Journal of Theoretical and Applied Information Technology
15th August 2016. Vol.90. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

EFFECTIVE BRANCH AND DU COVERAGE TESTING THROUGH TESTCASE PRIORITIZATION USING GENETIC ALGORITHM

[1]P.VELMURUGAN, [2]RAJENDRA PRASAD MAHAPATRA
[1]Research Scholar, Department of CSE, SRM University, India
[2]Professor / HOD, Department of CSE, SRM University, India
E-mail: [1]velmtechce2007@gmail.com, [2]hod.cse@srmuniv.ac.in

ABSTRACT

Software testing is all about ensuring an error free execution of the system or module being developed. The testing procedure initially consists of a large number of test cases that are needed to be reduced so that the execution time needed for testing could be minimized. After the reduction has been done, a few test cases are left. Here an important factor becomes the order of the execution of the obtained number of test cases. This ordering is called as test case prioritization. Prioritization of test cases is a process in which test cases are executed in an ordered fashion so as to increase the fault detection rate. There are many types of prioritization techniques used for statement coverage that have been used in past. In this paper, the testcase prioritization technique used for branch coverage and DU pair coverage to improve the effectiveness of testing process. The proposed methodology uses genetic process for prioritizing the test cases to detect the fault as earlier as possible to improve the effectiveness of branch and DU pair testing.

Keywords: *Test case reduction, prioritization, genetic algorithm, DU pair testing, branch testing*

1. INTRODUCTION

Software testing is basically a set of activities conducted with the intent of finding errors in software. Also, software testing is the process that validates and verifies that a program functions properly. One straight forward approach used for testing is to re-run all the existing test cases and detect if there are any errors. But it is practically impossible under the project deadline and also requires a lot of effort. Other alternative is to prioritize test cases according to their relevance for error detection and find an ordered sequence of test cases which contains those test cases first, which is more likely to find errors. Despite software testing being a very important process to be executed, often there is not enough time or resources to execute all planned test cases. In this case, it is desirable to prioritize the test cases in a way that the most important ones are in the first positions in an attempt to guarantee their execution. One of the many ways to perform testing is to order the test case based on some criteria to meet some performance goal. Testers may want to order their test cases so that the test cases with the highest priority (according to some criterion) are run first. So test case prioritization technique do not discard test cases, they can only avoid the drawback of test case minimization techniques.

To optimize the time and cost spent on testing, prioritization of test cases in a test suite can be beneficial [2, 3, 9, 10, 17, 18]. Test case prioritization (TCP) involves the explicit planning of the execution of test cases in a specific order with the intention of increasing the effectiveness of software testing activities by improving the fault detection rate earlier in the software process [17, 18].

Furthermore, Gregg Rothermel [15] has proven that prioritizing and scheduling test cases are one of the most critical tasks performed during the software testing process. He referred to the industrial collaborators reports, which shows that there are approximately 20,000 lines of code, running the entire test cases requires seven weeks. In this situation, test engineers may want to prioritize and schedule the test cases in order that those test cases with higher priority are executed first. Additionally, he [13], [16] stated that test case prioritization methods and process are required, because: (a) the regression testing phase consumes a lot of time and cost to run, and (b) there is not enough time or

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

resources to run the entire test suite (c) there is a need to decide which test cases to run first.

In this paper, genetic algorithm is being used for the prioritization of the test cases. Genetic algorithm is stochastic search technique, which is based on the idea of selection of the fittest chromosome. It is basically a probabilistic search method based on the mechanics of natural selection and natural genetics. GA applied to natural selection and natural genetics in artificial intelligence to find the globally optimal solution to the optimization problem from the feasible solutions. Nowadays GA's have been applied to various domains, including timetable, scheduling, robot control, signature verification, image processing, packing, routing, pipeline control systems, machine learning, and information retrieval.

GA's are characterized by 5 basic components as follow:
1) Chromosome representation for the feasible solutions to the optimization problem.
2) Initial population of the feasible solutions.
3) A fitness function that evaluates each solution.
4) Genetic operators that generate a new population from the existing population.
5) Control parameters such as population size, probability of genetic operators, number of generation etc.

