REGRESSION TESTING : TABU SEARCH TECHNIQUE FOR CODE COVERAGE

T.Prem Jacob [1]
[1] Research Scholar, Sathyabama University, Chennai-119 , India
premjac@yahoo.com

Dr.T.Ravi [2]
[2] Principal, Srinivasa Institute of Engineering & Technology, Chennai – 56 , India
travi675@yahoo.com

Abstract

Software testing is one of the most expensive and critical activities which carries out every time in order to give a best quality of a software product. Here the regression testing which is based on testing mechanism is used to analyze the source code changes and also to make sure that the changes that does not establish new bugs in the earlier validated codes. Now a days many innovative methods are raised in performing the software testing, among them the unit testing which uses minimum time frame and gives more effort in performing a task. Under lots of schedule the unit testing mechanism is done by more developers as the software companies has an enough time to find cooperation among different operations like functionality, quality and time to market. There is an essential to reduce the unit testing time by making it as an automated one and also by making its process as more optimistic.Here, this paper propose a technique called Tabu search based technique for an effective code coverage to cyclomatic complexity which is used to measure the complexity of a program.

Keywords- Software testing; cyclomatic complexity; tabu search; unit testing.

1. Introduction

There is a well-known discussion stating that “Under Testing is a crime and over testing is a Sin”. The exhaustive testing is the major challenges in testing where these testing are not possible, when to end testing could not be evaluated and there is no other way to display the absence of problems. The software testing is developed with the fabulous proliferation of software testing methodologies, increased pace of production schedules, programming languages and increased volume of software applications from a routine quality guarantee activity into a complex challenge and a sizable in terms of effectiveness and manageability.

In today’s business environment the major challenges to software testing are,

1.1.Thoroughness

How you can tell logically the program is error-free? How can you say when you are performing testing?

1.2.Efficiency

How can you make sure that each test is a fine investment of money and time? Whether the test cycle is too long

1.3.Resource Management

Are the functionally middle parts of the program receiving a suitable level of testing? Are testing resources strategically assigned, pointing on the highest threat fundamentals of the software? The unit level testing differs from the ad hoc tests which is completed by the developers as they are developing code to systematic white box difficult (testing), where the Unit level testing is a branch of each unit where each unit should be tested and recognized by QA and also by the Test group. In any case, the tester starts with the aim of coverage because it is the important reason of unit level testing [1] to reaches the maximum level of coverage feasible. Unit testing is more essential as it achieves soon in the development method and it is very high cost-effective at tracing the errors. The main seek of unit level testing is to find the minimum factors of unit level tests to execute. In a model world, each probable way of a program is tested and accounting for every performable choices in every possible combinations. But it is not possible when one think about the huge number of probable paths connected in any provided program. The aim is to separate a subset of paths that gives reporting for every testable unit, and to make that sub-factor as low and no cost of unit level redundancies as probable. The definition of Myers aptly about software testing is “an operation of executing a program with the intension of identifying the errors”. Using the medical diagnosis analogy, a fine investigation is one that make search and finds an issue, instead of one that discloses nothing and gives a fake sense of welfare. According to this definition a fine group of test cases contains very high possibilities of finding the previous unknown errors as a victorious test executes is one

which establish these types of errors. With the help of exhaustive testing, the whole possible error is detected in the program. Also the exhaustive testing is required to examine every possible logical execution paths and input. If not impossible apart from the unimportant programs then this is cheaply unachievable. So, a real aim of software testing is to maximize the possible of detecting errors using a set of test cases, done in less time and with less effort. A huge number of testing functions developed to help the tester with the choice of appropriate test data and have been created over the final decades because of middle importance of test case design for testing.

The Previous test case design functions are distinguished into white-box tests and black-box tests. Black-box test cases are obtained from the requirement of the program under test, while white-box test cases are inherited from inner structure of the software. If a formal order exists, the automation of the black-box testing is meaningfully probable. Tools accepting white-box tests are partial to program code instrumentation and reporting measurement, because of partials of symbolic invocation. In two sections full automation of test case design test case is very difficult [4, 9].The test case design is depends upon the tester. Test case design is normally done manually. The manual test case design conversely is susceptible to errors and time intensive. The accuracy of test if mostly depends on the single tester performance

