Improving White Box Testing Using Bi – Directional Symbolic Analysis and Test Case Slicing

P. Velmurugan[1], S. Ganesh Kumar[2]

[1]Assistant Professor / CSE, SRM Institute of Science and Technology, Chennai, India
[2]Associate Professor/ CSE, SRM Institute of Science and Technology, Chennai, India

ABSTRACT
Adequacy in programming testing comes when we have an arrangement or deliberately approach in the testing area to cover high estimation of a source code of any product by testing all its code successfully. For high scope and high programming quality level we have to accomplish high branch scope that is 90% scope must be finished by an analyzer then no one but, we could ready to found that the testing has accomplished some picked up in adequacy in programming testing territory. Essentially, a few strategies and procedures had been looked into and connected, additionally got achievement already on testing however the outcome was scored low in fact. This high disadvantage in programming testing space raises request to propose and build up a different strategy which can help in discovering mistakes and testing each source code successfully.

Keywords: Programming Testing, Software Reliability, Software Effectiveness, Test case cutting

Correspondence:
P. Velmurugan
Assistant Professor / CSE, SRM Institute of Science and Technology, Chennai, India

INTRODUCTION

Throughout the most recent decades, the PC framework has gone into people life in each region from home to workplaces. Along these lines, as per the significance of PCs in human life has made an engineer intense on its execution, unwavering quality, and support of the PC framework. For the great quality programming, the framework ought to be powerful. To get the successful framework we require great working of the source program which is finished with the method called programming testing. Compelling testing is an objective to accomplish a decent nature of programming by utilizing different blends of information sources sends to the framework to discover infeasibility in the source code. Successful testing is a mix of high scope score measurements and superb level. Already numerous strategies had been executed in programming testing space however scope metric was low. The explanation for was either because of the nearness of branches, circles or complex program, or else because of absence of experiments. The primary focal point of this examination paper is to build branch scope and disregarding the issues applicable to some code components can be infeasible. Numerous techniques like emblematic and console execution, seek based methodologies and arbitrary experiment age had connected to expand the scope rate however this outcome in low branch scope. Viability in testing requires a connection between high scope score and excellent level. For high scope and high programming quality level we have to accomplish high branch scope that is (90%) of scope on source code zone required then just it will demonstrate the testing is viable in testing area. We proposed a new technique called Bi-directional symbolic analysis which helps to cover the program by using external inputs and also indicates the measurement score i.e. (80%) of coverage will be done that leads developers to analyzed testing in an effective way. The goal of the Bi-directional symbolic analysis is to traverse every statement of source code and find the area with its reachability condition which is not been cover under testing. The bi-directional symbolic analysis is a combination of forward analysis and backward analysis. The goal of forward analysis, which is in the form of symbolic execution to execute not-yet-covered elements and backward analysis which are in the form of symbolic reachability analysis is to identify reachability condition and reveal unreachabilitystates.

PROBLEM STATEMENTS

A. Existing problem

There is a problem of high coverage in source code especially the uncovered area, branch coverage and the presence of many infeasible ones that are not executable in a testing domain with these issues it impacting on the effectiveness of the coverage to obtain with current approaches to get an exact metric measurement. For example, in source code, some area like a loop with the following statement have not been executed by external input during testing and to make it executable some method have been applied but could not be able to reach.

B. Interested Problem

This paper tries to enhance test execution as takes after:

- ✓ Try to cover code scope metric by utilizing some characterized test cases.
- ✓ Try to put forth infeasible expression and a reveal zone to be cover with test cases.
- ✓ Try to cover incompletely executable explanation.
- ✓ Try to discover reachability condition to uncover inaccessible state