The major differences between one Genetic Algorithm and another lie within the schemes used to represent chromosomes, the semantics of the genetic operators, and the measures used to evaluate their fitness. Yet, these very differences make Genetic Algorithms so complex to design and implement when opposed with most real-world optimization problems.

Always in genetic algorithm a small number of initial population is taken randomly from the whole solutions from which best is to be found out. Only some of the solutions are taken and rest are automatically generated and out of range solutions are rejected. Fitness function is made according to the problem. Like in knapsack problem fitness is evaluated by summing the benefit or value of the selected item in a solution. In simple real number search problem ,where best solution is the number with maximum value. Value of the number serves as the fitness function. In traveling salesman problem the distance of the whole tour acts as fitness function.

In the process of implementing genetic algorithm the first thing that needs to be done is deriving the fitness function. In this paper fitness evaluation is done using the function:

$$\text{Fitness value (ti)} = \sum_{k=s1}^{s12} \text{wt(k) X init(k)} \quad (1)$$

Genetic operators are basically three:

1. selection
2. crossover
3. mutation

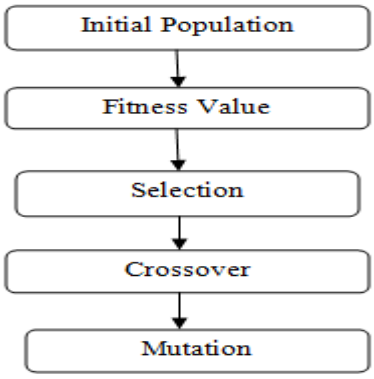


Fig 1 Genetic Algorithm

1. After we evaluate population's fitness, the next step is chromosome selection. Selection embodies the principle of 'survival of the fittest'. Selection procedure in this paper is accomplished using the Rowlett wheel.

2. Crossover is the genetic operator that mixes two chromosomes together to form new offspring. Crossover occurs only with some probability (crossover probability). Chromosomes are not subjected to crossover remain unmodified. The intuition behind crossover is exploration of new solutions and exploitation of old solutions. GA's construct a better solution by mixing good characteristic of chromosomes together.

3. Mutation involves the modification of the values of each gene of a solution with some probability (mutation probability). In accordance with changing some bit values of chromosomes, give the different breeds. Chromosomes may be better or poorer than old chromosomes. If they are poorer than old chromosomes, they are eliminated in selection step. The objective of mutation is restoring lost and exploring variety of data.

Journal of Theoretical and Applied Information Technology
15th August 2016. Vol.90. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2. LITERATURE SURVEY

In various research works performed over test case prioritization a number of techniques have been introduced.

Yu-Chi Huang et al has proposed a cost cognizant test case prioritization technique which is based on the use of historic records and genetic algorithm [1]. They run a controlled experiment to evaluate the effectiveness of the proposed technique. The technique however does not take care of the test cases similarity.

Sayogita Chaturvedi, A. Kulothungan has proposed a fault detection capability for statement coverage through test case prioritization using genetic algorithm. But in this paper, they are not considered branch and DU pair coverage [19].

Another method is a Coverage-based technique that consists of methods to prioritize test cases based on coverage criteria, such as requirement coverage, total requirement coverage, additional requirement coverage and statement coverage. Many researchers have researched in this area, such as Leon [4], Rothermel [5], [7] and Bryce [11].

There is a technique known as the code coverage based TCP Strategies. Coverage based TCP done their prioritization based on their coverage of statements [6]. For Prioritizing statement coverage the test cases are ordered for execution based on the number of statements executed or covered by the test case such that the test cases covering maximum number of statements would be executed first. Some of the other techniques used are branch coverage and function coverage. In this method test cases are prioritized based on their number of branch or function coverage by a test case respectively.

3. PROBLEM DESCRIPTION

3.1. Existing Problem

Previous work on test case prioritization demonstrates that prioritization techniques are effective for improving rate of fault detection. However, these approaches do not consider test suites that contain functional dependencies between tests. Functional dependencies are the interactions and relationships among system functionality determining their run sequence. As test cases mirror this functionality, they also inherit these dependencies; therefore, executing some test cases requires executing other test cases first. Such techniques which do not consider the existence of functional dependencies among test cases uses any order like breadth first search or random order for the prioritization of test cases. Using these techniques the achieved fault detection rate is not high and so these prove to be inefficient as the main goal of a testing case prioritization is to find maximum number of faults in least possible time. But many of the existing techniques haven't used testcase prioritization for branch and DU pair coverage technique.

