Multi-objective Test Suite Prioritization Using Concolic Testing

Gayatri Nayak, Mitrabinda Ray

Abstract*: Test suite prioritization is a testing procedure where test cases of individual test suite are ordered to maximize fault detection potential. Test Prioritization is an effective method, because it helps to cover more number of faults in executing less number of test cases. During our literature survey, we found many test case prioritization approaches, considering either branch coverage or path coverage. But, we feel it is not a complete coverage testing adequacy method. Because, there is no guaranty that a hundred percent branch coverage will lead to cover all paths of a program under test. Thus, we present a multi-objective test suite prioritization approach to prioritize the test suites of a system using concolic testing to achieve testing adequacy. Here, we have considered branch coverage and path coverage as multiple objectives for measuring strength of a test suite and ranking them. The sole reason behind choosing concolic testing method is that it always gives higher code coverage score with minimal number of test inputs. Because it is a hybrid testing method that uses concrete and symbolic execution with in-built mathematical model named constraint solver. Therefore, in this work, we have used concolic testing to test Java programs and obtain results like test suites, branch coverage percentage, and path coverage percentage. Then, these results are used by Test Suite Weight Evaluator Algorithm for prioritizing the test suites. This prioritization method is targeting to achieve high branch coverage, high path coverage. We have validated our proposed approach by executing twenty standard Java programs.

Index Terms*: Test Case Prioritization, Path coverage, Branch Coverage,Concolic Testing.

I. INTRODUCTION

Test case prioritization has been playing an important role to embellish theperformance of regression testing [5].In this method test cases are ordered in a testsuite. At themaintenance phase of Software Development Life Cycle (SDLC)different methods are used to prioritize test cases.Generally, the test case prioritization means in a test suite TS; all permutationsfor ordering the tests suites. Here, permutation can be defined as$permutation_i \in P$. Consider that each individual $permutation_i$ consists of nnumber of tests. After that, one function is defined to calculate the significance ofeach test suite. Then, test suites are reordered for prioritization purpose [4, 7, 9].To continue test suite competence, different test cases are combined to the existingtest suite, that leads to expansion of the test suite size and increase inregression testing cost. Therefore, there is a precise need of test case prioritizationfor decreasing regression testing cost [7].

A. Motivation

Prioritization method is used to improve the test adequacy criterion by reorderingthe test cases. Till now most of the researchers have worked in single objectiveoptimization problem like fault detection capability, coverage criteria etc. and not fulfilled the objective of industries[1, 3]. Due to frequent changes in requirementsof software products, it is getting complicated to develop the product inless time and cost. Therefore, most software industries targeting multi-objectivetest case prioritization. In this work, branch coverage, path coverage criteriaof a test suite are used as two objectives for test suite prioritization. Most ofthe researchers consider statement coverage/branch coverage/function coverageas a coverage criterion for test case optimization. We found that, there is very few work on test suite prioritization using multi-objective testing criteria. Thus, we have considered these two criteria as main factors for prioritization in our proposed approach.

B. Objectives

The proposed approach has the objective to prioritize test suites that shouldsatisfy complete testing adequacy and to reduce regression testing cost. In thisapproach, the stability of test suite is represented by branch coverage and pathcoverage criteria of each test suite [6,9]. In this paper, a novel methodology isused to prioritize test suites for a software system using path coverage, branchcoverage criteria [2]. To generate test suites from some Java programs Concolictesting is used. It also finds execution time, percentage of branch coverage andpath coverage. After obtaining these things for each test suite, a novel algorithm(Test Suite prioritization) is proposed based on two coverage metrics to prioritizethe generated test suites within a software project. The real contributionof our work is to achieve high path coverage and high branch coverage. Thus,we improve the regression testing process.

The remaining sections are organized as follows: Section 2 briefly discusses theworks which are related to the proposed approach, while Section 3 presents thedetail description of the proposed work. Section 4 does result analysis. Finally, Section 5 presents conclusion with some important points for future work.

I. BACKGROUND CONCEPT

In this section, we discuss some basic concepts, which may necessary to understand the whole paper.

