution Time among Adaptive and Proposed Approach.

5. Conclusion and Future Scope

In this Research we proposed an approach that improves APSC (average percentage of statement coverage). Our work is extension into the adaptive approach for APFD (average percentage of fault detection) into adaptive genetic algorithm hybrid approach from which we conclude that our proposed approach improve the APSC.We take hundred java test cases package of apache server to evaluate our approach. First we apply adaptive approach and calculate APSC. Than we apply our proposed algorithm adaptive genetic algorithm hybrid approach than we calculate APSC than we found that our approach gives better results than adaptive approach for APSC only. Basically in this research we focused on APSC only but while we calculate Execution time for both approach we found that our proposed approach take large time to execute as compare to adaptive approach. But as the tester view our main aim to cover all statements of the code for better quality. So, we considering this work as our next future work and we believe that if we apply any other technique we can improve execution time as well APSC together. And we take small data set in our research while in future we take large data set of test cases for efficient results.

REFERENCES

[1]. A.B Taha, S.M. Thebaut, and S.S. Liu.,"An approach to software fault localization and revalidation based on incremental dat flow analysis". in proceeding of the 13th Annual International Computer Software and Applications Conference..

[2] A. Marback, H. Do, and N. Ehresmann, "An effective regression testing approach for PHP web applications," in Proceedings of the International Conference on Software Testing, Verification and Validation, Apr. 2012, pp. 221–230.

[3] Dan Hao, Xu Zhao, "Adaptive Test-Case Prioritization Guided by Output Inspection" 37th Annual International Computer Software and Applications Conference (COMPSAC 2013), 22-26 July 2013, pages 169-179, Tokyo, Japan

[4] D.Hoffman and C.Brealey. "Module test case generation" in proceedings of the Third Workshop on Software Testing, Analysis, and Verification, pages 97-102, December 1989.

[5] G. Rothermel and M. J. Harrold, Analyzing Regression Test Selection Techniques, IEEE Transactions on Software Engineering, V.22, no. 8, August 1996, pages 529-551.

[6] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizingtest cases for regression testing," IEEE Transactions on Software Engineering, vol. 27, no. 10, pp. 929–948, October 2001.

[7] http://www.ehow.com/facts_5835705_difference-between-operand-operator.html

[8] J. Hartmann and D.J. Robson." Revalidation during the software maintenance phase"in Proceeding of the conference on Software Maintenance.

[9] J.Ziegler, J.M. Grasso, and L.G. Burgermeister." An Ada based real-time closed-loop integration and regression test tool". in Proceedings of the Conference on Software Maintenance -1989, pages 81-90, October 1989

[10] J. Offutt, J. Pan, and J. M. Voas." Procedures for reducing the size of coverage-based test sets" in Proceedings of the Twelfth International Conference on Testing Computer Software, pages 111–123, June 1995.

[11] Keith H. Bennett and V_aclav Rajlich. "Software maintenance and evolution: a roadmap." in Proceedings of the International Conference on Software Engineering (ICSE'00), pages 73{87, 2000}.

[12] K. Onoma, W-T. Tsai, M. Poonawala, and H. Suganuma."Regression testing in an industrial environment".

[13] Mithun Acharya ,"Configuration Selection Using Code Change Impact Analysis for Regression Testing", 28th IEEE International Conference on Software Maintenance (ICSM), 2012

[14] Md. Hossain , "Regression Testing for Web Applications Using Reusable Constraint Values ," IEEE International Conference on Software Testing, Verification, and Validation Workshops , 2014.

[15] Md. Junaid Arafeen and Hyunsook , "Test Case Prioritization Using Requirements-Based Clustering ", IEEE Sixth International Conference on Software Testing, Verification and Validation ,2013

[16] Mitchell Melanie, Fifth printing, 1999 An Introduction to Genetic Algorithms , A Bradford Book The MIT Press , Cambridge, Massachusetts • London, England

[17] M.J. Harrold, R. Gupta, and M.L. Soffa. "A methodology for controlling the size of a test suite. ACM Transactions on Software Engineering and Methodology".

[18] Nicolas Frechette, "Regression Test Reduction for Object-Oriented Software: A Control Call Graph Based Technique and Associated Tool" , Hindawi Publishing Corporation ISRN Software Engineering Volume 2013, Article ID 420394, 10 pages

[19] P.A Brown and D. Hoffman. "The application of module regression testing at TRIUMF. Nuclear Instruments and Methods in Pysics Research", Section A, . A293(1-2):377-381, August 1990.

[20] Prof. A. Ananda Rao and Kiran Kumar J "An Approach to Cost Effective Regression Testing in Black-BoxTesting Environment "IJCSI international journal of computer science issues vol.8 issue 3,No. 1 may 2011

[21] Regression Test Selection by Exclusion ,Durham E-Theses, Durham University.

[22] R. Lewis, D.w. Beck, and J.Hartmann. "Assay – a tool to support regression testing". In ESEC' 89.2nd European Software Engineering Conference Proceedings, pages 487-496,

[23] S. Elbaum, A. G. Malishevsky, and G. Rothermel. "Test case prioritization: A family of empirical studies." IEEE Transactions on Software Engineering, 28(2):159–182,February 2002.

[24] Swarnendu Biswas , "Regression Test Selection Techniques: A Survey" , Informatica 35 (2011) 289–321 289

[25] http://www.chartgo.com

[26] Xuan Lin . "Regression Testing in Research And Practice", University of Nebraska, Lincoln1-402-472-4058

[27] Yu-Chi Huang "A history-based cost-cognizant test case prioritization technique in regression testing , The Journal of Systems and Software 85 (2012) 626– 637.