3.2. Proposed Solution

In this paper, prioritization of test cases is to be done using genetic algorithm where the fitness function is calculated using the initial population which basically tells about the branch and DU pair statements executed by each test case and the weight assigned to each statement. The weight is assigned depending upon the criticality of the statement of introducing an error. The basic operators of genetic algorithm, i.e selection, crossover & mutation are used in order to receive the prioritized list of test cases. When the execution of test cases is done in the order as in the prioritized list, it is expected to give an increased rate of fault detection in branch and DU pair coverage testing.

4. BLOCK DIAGRAM OF A PROPOSED SYSTEM

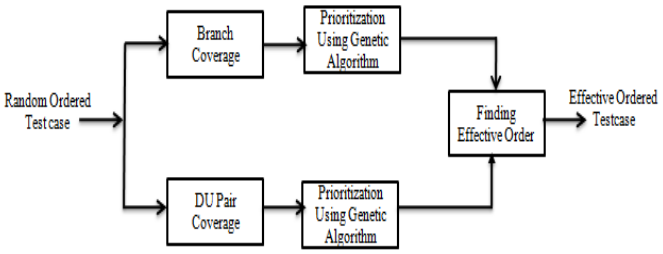


Figure1. Block Diagram

Figure 1. shows that overall module of this proposed approach. Initially Both branch and DU pair coverage uses randomly ordered testcases and find number of branches and Definitions and Use (DU) pair covered by random ordered testcases. These random ordered testcases of both branch and DU pair prioritized by genetic algorithm. The proposed technique finding the effective ordered testcases for both branch and DU pair by combining prioritization of both.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

5. PROPOSED TECHNIQUE

In this paper, genetic algorithm has been used for the prioritization of the test cases existing in a reduced test suite. Firstly a small code is taken which performs few simple operations. A reduced test suite consisting of very few numbers of test cases that are needed to be executed is then obtained. These reduced test cases are then allotted with their initial population which is decided on the basis of the statements executed by each test case. Then after fitness value for every test case is calculated. The fitness value depends on the initial population and the weight of every statement in the source code taken.

Further all of the test cases are placed in a Rowlett wheel and in every cycle,

1. A pair of test cases is taken out, XOR operation is performed over them.
2. If the result of XOR does not give all 1's then firstly crossover operation is performed on both the test cases and then they are again checked for all 1's applying the XOR operation.
   - If the result of XOR after crossover give all 1's then the case with higher fitness value is put in the prioritized list and the other is put back in the Rowlett wheel(If both the test case have same fitness value then any one of them is chosen randomly to be sent in the prioritized list while the other is held back to be XORed with the test case received in next cycle).Goto 1.
   - If the result of XOR after crossover does not give all 1's then both the test cases are operated with mutation.Then again XOR operation is performed to check for receiving all 1's.
   - If the result of XOR after mutation give all 1's then the case with higher fitness value is put in the prioritized list and the other is put back in the Rowlett wheel(If both the test case have same fitness value then any one of them is chosen randomly to be sent in the prioritized list while the other is held back to be XORed with the test case received in next cycle). Goto 1.
   - If the result of XOR after mutation does not give all 1's then the test case with higher fitness value is put in the prioritized list and the other is put back in the Rowlett wheel. Goto 1.
3. If a success occurs ,i.e result of XOR give all 1's then both the cases are taken out of the Rowlett wheel. Any one of both the cases is randomly sent in the prioritized list and the other is kept to perform XOR operation with a test case received in a later cycle of the above stated operation.

The above operation continues to be performed until the Rowlett wheel becomes completely empty which means all the test cases are placed in prioritized list. Then the execution of the test cases is done based on the order defined in the prioritized list of the test cases.

5.1. Algorithm for Branch Coverage

1. Ordered test suite, toc=ϕ
2. while(trc != ϕ)
3. {for trc= 1 to n
   toc=ϕ
4. {for j=1 to n
5. { if val[j]==1
6. Ti= {t Ụ s[j] ; if seen(trc)
   { ϕ ; otherwise
   } }
7. No. of statements executed by trc = trc}
8. max toc = find max[{trc}]
9. Toc =toc Ụ max toc
10. if(trc ==max toc || trc Є toc)
11. trc=seen(trc)
12. Goto 2}

where trc = random test cases{1….n}
j=branch no.