I. Overall Block diagram of a proposed technique

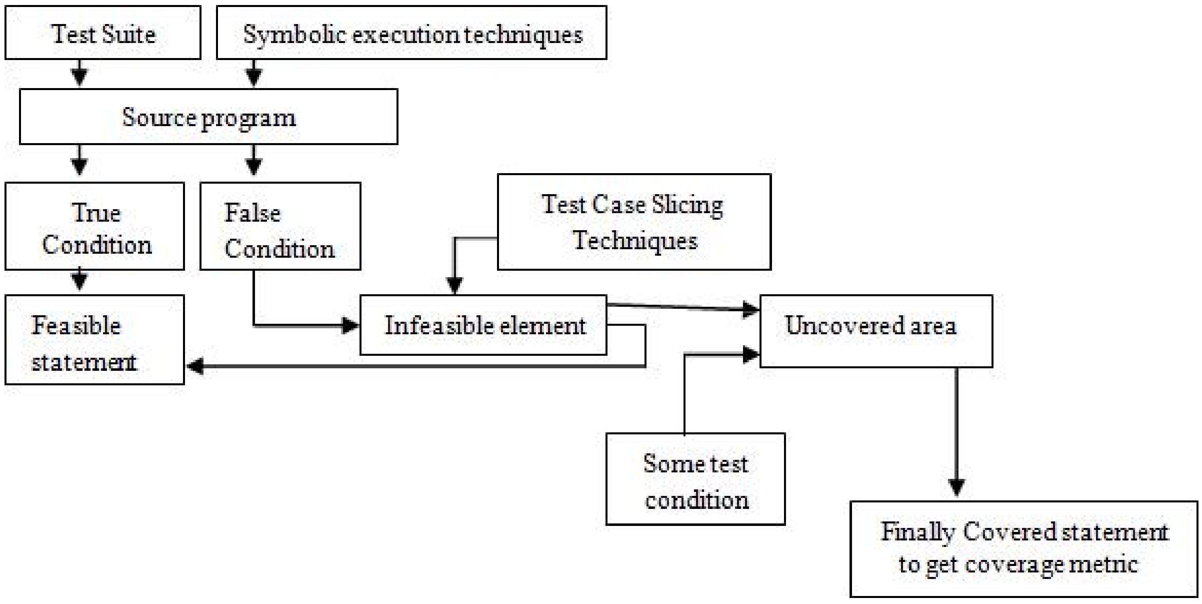


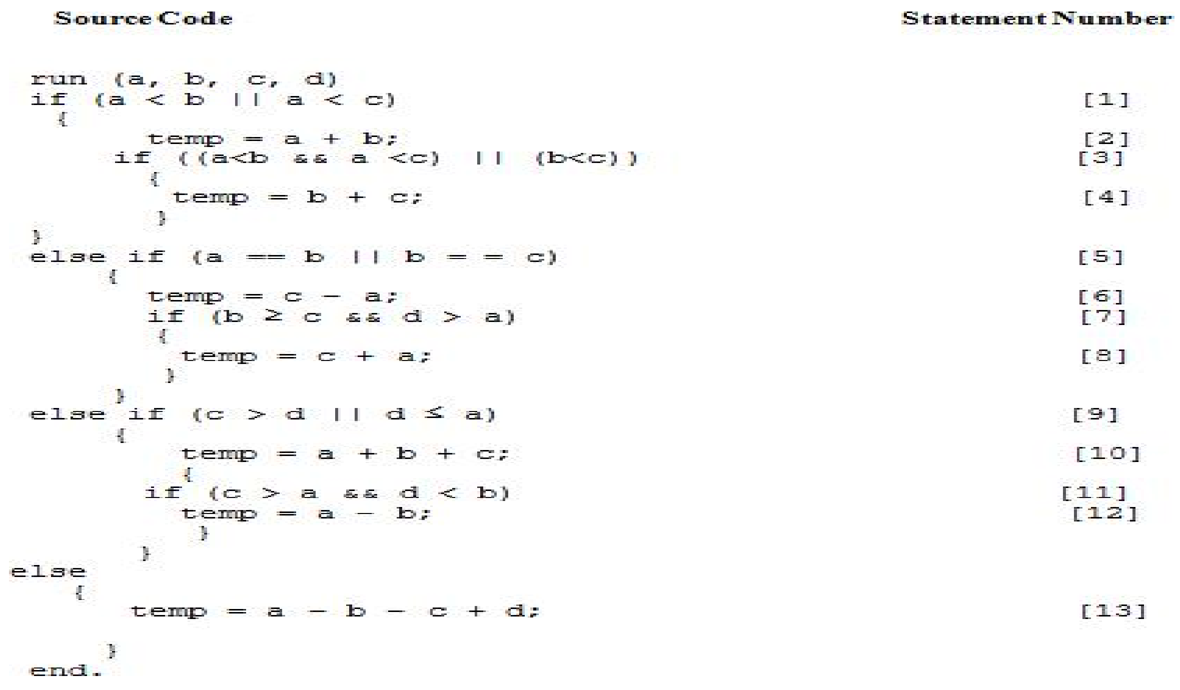
Fig 1. Proposed Module for Effective testing using Bi – directional symbolic analysis and Test case slicing

The principle point of compelling testing utilizing bi-directional emblematic investigation is to check each program proclamation whether the announcement is practical, infeasible, and revealed region of a source code. Thus, that the scope metric score or estimation can be accomplished. Bi-directional representative investigation is a blend of emblematic execution as emblematic examination to cover each component in a source code whereas emblematic reachability is to locate the reachable condition to uncover inaccessible states.