A. CONCOLIC TESTING

Koushik Sen et al. [8] said that concolic testing process begins by executingthe program. Randomly generated inputs are used in this process. At the same time, Concolic testing tries to detect all the possible combination of paths withthe order of execution. In this method, constraint solvers are used to find apossible path with the help of symbolic input values. This method also gathersinput values with the constraints solver so that, it should generate high

Revised Manuscript Received on May 31, 2019

GayatriNayak, Department of Computer Science & Engineering, Siksha O Anusandhan Deemed to be University, Bhubaneswar, India.
Mitrabinda Ray, Department of Computer Science & Engineering, Siksha O Anusandhan Deemed to be University, Bhubaneswar, India.

pathcoverage. After every constraint are solved, this process uses concrete values.This executes the algorithm for the complete process. Let us explain the concolictesting using the following steps.
– First execute program by providing concrete values as input.
– Secondly this algorithm generates the symbolic path at the time of execution.
– Then, this process tries to negate a constraint on the path condition. Thisprocess executes it and solve to get a model.
– Finally, with the help of new concrete input values, the program is executedagain.

B. CONCOLIC EXECUTION

The concolic execution [4] procedure is given below with detail descriptions.
Code Instrumentation: This is a process where, the source code is addedwith additional instructions. This is required to help concolic execution. Thisinstrumentation process transfers original java code to a program syntax requiredfor the concolic tester tool.
Substitute Symbolic Variables: In the instrumented java code the actualvariables or parameters are replaced by their alias names in the program. Thesealias names are called symbolic notations for true parameters.
Test inputs: A random number generator function is used to fire random testinputs for carrying the execution process. Then, these steps are repeated untilall paths are covered.
Random Execution: It allows program execution with the random test inputs.Only in first iteration random test inputs are used. In subsequent iterationsthe constraint solver is used to solve the conditional statement. The functionalityof the constraint solver is to generate values for negating the condition. Thus bynegating the condition, it create a chance to visit the alternate path compare toits early run.
Symbolic Execution: Symbolic execution is the process of executing symbolicvariables with the help of constraint solver. This execution takes input fromrandom test generator.

II. PROPOSED WORK

Our proposed method is used to prioritize test suites using concolic testingfor the input Java programs.This approach uses branch coverage and path coveragecriteria as the metric to measure the potential of a test suite. The skeletaldiagram of our proposed model is shown in Figure 1. From the figure it is visiblethat, the input java program is instrumented first and then given as input tothe concolic tester named jCUTE. It generates a test suite file for each inputprogram. Once, the concolic testing process is over, we obtain the results such as branch coverage, path coverage percentage and execution time. Then, a modulenamed Test Suite Weight Evaluator is used to generate weight of each test suiteusing these three factors and prioritize the test suites.

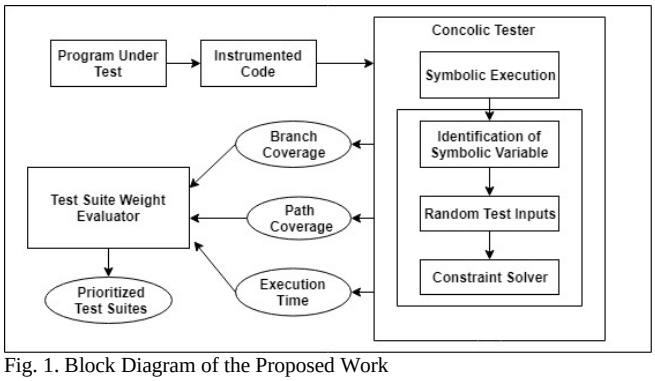

Fig. 1. Block Diagram of the Proposed Work

A. Overview

The proposed method gives a new approach to prioritize test suites withtwo objectives such that branch coverage, path coverage should be maximized.Concolic tester is used to generate test suites, branch coverage and path coverage.After achieving coverage scores, we compute weight values for each test suite using Eq. 3. Then, it helps us to prioritize the test suites.

B. DETAIL DESCRIPTION

This approach prioritizes test suites that are generated from input Java programs.In the initial phase, the given source Java file is translated to an instrumentedJava file. Here instrumentation means rewriting the Java file as perthe jCUTE syntax. This concolic testing process generates test cases and storesthem on a file called as a test suite. Apart from generating test cases, it generatesbranch coverage, path coverage etc. These two coverage metric are given as input to the Test Suite Weight Evaluatorto measure weight of each test suite. This weight value is calculated as stated inEq. 3. Based on these weight values, we prioritize the generated test suites stated in Eq. 1 and path coverage as mentioned in Eq. 2.These two coverage metric are given as input to the Test Suite Weight Evaluatorto measure weight of each test suite. This weight value is calculated as stated inEq. 3. Based on these weight values, we prioritize the generated test suites.