2.Related Work

This section takes survey on earlier work in different algorithms and techniques. These algorithms and techniques have been used in journals to compact with the test case prioritization difficulties. For System level prioritization the test case prioritization algorithm [7] regards two features. First feature is a test case detecting average number of faults per minute. The second factor is the error crash. That is testing efficiency could be increased by focusing on the test case. That is possible to have very huge number of harsh error. Therefore for every fault harshness value was allocated based on their collision of the error on the creation [8] presents a bee colony optimization algorithm. That implies natural workers food forging activities bees as an algorithm. That is for prioritize test cases regression test suite depend on huge error coverage. [9] Hybrid Particle Swarm Optimization is an artificial intelligence based technique. It is a group of particle Genetic Algorithms and Swarm Optimization technique. The resident's plays main role in choosing which path the result will approach. The collections of particles creating population or the population contains of particles. From a provided issue, velocities and position are allocated on the basis of true particle Swarm Optimization technique the population are randomly selected. In the test case prioritization operation, it considers particle velocity as total implementation time taken by it in covering the issue and particle position because numeral of faults cover by particle. The finalizing criterion is to be choosing and on the basis of this Hybrid Particle Swarm Optimization algorithm conclude. The genetic algorithm used for prioritize the regression test suite. That genetic algorithm will prioritize the test cases based on the code coverage presented in [10].The conditions are found in every separate path this is used in the genetic algorithm fitness function will be the conditions of coverage. The technique available in [11] initialized a regression test suite prioritization algorithm. That technique prioritizes test cases with the aim of increasing the number of issues that are to be obtained while the inhibited execution.

The algorithm finds average issues which are calculated by per minute. The results are showed to find success of prioritized and un-prioritized case that is obtained with the help of APED metric.[12] which initiates a regression test suite prioritization algorithm. It authenticates necessary support system level test case prioritization method to improve customer-perceived software quality and to produce more severe issues at prior stage by using genetic algorithm. The Genetic algorithm used for prioritize the test suite with the desire of increasing the number of founded faults. With the help of test adequacy criteria the ordering was obtained. This find how possible errors are obtained instead of how many issues the test finds in other sense this function is used to notify how many faults to be present in the code. These been used to notify the entire possible test case ordering fitness[12].

Projected a collection of prioritization factors to design the planned system. It submit to these factor consider as prioritization sets. These factors which are customers allotted priority of necessaries, completeness, Traceability, invoke time based on these sets the weights are described for every test case in the application. According to the prioritization the evaluation time and cost could be degraded by focusing on exacting test cases. In [13] the approaches utilize the requirement prioritization concepts.

This prioritization of necessaries could be performed by the developer, by the tester or also depend on the necessaries specification document or considering how hard or easy to implement that necessaries. Then, invokes the test cases of the specific test suit and find issues. The object is to invoke highly prioritized necessaries. So, the faults will be most harsh faults. This problem is solved using genetic algorithm by covering the tuples covering maximum number of necessaries. This genetic approach compared to fault as well as random based approach, the genetic algorithm finds many severe faults shortly in the testing process.

3.Cyclomatic Complexity Measurement

Cyclomatic complexity is a software structural metric, which is used to measure the program complexity. Usually program complexity is measured using Control flow graph. The cyclomatic complexity of a planned program is shown as

$$C=nE-nN+2D$$

Here C is the cyclomatic complexity, nE be the number of edges of the graph, nN be the number of nodes of the graph and D- be the number of disconnected components. It gives number of lower bound on the numeral of test cases essential to achieve the branch coverage. Cyclomatic complexity calculates the quantity of test effort. Coverage can be attaining with fewer number of test cases while here are smaller quantity test cases that can be measured.

4.System Architecture

In this paper, for test suite generations we use a tool named as ST tool which takes control over the flow graph as input and create the test cases as for the input of different variables by means of tabu search technique. The architecture of code coverage using tabu search is shown in figure 1. Also the Control Flow Graph Generator receives the program source code as input by using tabu search technique where it generates test case from the input of different variables.