Figure 1 portrays the general outline chart of powerful testing. It comprises of two noteworthy stages: representative execution, experiment cutting. Crafted by representative execution is to navigate each program articulation and furthermore crossed infeasible components whereas crafted by emblematic reachability are to locate the correct condition or reachability condition to uncover inaccessible region in a program.

A. Source Code

Fig 2. Source Code



```
Source Code                                              Statement Number

run (a, b, c, d)
if (a < b || a < c)                                          [1]
  {
        temp = a + b;                                        [2]
     if ((a<b && a <c) || (b<c))                             [3]
        {
         temp = b + c;                                       [4]
        }
}
else if (a == b || b = = c)                                  [5]
    {
        temp = c - a;                                        [6]
        if (b ≥ c && d > a)                                  [7]
         {
          temp = c + a;                                      [8]
         }
    }
else if (c > d || d ≤ a)                                     [9]
    {
          temp = a + b + c;                                  [10]
          {
        if (c > a && d < b)                                  [11]
          temp = a - b;                                      [12]
           }
       }
else
     {
       temp = a - b - c + d;                                 [13]

    }
end.
```

In the figure 2 source code we do testing by giving ten diverse experiments each having four information variable like Test case T1 [2 3 4 1] where variable a = 2, b =3, c = 4 and d = 1. We have mark proclamation number on each source code to show the announcement number with the goal that it can help in distinguishing the specific source code which is executable by specific experiment. In the wake of checking with all the ten diverse experiments on a source program we can be ready to see the attainable part meant with "E" and incomplete executable proclamation "P" and the revealed explanation meant with "N". Following figure demonstrate executable proclamation, fractional articulation and non-executable explanation by specific experiments.

Table 1: Statement Coverage by Particular Test Cases.

[a b c d]	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13
T1[2 3 4 1]	E	E	E	E	N	N	N	N	N	N	N	N	N
T2[8 4 15 17]	P	E	P	E	N	N	N	N	N	N	N	N	N
T3[1 17 3 2]	E	E	P	N	N	N	N	N	N	N	N	N	N
T4[11 10 2 1]	N	N	N	N	N	N	N	N	E	E	P	N	N
T5[5 5 4 11]	N	N	N	N	P	E	E	E	N	N	N	N	N
T9[119 2021 12]	N	N	N	N	N	N	N	N	E	E	P	N	N

From table 1, we have selected few test cases on the basis on highest ration in executable statement and the selected test cases are T1, T4 & T5. We can find that test case T1 the beginning tree statement are fully executable whereas in test case T5 and T4 has fully executable statement are statement 6,7,8,9 & 10 and partial executable statement are 5& 11 respectively. With test cases T2, T3 and T9 have some statement is infeasible and uncovered statement. This can be done by the techniques called Test Case slicing method.