$$Branch\_Coverage(BC) = \frac{No\_of\_Branch\_Executed}{Total\_No\_of\_Branches} \quad (1)$$

$$Path\_Coverage(PC) = \frac{No\_of\_Path\_Executed}{Total\_No\_of\_Paths} \quad (2)$$

$$Test\_Suite\_Weight(TSW) = \alpha * BC + \beta * PC \quad (3)$$

C. MTSP: Multi-objective Test Suite Prioritization

Algorithm 1	Multi-objective Test Suite Prioritization
Input:	Source Program(P)
Output:	Prioritized Test Suite
1	Read a Java Program (P) // Input Program
2	For each conditional statement S ∈ P
3	Replace actual variable by Symbolic variable
4	End of For // Symbolic execution
5	For each conditional statement S ∈ P do
6	fire random inputs to symbolic variable in each condition. // Random Execution
7	record path if not visited

8	Use constraint solver to negate the condition
9	End of For
10	Record total Path Coverage (PC)
11	Record total Branch Coverage (BC)
12	Input PC, BC to the Test Suite Weight Evaluator
13	Compute Test Suite Weight using equation 3
14	Rank the test suites
15	Display Prioritized order of test suites
16	End

III. IMPLEMENTATION

This novel approach is experimented on Java programs. The main objectiveof this implementation process is to measure the importance of a test suite interms of its branch coverage and path coverage. This approach uses concolicexecution strategy to generate branch coverage and path coverage.

A. Setup

This experimental work is done on a standalone machine having i5 processor,4GB RAM and 32 bit windows 7 Operating System. For the concolic testerjCUTE, we need to have minimum JDK 1.6 version.

B. Result Analysis

Table 1. Experimental Result Table

TS_No	Input_Program	#TestCases	Branch%	Path%	TS_Weight	TS_Rank
1	GradeCalculation.java	5	0.98	0.63	1.61	10
2	SwitchTest.java	7	0.69	0.89	1.58	8
3	StringBuffer.java	6	0.63	1.00	1.63	4
4	SampleXCMLPolicy.java	21	0.86	1.00	1.86	5
5	MyQuickSort.java	3	0.83	1.00	1.83	16
6	Maths_call.java	9	0.66	1.00	1.66	11
7	SSort.java	9	0.86	0.28	1.40	6
8	Condition.java	8	0.90	0.98	1.88	3
9	Fruit_sales.java	11	0.55	1.00	1.55	1
10	Comparision1.java	19	0.98	1.00	1.98	2
11	SortingAlgos.java	6	0.74	1.00	1.74	9
12	HelloWorld.java	4	1.00	0.09	1.09	20
13	Elevator.java	7	0.93	0.23	1.16	19
14	Ordernumb.java	8	1.00	0.08	1.08	7
15	BSTree.java	9	0.45	0.06	0.51	13
16	SwitchTest2.java	14	0.77	1.00	1.77	12
17	AssertTest.java	7	0.88	0.01	0.89	14
18	LargestNumber.java	8	1.00	0.07	1.07	18
19	CAssume.java	2	0.50	0.92	1.42	17
20	Demo1.java	4	0.56	0.94	1.50	15

The experiment is conducted on twentystandard Javaprograms. These programsare available at an Open Systems Laboratory Repository [1]. Each Java programis tested using jCUTE, which is an open source tool available on theInternet[2]. This tool generates test cases, branch coverage, path coverage as itsoutput. During concolic execution, actual variables in a conditional statementare replaced by symbolic notations. Then, random numbers are given as input tosolve the condition and visit a path. Here, constraint solver is used to negate thegiven condition and then evaluate, which leads to visit the uncovered path. Table1 shows the results obtained after executing twenty Java programs. In Table 1,Column 2 shows list of Java programs tested, Column 3 represents test cases,Column 4 shows execution time, Column 5 shows path coverage percentage andColumn 6 shows the branch coverage percentage. Column 7 shows weight of eachtest suite calculated using Eq. 3. Column 8 shows rank of each test suite.