5.2. Algorithm for DU Pair Coverage

1. Find variable in function
2. For every statement (S_i) in program
   do
   for every variable (V_j)
   do list DU statement
   for every DU_k statement
   if(DU_k is covered by Tm)
   mark executed
   else
   mark failed
   end loop
   end loop end loop

Journal of Theoretical and Applied Information Technology
15th August 2016. Vol.90. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

S_i = Set of Statements ($S_1 \ldots S_n$)
V_i = Use of variable for every statement ($V_1 \ldots V_n$)
T_i = List of testcases ($T1 \ldots T_n$)

6. CASE STUDY

In this paper, a small code is taken which performs simple operation and three input variables are needed namely A, B and C. In this case study we have taken six different testcases and ordered randomly. Then unordered testcases is effectively ordered with genetic algorithm to decide the priority of each test case so that the execution of test cases could be done according to the assigned priority. This ordered execution helps in increased rate of fault detection in least possible time.
Every statement in the source code is assigned a weight depending upon its possibility of introducing an error. This weight is used in calculation of fitness value of the test case.

6.1. Source Code:

```
mid() {                          Statement No      Weight (Wi)
    int a,b,c,n;
read(a,b,c);                     1                 0.9
n=c;                             2                 0.9
if(b>c)                          3                 0.8
    if(a<b)                      4                 0.2
        n=b;                     5                 0.2
    else if(a<c)                 6                 0.3
        n=b;                     7                 0.4
else                             8                 0.4
    if(a>b)                      9                 0.1
        n=b;                     10                0.1
    else if(a>c)                 11                0.3
        n=a;                     12                0.3
    print(n);                    13                0.2
}
```

Figure2. Source Code with Statement and weight

The test suite taken is:
T={t1,t2,t3,t4,t5,t6}

Table 1.Testcase and Values

Testcase	Test value for a, b, c
T1	3, 3, 5
T2	1, 2, 3
T3	3, 2 ,1
T4	5, 5, 5
T5	5, 3, 4
T6	2, 1 ,1

6.2. Branch Coverage for randomly ordered testcase

Table2. Individual Testcases Branch Coverage

Branch statement	T1	T2	T3	T4	T5	T6
Start	1	1	1	1	1	1
3T	1	1	0	0	1	1
3F	0	0	1	1	0	0
4T	0	1	0	0	0	0
4F	1	0	0	0	1	1
6T	1	0	0	0	0	1
6F	0	0	0	0	1	0
9T	0	0	1	0	0	0
9F	0	0	0	1	0	0
11T	0	0	0	0	0	0
11F	0	0	0	1	0	0

T – True branch
F – False branch
If Ti is executed then 1 otherwise 0

6.3. DU Pair Coverage for randomly ordered testcase

Table3.Individual Tetscases DU Pair Coverage

Statement No	DU statement	T1	T2	T3	T4	T5	T6
1	1, 2, c	1	1	1	1	1	1
2	1, 3, b	1	1	1	1	1	1
3	1, 3, c	1	1	1	1	1	1
4	1, 4, a	1	0	0	0	0	1
5	1, 4, b	1	0	0	0	0	1
6	1, 5, b	0	0	0	0	0	0
7	1, 6, a	1	0	0	0	1	1
8	1, 6, c	1	0	0	0	1	1
9	1, 7, a	1	0	0	0	0	1
10	1, 9, a	0	0	1	1	0	0
11	1, 9, b	0	0	1	1	0	0
12	1, 10, b	0	0	1	0	0	0
13	1, 11,a	0	0	0	1	0	0
14	1, 11, c	0	0	0	1	0	0
15	1, 12, a	0	0	0	0	0	0
16	2, 13, n	0	0	0	1	1	0
17	5, 13, n	0	1	0	0	0	0
18	7, 13, n	1	0	0	0	0	1
19	10, 13, n	0	1	0	0	0	0
20	12, 13, n	0	0	0	0	0	0

Journal of Theoretical and Applied Information Technology
15th August 2016. Vol.90. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