B. Control Flow Graph

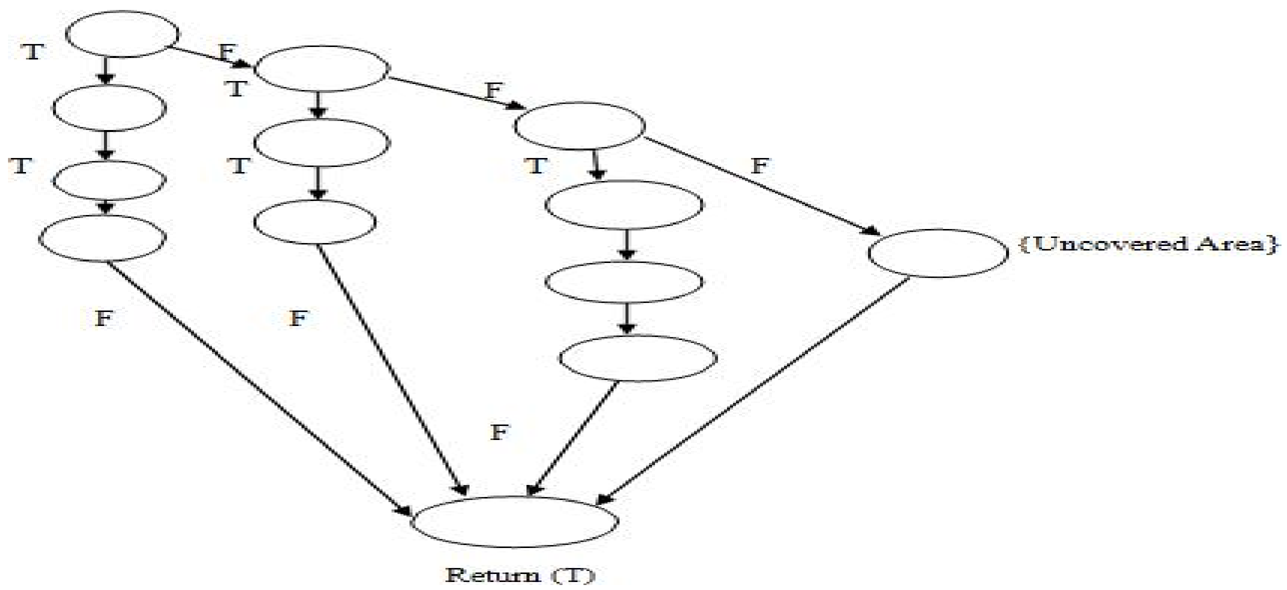


Fig 3. Control Flow graph of Statement

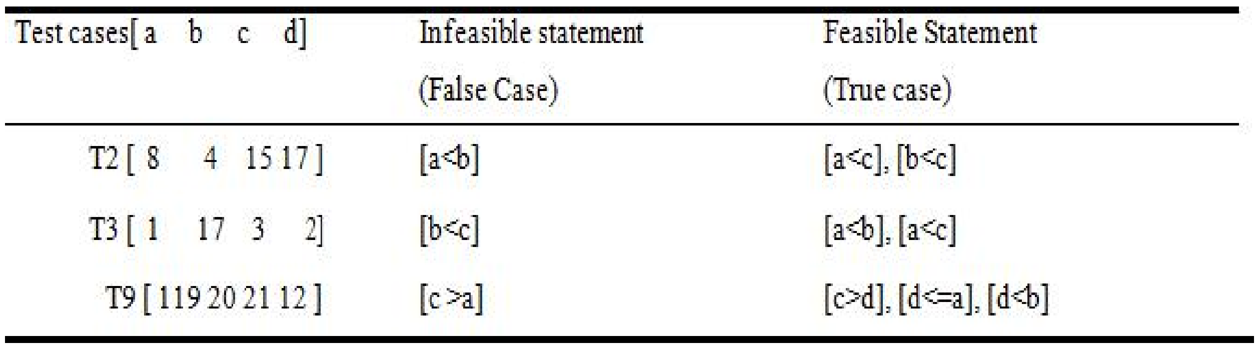
SYMBOLIC EXECUTIONS

The objective of representative execution is to navigate every last explanation of a source program and identify the attainable proclamation and an infeasible articulation and a reveal zone in a source program. In emblematic execution each code of a c/c++ program is check by given outer info variable and recognize which input variable a specific articulation is run, and which proclamation is demonstrating blunder. Like in current venture, there are add up to thirteen articulation in a program and each announcement is confirm by utilizing ten test suite and each test suite have four information variable a, b, c and d. With some test suite some announcement executes, and some territory gets revealed and some are infeasible.

TEST CASE SLICING TECHNIQUES

In experiment cutting procedure, the entire program can be partitioned into two classes: Feasible proclamation and infeasible explanation or revealed zone in a source code. Given coming up short single declaration experiment coming about because of emblematic execution, we cut these experiments by expelling unimportant articulation from testing area or methodology. The primary objective in experiment cutting procedures is to discover the achieve capacity condition to cover infeasible proclamation and uncover inaccessible state in a program.