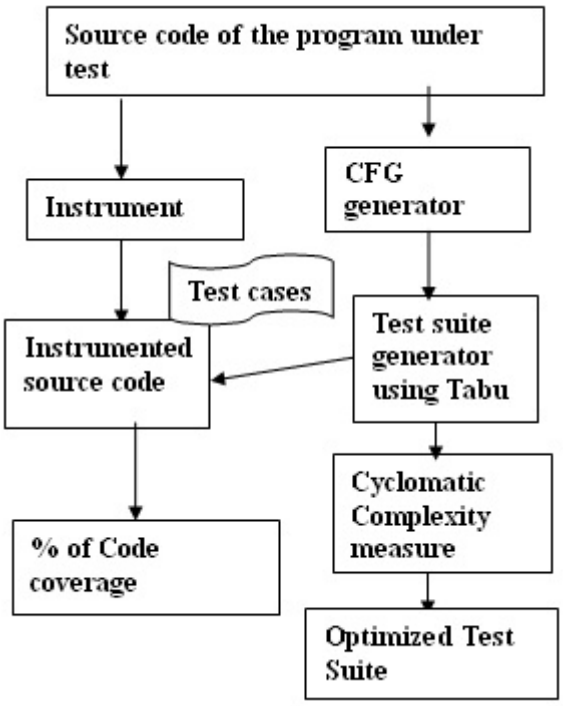

Fig. 1- Architecture of code coverage using tabu search

In order to obtain the code coverage for finding its best set of optimal test cases for regression testing the various steps are followed to create the test cases. The first step is the source code of the program is taken as input to the Control Flow Graph generaor.

The different steps in the automated test case are as follow:

1.Control Flow Graph generator which is used to generate the control flow graph by taking the source code of the particular application as input.

2. Amount of test effort is calculated in the second step by using Cyclomatic Complexity measure.

3. Control Flow Graph is analysed and the branching condition of the source code information is extracted.

4. With the help of tabu search technique test cases are generated for every condition in the source code from the input field of the variables occupied in the branch condition.

5. With the use of Cyclomatic Complexity measure has to find fulfillment of numeral of test cases

6. In order to check the branch coverage the generated test cases are practical to the instrumented source program

7. Finally by finding the suitable test cases from a test suite for the given particular application source code under test.

5.Code Coverage : TABU Technique

The tabu search technique is a metaheuristic technique. This is established successfully in real world applications such as traveling salesman problem. Newly it is obtained appropriate for test case generation issues in software testing. In that only few outcomes have been distributed with comparatively few samples. Also it is established with many samples and all input domain data types.

The tabu search algorithm is gives as,

```
CUS=value;
 C=CUS;
TLS.add(CUS);
S=A, B, C,
DO
FIND ADC
For (int i=0;i<=ADC.length;i++)
{
If (CAN<Ca)
{
C=Ca;
}
}
if(sgn!=S)
{
TLs.add (CUS);
}
else
{
delete (TLS);
}
Boolean b=true;
if (CUS=b)
{
TlT.add (CUS);
TlT.bt ();
}
While (sgn! =s && iterations. Length<MAXIT)
Break
```

Parameters:

CUS-current solution;
C-CFG;
TLS-tabu list software testing;
ADC-adjacent candidates;
CAN-candidates value in node n;
Ca-CFG in node n
A, B, C-covered sub goal node
Sgn-subgoal node
TlT-tabu list LT;

Pseudocode:

Start

Step1: Current Solution should be initialized.

Step2: Save the Current Solution in Control Flowgraph Genetor.

Step3: In Tabu list add current value.

Step4: Choose a sub goal node which have to be covered

Step5: Has to Do calculation with the neighborhood candidates.

Step6: Each and every candidates repeat the step 5

Step7: If (candidate value in node n <CFG in node n) then

Step8: Again Save the candidate in CFG

Step9: end if

Step10: end for

Step11: if the sub goal node is not covered, then have to do following operations

Step12: Again Add Current Solution to tabu list ST

Step13:else Delete tabu list ST

Step14:end if

Step15:Choose a sub goal node whih has to be covered and

Current solution

Step16:if Current Solution is depleted, then

Step17: Now,Add Current Solution to tabu list LT after completing this process

Step18: Now,Apply a backtracking process:

Step19: The New Current Solution and maybe new sub goal node

Step20:end if

Step21:while (NOT all nodes covered AND number

of iterations<MAXIT)

Step22:end

A brief explanation about the tabu search algorithm, initially initialize the current solution (CUS) and store in CFG. Then add the current solution to tabu list software testing .select the goal node to be cover up and calculate the zone candidates for every candidate. If the candidate value in node n is less than the CFG in node n then store the candidate in CFG. If cub goal node not in covered then add current solution to tabu list software testing. Otherwise destroy the tabu list software testing .If current solution is used up then add the current solution to tabu list LT and apply backtracking process: new current solution and may be sub goal node. check if all nodes are not covered and number of iterations is less than MAXIT means end the process.