6.4. Initial Population for Branch Coverage

Initial population of every test case is calculated which is basically the output that shows what are the statements that are being executed by each test case. If a test case executes a statement then a value 1 is assigned to that test case corresponding to the respective statement else a value 0 is assigned. The Initial population of the test cases is given below:

Table4. Initial Population (Branch Coverage)

Test case	Statements										
	S 1	S 2	S 3	S 4	S 5	S 6	S 7	S 8	S 9	S 10	S 11
T1	1	1	0	0	1	1	0	0	0	0	0
T2	1	1	0	1	0	0	0	0	0	0	0
T3	1	0	1	0	0	0	1	1	0	0	0
T4	1	0	1	0	0	0	0	0	1	0	1
T5	1	1	0	0	1	0	0	0	0	0	0
T6	1	1	0	0	1	0	0	0	0	0	0

6.5. Initial Population for DU Pair Coverage

Table4. Initial Population (DU Coverage)

TestCase	Statements																			
	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20
T1	1	1	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0
T2	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
T3	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
T4	1	1	1	0	0	0	0	0	0	1	1	0	1	1	0	1	0	0	0	0
T5	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0
T6	1	1	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0

6.6. Fitness value calculation for branch

The next step is to calculate the fitness value of every test case. It is calculated using the formula given:

$$\sum_{k=s1}^{11} wt(k) \times init\ (k) \qquad (2)$$

Table5. Fitness function Value (Branch)

TestCase	T1	T2	T3	T4	T5	T6
Fitness Value	2.3	2.0	2.1	2.1	2.4	2.0

6.7. Fitness value calculation for DU pair coverage

$$\sum_{k=s1}^{20} wt(k) \times init\ (k) \qquad (3)$$

Table6. *Fitness function Value (DU Pair)*

TestCase	T1	T2	T3	T4	T5	T6
Fitness Value	11.7	5.9	8.2	10.7	8.7	11.7

6.8. Genetic Loop for branch coverage

After the initial population and fitness value is calculated, all of the test cases are put in Rowlett wheel. In every genetic cycle, a pair of test case is taken out from the Rowlett wheel. Both of the test cases are operated with XOR operator taking their respective initial population. In the end of every loop a test case is received which is put in the prioritized list. The test case being included in the list earlier has a higher priority while the one included later has a lower priority. The genetic loop runs until the Rowlett wheel becomes completely empty which means none of the test cases are left in the wheel and all of them are placed in the prioritized list of test cases.

Firstly, test case t1 and t6 are taken out of the Rowlett wheel and OR operation is performed over their initial population.

t1 11001100000
OR = OR = 11001100000
t6 11001000000

Now, since the output received does not consist of complete 1's or target is not achieved, both the test cases are sent for crossover operation.

11001100| 000
110 | 01000000

After crossover:

11001100110
OR = 11001100110
00001000000

Since the resultant output does not have complete 1's or target not achieved, mutation operation is performed over cross over result.

Mutation applied on 9th and 10th bit of cross over result: 11001100110

Hence, there is an output with all 1's not came and target not achieved and so the test case with higher fitness value is send to the prioritized list. Then

Journal of Theoretical and Applied Information Technology
15th August 2016. Vol.90. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

lower fitness value testcase and one new testcase from Rowlett wheel will go for next genetic loop. This process will continue until there is no test case left.
According to our testcase after the completion of entire process the test case in the prioritized list are as:

Priority List1 = {t5 > t4 > t2 > t1 > t3 > t6}.

The list obtained shown above denotes the order in which the test cases are needed to be executed in order to improve the fault detection rate of the testing process.

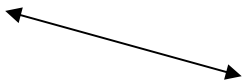
6.9. Genetic Loop for DU pair coverage

After the initial population and fitness value is calculated, all of the test cases are put in Rowlett wheel same as genetic loop for branch coverage.

For Example

Firstly, test case t1 and t5 are taken out of the Rowlett wheel and OR operation is performed over their initial population.
t1 1111101110000000000100
OR = OR =
1111101110000000010100
t5 1110001100000000010000
Now, since the output received does not consist of complete 1's or target is not achieved, both the test cases are sent for crossover operation.