Table 2. Infeasible and Feasible Statements in a Source Code

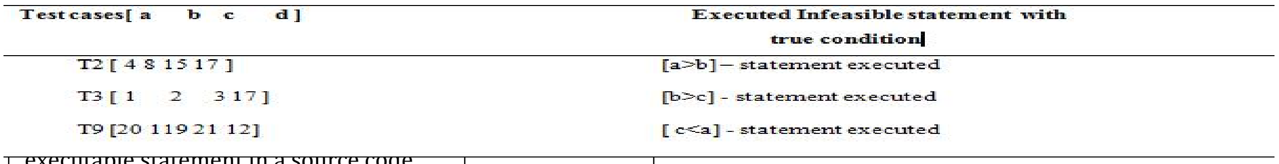


Test cases[a b c d]	Infeasible statement (False Case)	Feasible Statement (True case)
T2 [8 4 15 17]	[a<b]	[a<c], [b<c]
T3 [1 17 3 2]	[b<c]	[a<b], [a<c]
T9 [119 20 21 12]	[c >a]	[c>d], [d<=a], [d<b]

From table 2, we assure that the condition are false for infeasible statement and with the test case slicing method we need to find the reachability condition to execute infeasible statement and we can do by interchanging the input variable to find the reachability condition.

Table 3. Resultant Table after Test case slicing

II. REACH COVERAGE



Test cases[a b c d]	Executed Infeasible statement with true condition
T2 [4 8 15 17]	[a>b] – statement executed
T3 [1 2 3 17]	[b>c] - statement executed
T9 [20 119 21 12]	[c<a] - statement executed

executable statement in a source code

Test Case T10 we need to find the reachability condition or reachability inputs value to reach uncover statement ci.e. statement number [13] and for that we need to find the maximum and minimum value of given four input variable on the basis of given Conditional Statement in a source code to reach statement number

[13] Conditional Statement in source code is: Statement number [1], [3], [5], [7], [9], & [11]

(i) To discover greatest and least factor

To Reach Statement Number [13] - {(a>b), (a > c)} {(a≠b), (b≠c)} {(c<d), (d>a)}

T7 = [10, 3, 9, 15] will reach to explanation number [13] where all the contingent proclamation with this specific information variable of a, b, c and d will encourages us in covering articulation number [13]. With these new proposed strategies in testing space, bi- directional representative examination covers or achieves the scope metric score that is 90% scope has come to in this venture. Bi-directional serves to identify the possible component.

B. ALGORITHM

Inputs: inputs variable a, b, c & d of test suite Ti
Output: Executive uncovered area in a source program P.

Step1: Inputs variable (a, b, c & d) of test cases Ti in a source program “P” to find the feasible statement and infeasible statement.
Step 2: P – Total statement in a program (S1-S13 statement).
Step 3: Ti (a, b, c & d) to execute step by step (S1-S13) to find feasible and infeasible part in a program P.
Step 4: For P – (S1- S13) if execution statement is TRUE, return ‘PASSED’ as status of test case Tp.
Step 5: For P – (S1-S13) if not executed statement is False, return ‘Failed ‘as status of test case Tp.
Step 6: Shows the percentage of test cases with status ‘PASSED’ with respect to Ti number of test cases.
Step 7: Find reach ability condition by interchanging inputs variable to execute uncovered area in a source program P.

Result Analysis

In the wake of assessing with ten distinct instances of information variable of "a", “b“,“c" and "d” in a program we would ready to locate the possible proclamation and infeasible articulation of a program code. Experiment Slicing methods finds the condition or achieve capacity state to cover those infeasible part by trading inputs estimation of false condition to influence it in consistent with cover to the infeasible part without affecting the condition which are genuine at first for doable proclamation. At whatever point test cases has executable articulation it return experiment "Go" as status, else it will return "Bombed" as status, and in light of the quantity of experiments that arrival "Go" as status there is increment in level of experiments which are effectively executed in a source code. The Graph appears in figure 4, portrays the outcome investigation of this examination paper.