6.Results

In the test suite generation the triangle classifier program is commonly used in many research papers [1, 3, and 13] as well as with the 20 benchmark application the proposed technique is checked. The outputs founded are hopeful and tabu search techniques has done a better operation than the random technique. The operation is measured for differet instance ratios for a code coverage and test suite size measurement is done for the similarity measurement of the techniques. Under the test, the test suite range is evaluated with the cyclomatic complexity program structure. The comparison provides to determine the number of test cases that are needed to wrap the program.

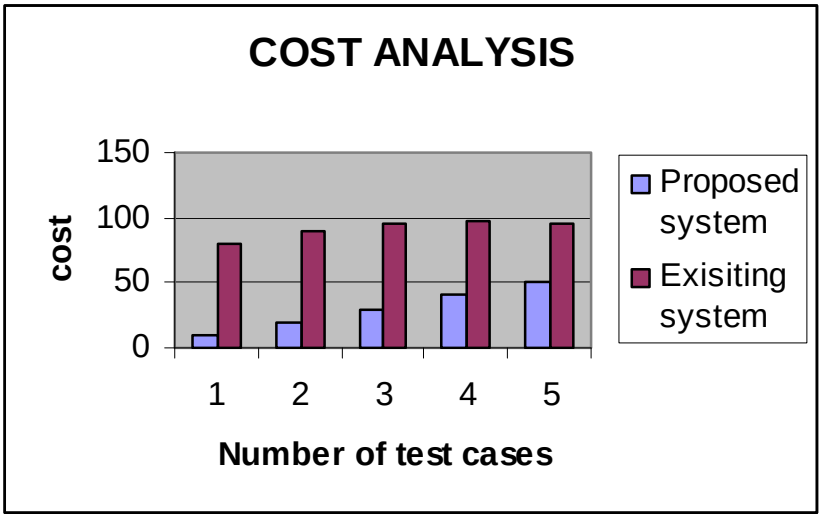

Fig 6.1- Cost Analysis

In exsisting system, cost for testing the software is very high when compared to the proposed system as shown in figure 6.1. In case of exsisting system, when the number of test cases increases the time to execute the test case will also be increased and also cost gets increased,but while using tabu search technique in the proposed system, test case execution time and cost is reduced to test the software.

Fig. 6.2- Performance between code coverage and branch coverage

Figure 6.2 explains about the overall comparision between the Code coverage and Branch coverages using Tabu Search Technique. The Branch coverage performance is very less than the code coverage when using the Tabu Search mechanism.

Test Suite Samples	Size of the Test Suite	Existing Code Coverage	Proposed Code Coverage in %	Values of Cyclomatic Complexity
SA1	3	78	99.9	3
SA2	2	80	98.9	2
SA3	4	74	98.9	4
SA4	4	82	98	4
SA5	3	85	99	3
SA6	2	82	100	2
SA7	3	87	99.5	3
SA8	3	85	98	3
SA9	4	79	97.9	4
SA10	2	75	99	2

Table 1- Results of the Existing and Proposed(TABU SEARCH) Technique

The table1 shows the result performance of the existing technique with the proposed tabu search technique. Thus the result describes about the overall code coverage varies from from 98% to a maximum of 100% and is been achieved by applying with more number of test suites and than which finally calculates the Cyclomatic Complexity measured value.

Thus the proposed approach is very suitable for the code coverage and it provides a higher accuracy rate compared with the existing algorithm.

7.Conclusion

In this paper a proposed technique is used in order to increase the test code coverage which is also in more than one coverage item. Also as an improvemet the test suite prioritization for code coverage is done by the Tabu search Searching mechanism which is also doe for the structural software testing. By using the Tabu search mechanism Testers and Developers can perform the testing with minimum resource and time frame. Secondly with the help of regression testing more number of test cases are produced and also the code coverage is obtained by using tabu search which uses a multi coverage criteria. Thus the effectiveness of code coverage is identified mosly with the fault detection efficiency.

References

[1] Adam M. Smith and Gregory M. Kapfhammer. An empirical study of incorporating cost into test suite reduction and prioritization. In Proceedings of the 24th Symposium on Applied Computing, 2009.

[2] GregoryM. Kapfhammer. A Comprehensive Framework for Testing Database-Centric Applications. PhD thesis, University of Pittsburgh, Pittsburgh, Pennsylvania, 2007.

[3] Hao Zhong, Lu Zhang, and Hong Mei. An experimental study of four typical test suite reduction techniques. Information and Software Technology, 50(6), 2008.