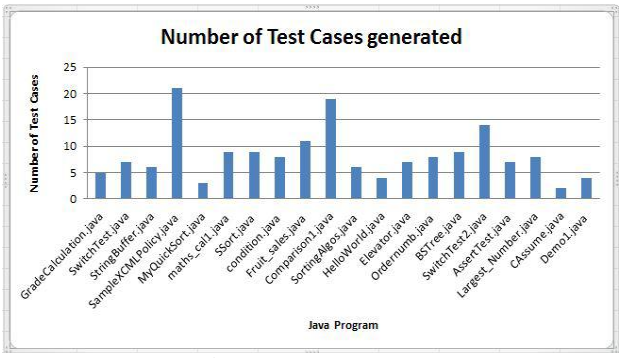

Fig. 2. Number of test cases generated by each program

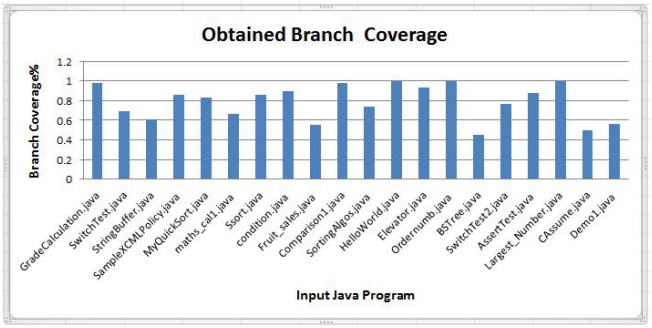

Fig. 3. Branch Coverage% Generated by Program

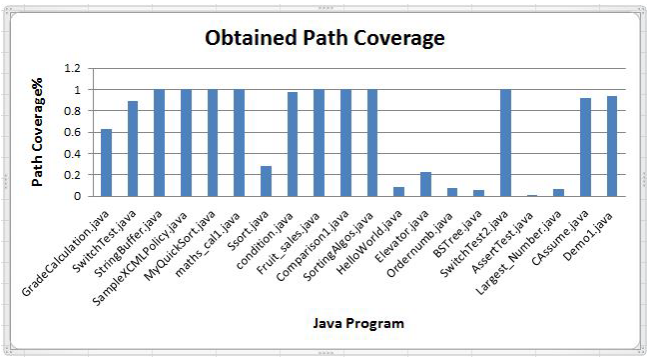

Fig. 4. Path Coverage% Generated by Program

IV. CONCLUSION

The proposed approach is implemented to measure the effectiveness of test suites by considering multiple coverage criteria. From the implementation and result analysis it is found that it gives a better ordering of test suites for achieving complete coverage adequacy. This approach is validated through anexperiment that is conducted on twenty Java programs. To have better test adequacy, we have taken branch coverage and path coverage together. This concept helps to build a quality software with better coverage. In future, this work can extend to prioritize test suites by measuring complexity of each branch condition and try to establish a relation between complexity of a condition and its fault detecting potential.

REFERENCES

1. Khatibsyarbini, Muhammad, MohdAdham Isa, Dayang NA Jawawi, and Rooster Tumeng. "Test case prioritization approaches in regression testing: A systematic literature review." *Information and Software Technology* 74-93 (2018).
2. Elbaum, S., Rothermel, G., Kanduri, S., Malishevsky, A.G.: Selecting a costeffectivetest case prioritization technique.

Software Quality Journal 12(3), 185–210(2004).
3. Elbaum, S., Malishevsky, A. G., &Rothermel, G. :Test case prioritization: A family of empirical studies. *IEEE transactions on software engineering*, *28*(2), 159-182(2002).
4. Holloway, C.M.: Towards understanding the do-178c/ed-12c assurance case. In:System Safety, incorporating the Cyber Security Conference 2012, 7th IET InternationalConference on. pp. 1–6. IET (2012).
5. Myers, G.J., Sandler, C., Badgett, T.: The art of software testing. John Wiley &Sons (2011).
6. Panigrahi, C.R., Mall, R.: A heuristic-based regression test case prioritization approachfor object-oriented programs. Innovations in Systems and Software Engineering10(3), 155–163 (2014).
7. Rothermel, G., Untch, R.H., Chu, C., Harrold, M.J.: Prioritizing test cases for regressiontesting. IEEE Transactions on software engineering 27(10), 929–948 (2001).
8. Sen, K., Agha, G.: Cute and jcute: Concolic unit testing and explicit path modelcheckingtools. In: International Conference on Computer Aided Verification. pp.419–423. Springer (2006).
9. Wong, W.E., Horgan, J.R., London, S., Agrawal, H.: A study of effective regressiontesting in practice. In: Software Reliability Engineering, 1997. Proceedings., TheEighth International Symposium on. pp. 264–274. IEEE (1997).

AUTHORS PROFILE

GayatriNayak: She is an Assistant Professor and pursuing Ph.D. in Computer Science & Engineering Department of Siksha O AnusandhanDeemed to be University. Her research area includes Softeare Testing, and soft computing.

Mitrabinda Ray: She is working as Associate Professor of the Computer Science & Engineering Department of Siksha O AnusandhanDeemed to be University. Her research area includes Software Testing, Data Mining and soft computing techniques