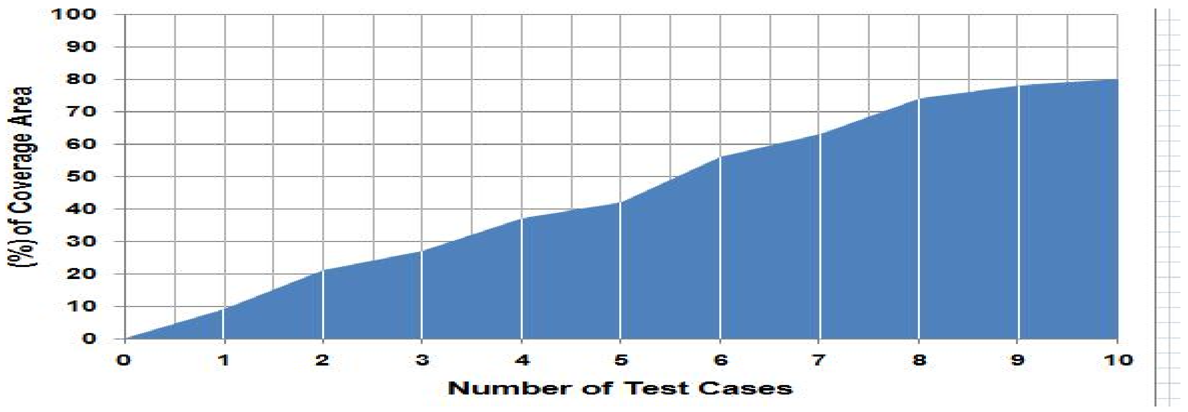


Fig 4. Result Analysis graph showing increase in percentage of test cases successfully executed based on number of test cases.

CONCLUSION & FUTURE WORK

Subsequent to assessing with ten distinct instances of info variable of a,b, c and d in a program we would ready to locate the achievable explanation and infeasible proclamation of a program code. Experiment Slicing systems finds the condition or achieve capacity state to cover those infeasible part by exchanging inputs estimation of false condition to influence it in consistent with cover to the infeasible part without affecting the condition which are genuine at first, for plausible proclamation. Till now we have broken down each announcement of a code by giving diverse ten cases and found the doable and infeasible piece of a program code and connected "Experiment cutting "strategies to achieve infeasible proclamation. Subsequent to applying Test Case Slicing on program code to cover the revealed territory by specific experiments and after that we discover most extreme most noteworthy incentive between input variable a, b, c, and d to encourages us to achieve the condition to cover articulation "S13" which are not executable in all the ten experiments. Likewise we would ready to state that the quantity of experiments executed the announcement is TRUE return PASSED as status generally will return FAILED as status, and there is increment in level of experiments returning positive outcome as for add up to number of experiments and the same has been appeared in Figure 2. As a future work, we will be further analyzing a similar positive work of our undertaking on other dialect venture like java venture, information structure venture. We will likewise attempt to check for similarity of a similar technique for our undertaking on other dialect venture utilizing successful testing strategy with various situation, where in we can spare the cost, time and ability in more expand.

REFERENCES

1. Sayogita Chaturvedi and Kulothungan, A., 2014. Improving fault detection capability using coverage-based analysis. *IOSR Journal of Computer Engineering*, *2*,pp.22–30.
2. Duggal, G. and Suri, B., Understanding Regression Testing Techniques. 2008
3. Leon, D. and Podgurski, A., 2003. A Comparison of Covergae- Based and Distributed-Based Techniques for Filtering and Prioritizing Test Cases. *Proc. Int'l Symp. Software Reliability Eng.*, pp.442–453.
4. Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. J. Mol. Biol. 147, 195--197 (1981)
5. May, P., Ehrlich, H.C., Steinke, T.: ZIB Structure Prediction
6. Pipeline: Composing a Complex Biological Workflow through Web Services. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006.LNCS, vol. 4128, pp. 1148--1158. Springer, Heidelberg (2006)
7. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
8. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid Information Services for Distributed Resource Sharing. In: 10th IEEE International Symposium on High Performance Distributed Computing, pp. 181--184. IEEE Press, New York (2001)
9. P.Velmurugan and Pradnya Goel: Test Data Generation for Improving the Effectiveness of Test Case Selection Algorithm Using Test Case Slicing, Journal of Computational and Theoretical Nanoscience, Vol.16, Issue No:5/6, Pg.No:1848-1853, 2019.
10. Sumon Chatterjee, M.Mohan: Test Case Prioritization for Regression Testing Based on Fault Dependency and Fault Severity, International Journal of Engineering Studies and Technical Approach, Volume 2, Issue No:4, Pg.No:199-206 April 2016.
11. Anushree and M.Mohan: Efficient automatic test case generation for data flow testing using Neighborhood crossover method in genetic algorithm, Asian Journal of Science and Technology, Volume 6, Issue No:4, Pg.No:1354-1658.