[4] Sreedevi Sampath, Renee C. Bryce, Gokulanand Viswanath, Vani Kandimalla, and A. Gunes Koru. Prioritizing user-session-based test cases for web applications testing. In Proceedings of the 2nd International Conference on Software Testing, Verification, and Validation, 2008.

[5] Boris Beizer (2000) 'Software Testing Techniques',2ndedition, Dreamtech publisher, New Delhi.

[6] Antoniol, G., Penta, M.D., and Harman, M. (2005), Search-Based Techniques Applied to Optimization of Project Planning for a Massive Maintenance Project, Proc. 21st IEEE Int'l Conf. Software Maintenance

[7] Deb, K., Agrawal, S., Pratab, A., and Meyarivan, T. (2000), A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II, Proceedings of the Parallel Problem Solving from Nature VI Conference

[8] Elbaum, S., Rothermel, G., Kanduri, S., and Malishevsky, A. (2004), Selecting a Cost-Effective Test Case Prioritization Technique, Software Quality Control, vol. 12, no. 3, pp. 185-210

[9] Glover, F., and Kochenberger, G., Handbook of Metaheuristics, Springer, 1st edition, 2003

[10] Harman, M. (2007), The Current State and Future of Search Based Software Engineering, International Conference on Software Engineering - Future of Software Engineering

[11] Scott McMaster and Atif Memon. Call stack coverage for GUI test-suite reduction. In Proceedings of the 17th International Symposium on Software Reliability Engineering, 2006.

[12] Zheng Li, Mark Harman, and Robert M. Hierons. Search algorithms for regression test case prioritization. IEEE Transactions on Software Engineering, 33(4), 2007.

[13] Sebastian Elbaum, Alexey G. Malishevsky, and Gregg Rothermel. Test case prioritization: A family of empirical studies. IEEE Transactions on Software Engineering, 28(2), 2002.

[14] David S. Rosenblum and Elaine J. Weyuker. Using coverage information to predict the costeffectiveness of regression testing strategies. IEEE Transactions on Software Engineering, 23(3),1997.

[15] Kristen R. Walcott, Mary Lou Soffa, Gregory M. Kapfhammer, and Robert S. Roos. Time-aware test suite prioritization. In Proceedings of the International Symposium on Software Testing and Analysis, 2006.
[16] Christian Murphy, Kuang Shen, and Gail Kaiser. Automatic system testing of programs without test oracles. In Proceedings of the International Symposium on Software Testing and Analysis, 2009.
[17] R.Aroul canessane, Dr.S.Srinivasan, Software Architecture modeling framework using UML, IJCSE, Vol. 4 No.2 Apr-May 2013.
[18] JeffreyM. Voas. PIE: a dynamic failure-based technique. IEEE Transactions on Software Engineering, 18(8):717–735, 1992.
[19] Mr.T.Prem Jacob, Dr.T.Ravi, Detecting of Software Source Code Defects using Test Case Prioritization Rules,2nd International Conference on Latest Computational Technologies (ICLCT'2013) June 2013 London (UK).
[20] Monica Hutchins, Herb Foster, Tarak Goradia, and Thomas Ostrand. Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria. In Proceedings of the 16th International Conference on Software Engineering, 1994.

Authors Profile

T.PREM JACOB received the B.E degree in Computer Science and Engineering from C.S.I Institute of Technology, Manonmaniam Sundaranar University, Nagercoil, India in 2004 and M.E degree in Computer Science and Engineering from Sathyabama University, Chennai, India in 2006, where he is currently working towards the Ph.D. degree in Computer Science and Engineering at Sathyabama University, Chennai, India. He is an Assistant Professor of Computer Science and Engineering in Sathyabama University and he has more than 7 Years of Teaching Experience. He has participated and presented many Research Papers in International and National Conferences. His area of interests includes Software Engineering, Data mining and Data warehouse.

Dr. T. Ravi**,** Principal of Srinivasa college of Engineering & Technology, Chennai. He has graduated in computer science and Engineering from Madurai Kamaraj University, Masters and Ph.D in computer Science and Engineering from Jadavpur University, Kolkata. He has more than 20 years of teaching experience in various engineering institutions in Tamil Nadu. More than 25 research papers are published in International & National Journals and conferences and also 5 text books are published through various publications. He is the Recognised Research Supervisor in Anna University and Sathyabama University Chennai and MS university, Tirunelveli.