111 | 11011100000000100



11100011000000010 | 000

After crossover:

00011011100000000100

OR = 1111101110000000010111

11100011000000010111
Since the resultant output does not have complete 1's or target not achieved, mutation operation is performed over cross over result.
Mutation applied on 9th and 10th bit of cross over result:

11111011110000010111

Hence, there is an output where all 1's did not come hence target not achieved, so the test case with higher fitness value is send to the prioritized list. Then lower fitness value testcase and one new testcase from Rowlett wheel will go for next genetic loop. This process will continue until there is no testcase left.
According to our testcase after the completion of entire process the test case in the prioritized list PL are as:

Priority List 2= {t5 > t6 > t4 > t2 > t3 > t1}.

The list obtained shown above denotes the order in which the test cases are needed to be executed in order to improve the fault detection rate of the testing process.

6.10. Finding effective order for DU and Branch Coverage

Priority List 1 = {t5 > t4 > t2 > t1 > t3 > t6}.
Priority List 2 = {t5 > t6 > t4 > t2 > t3 > t1}.

From Priority List 1 and Priority List 2 t5 has higher priority in both Priority List 1 and Priority List 2 so we can choose t5 in the effective ordered set. Then t4 has second highest priority in Priority List 1 but third highest priority in Priority List 2 and t6 has second highest priority in PL2 but last priority in Priority List 1 so we can choose t4 before t6 because t4 (3-2) has lowest priority difference than t6 (6-2) in Priority List 2. Similarly same process have to apply for remaining test cases. The resultant order is

Effective Priority List = {t5, t4, t2, t6, t1, t3}

7. RESULT ANALYSIS

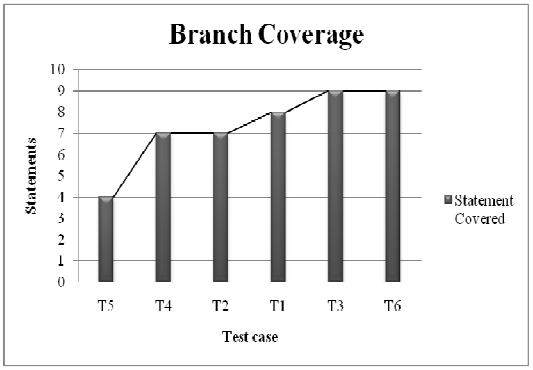


Figure3. Individual testcases statement coverage (Branch)

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

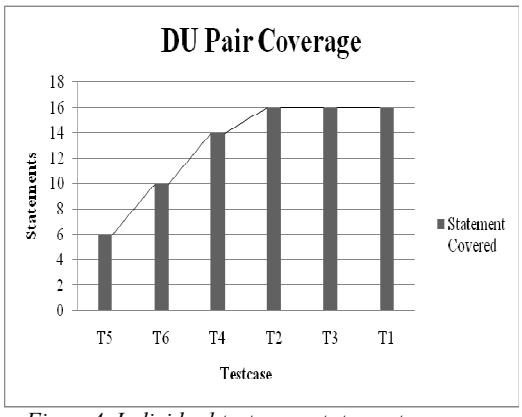


Figure4. Individual testcases statement coverage (DU Pair)

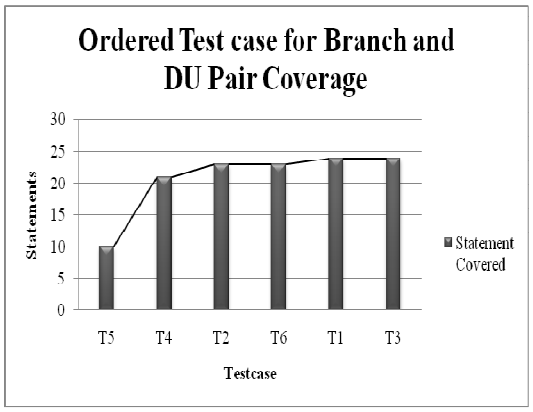


Figure5. Ordered Testcases Coverage(Both Branch and DU Pair)

8. Conclusion and Future Work

In this paper, we have taken one case study in which Genetic Algorithm and random ordered techniques has been used for testcase prioritization in both branch and DU pair Coverage testing. But this research mainly focused genetic algorithm for testcase prioritization by which effective order testcase has been generated and analyzed with random prioritization technique. Finally this analysis proven that effective order suite is better order for testing both branch and DU pair coverage. But in this order some higher order statements are not covered effectively as earlier possible. In future, this research focuse on above problem and try to resolve it.

REFERENCES

[1] Yu-Chi Huang, Chin-Yu Huang, Jun-Ru Chang and Tsan- Yuan Chen "Design and Analysis of Cost-Cognizant Test Case Prioritization Using Genetic Algorithm with Test History", IEEE 34th Annual Computer Software and Applications Conference 2010.

[2] F. Basanieri, A. Betolino, and E. Marchetti, "CoWTeSt: A Cost Weighed Test Strategy," Escom-Scope 2001, London, England, April 2001, pp. 387-396.

[3] F. Basanieri, A. Betolino, and E. Marchetti, "The Cow_Suite Approach to Planning and Deriving Test Suites in UML Projects," Fifth International Conference on the Unified Modeling Language - the Language and its applications UML 2002, Dresden, Germany, September 2002, pp. 383-397.

[4] Bogdan Korel and Ali M. Al-Yami, "Automated Regression Test Generation", ISSTA98, 1998.

[5] B. Korel and J. Laski, "Algorithmic software fault localization", Annual Hawaii International Conference on System Sciences, pages 246–252, 1991.

[6] Zheng Li, Mark Harman, and Robert M. Hierons, "Search algorithm for Regression Test Case Prioritization," IEEE Transactions on Software Engineering, Vol. 33, No.4, April 2007.

[7] Cem Kaner, "Exploratory Testing", Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL, 2006.

[8] Harsh Bhasin, Surbhi Bhatia: Use of Genetic Algorithms for Finding Roots of Algebraic Equations. International Journal of Computer Science and Information Technology, 2011. Volume 2, Issue 4, pages 693-696.

[9] S. Elbaum, A. Malishevsky, and G. Rothermel, "Prioritizing Test Cases for Regression Testing," *Proceedings of the ACM International Symposium on Software Testing and Analysis*, vol. 25, no. 5, pp. 102- 112, August 2000.

[10] S. Elbaum, A. Malishevsky, and G. Rothermel, "Test Case Prioritization: A Family of Empirical Studies," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159-182, February, 2002.

[11] Hyunsook Do and Gregg Rothermel, "A Controlled Experiment Assessing Test Case Prioritization Techniques via Mutation Faults", *Proceedings of the IEEE*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

International Conference on Software Maintenance, pages 411-420, 2005.

[12] J. Li, "Prioritize Code for Testing to Improve Code Coverage of Complex Software," Proc. 16th IEEE Int'l Symp. Software Reliability Eng., pp. 75-84, 2005.

[13] B. Korel and J. Laski, "Algorithmic software fault localization", Annual Hawaii International Conference on System Sciences, pages 246–252, 1991.

[14] Z. Li, M. Harman, and R. Hierons, "Search Algorithms for Regression Test Case Prioritization," IEEE Trans. Software Eng., vol. 33, no. 4, pp. 225- 237, Apr. 2007.

[15] David Leon and Andy Podgurski, "A Comparison of Coverage- Based and Distribution-Based Techniques for Filtering and Prioritizing Test Cases", *Proc. Int'l Symp. Software Reliability Eng.*, pp. 442-453, 2003.

[16] Dennis Jeffrey and Neelam Gupta, "Test Case Prioritization Using Relevant Slices", In *Proceedings of the 30th Annual International Computer Software and Applications Conference*, Volume 01, 2006, pages 411-420, 2006.

[17] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Test Case Prioritization," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929-948, October, 2001.

[18] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Test Case Prioritization: An Empirical Study," International Conference on Software Maintenance, Oxford, UK, September 1999, pp. 179 - 188.

[19] Sayogita Chaturvedi, A. Kulothungan, Improving Fault Detection Capability Using Coverage Based Analysis", IOSR Journal of Computer Engineering (IOSR) – (JCE) e - ISSN: 2278 - 0661, p – ISSN: 2278 8727Volume 16, Issue 2, Ver. VI (Mar-Apr. 2014), PP 22-30. IEEE Press, Dec. 2007, pp. 57-64, doi:10.1109/SCIS.2007.357670.