A HYBRID APPROACH FOR TEST CASE GENERATION

Vijayakumar.R, N.Basker

PG scholar, Department of computer science and engineering, Sona college of technology, Salem, TN, India
Assistant Professor, Department of Computer Science and Engineering, Sona College of Technology, Salem, TN, India
Vijaykumarcse12@gmail.com, bas2k9@gmail.com

Abstract: Today, Software testing is an essential part of successful software development process. The input executes the program and produces the expected output. The outcome of the software product depends on software testing. Manual testing is difficult to produce expected output. The manual testing takes long time to test. The major problem in manual testing is code coverage is not done at regular interval. Many techniques are used to automatically produce inputs in recent years. The test suite generation method is used to produce test suites with high code coverage. We produce a hybrid approach which provides the methodology to improve coverage level of branches in code at regular intervals. The main objective of the proposed system is to increase the coverage level with minimum test suite in a short time. The hybrid approach combines two techniques for improving the accuracy of branch coverage. The test cases are generated by using EVOSUITE tool. Optimization technique is based on coverage level of branch statements.

Keywords: Test case generation, branch coverage, search based technique.

I. INTRODUCTION

Software testing is the process of evaluation a software item to detect differences between given input and expected output. It also used to assess the feature of a software item. Generally software testing is an essential process for checking the quality of a software product. The test cases for a procedure consists of a sequence of input [6] values.There are several techniques which automatically generate inputs have been developed over the years. One of the methods is code coverage technique. Today we can able to produce test suites with high code coverage. The main problem in this process is the generated input may cause complex data structures. Our work concentrated to deliver an efficient test input generation [1] that will cover high code coverage that process at minimum time consumption. There has been an active research community investigating the generation of test inputs oracle with the use of model checking, the focus is on specification based test input generation where coverage of the specification is the goal. The problem of the expected outcome continues and has become known as the oracle problem. Sometimes, essential properties of programs are formally specified or have to hold universally such that no definite oracles need to be defined. Search based testing is used for test data generation. It is combined with symbolic execution [8] and given as input to test data. The proposed technique we combine search based software testing [2] (SBST) with dynamic symbolic execution and implementing using evosuite. The genetic algorithm[4] and it functions are used for optimizing the test suite. The mutations and crossover are used as GA function. The evosuite tool is used for test case generation. The evosuite works on byte code [6] level and collect the information fitness values.

II. LITERATURE STUDY

A. Pair-22Wise Test Coverage:

Combinatorial Interaction Testing (CIT) is a technique used to discover faults caused by parameter interactions in highly configurable systems. These systems tend to be large and exhaustive testing is generally impractical. Indeed, when the resources are limited, prioritization of test cases is a must. Important test cases are assigned a high priority and should be executed earlier. On the one hand, the prioritization of test cases may reveal faults in early stages of the testing phase. But, on the other hand the generation of minimal test suites that fulfill the demanded coverage criteria is an NP-hard problem. Therefore, search based approaches are required to find the near optimal test suites. In this work we present a novel evolutionary algorithm to deal with this problem.

B.Search based software test data generation for string data:

The test data is to cover program branches which depend on string predicates [2] such as string equality, string ordering and regular expression matching. Here the string cost functions are assessed by comparing their performance on a number of sample test programs.

1) String search space: The space of strings formed from the 16-bit character. The search space depends on the maximum and minimum length of the input string that is generated.
2) Character distance: the new cost function based on the pair wise comparison of character values. It define sum of the absolute differences between the ordinal character values of corresponding character pairs.

$$CD(s,t) = \sum_{i=0}^{i=k-1} |s_i - t_i| + 128(l-k)$$

C. Empirical analysis of the role of the test sequence length:

The internal state of the code is considered for software testing. The internal states [3] are presents in both object oriented and procedural software code. It analyzes a test length for particular code coverage. Which consists of four

parts of search based techniques: Random search, Hill climbing, Evolutionary algorithm, Genetic algorithm. In random search algorithm the fitness function is used for getting optimal solution.

D. Symbolic search based technique:

This method avoids some problem associated with symbolic execution [11] in loops. Meta heuristic algorithms rely on fitness function to find optimal solution. Whenever an input misses the target branches, the branch distance is used to find closely related branches. The branch distance is computed through,

$$|x - 0| + K,$$

Where,
x- Distance between two branches.
K- Failure constant.

E. Optimization of test cases:

The optimization technique is used in regression testing. Genetic algorithm [12] is used for optimization. Test cases are prioritized on the basis of lines of code modified in testing. The genetic algorithm functions like crossover and mutations are used.

III. EXISTING SYSTEM

The following figure shows the process of existing approach:

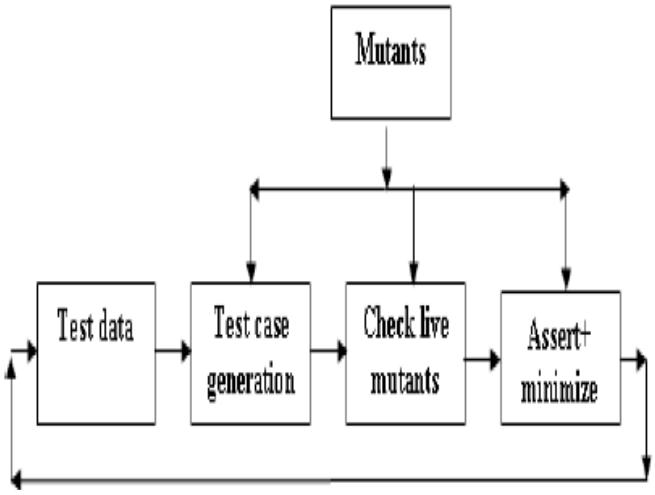

Fig.1 Architecture diagram

Traditional testing methods cover maximum branch statements. But it not covers all branches in a regular interval and it takes long time. The infeasible branches are not covered. The whole test case generation method presents a search based testing method [8] with genetic algorithm to cover all branches. This method covers all branches and feasible branch statements in minimum number of test cases at regular intervals. The genetic algorithm [12] is used for optimizing and prioritizing the test cases. The whole test case generation method consists of four modules such as: Search based testing, User test generation, Test suite optimization, Evolution of test analysis.

IV. METHODOLOGY

A. Search based testing:

The search based testing [2] can easily produce test data to satisfy the first and the third branch condition, but the second branch is an example of the flag problem, which gives the search no guidance. These testing make use of modern self control which is not dependent on search heuristics [4], but there are limits to both scalability and the types of constraints that can be handled. A problematic non-linear constraint, and if the Math library is not depend on source code or byte code, then the formation can be difficult in the first place.

B. Test case generation:

The test cases are generated based on test data. The test case generator executes the test data. The overlap between execution traces and control/call dependences leading to the current target determines the fitness of the individuals. The resulting test class is manually edited, in order to add declaration inside the body of each test method. This is the most human intensive phase in the whole testing process. It happens only at the end of the test case generation process [5], when the algorithm stops and returns the test suite. These test cases are executed by the test case executor. Their performance is directly provided by the tool which produces information about passed and failed test cases, as well as about the specific assertion that are not satisfied. The test suite produced by the Test case generator and is guaranteed to provide the level of coverage reached by the genetic algorithm.

D. Test suite optimization:

The genetic algorithm is based on the concepts of a quantum bits which are in quantum mechanics [7]. This algorithm generates a new test case they are targeted to be covered or to reach a maximum execution time. For each target to be covered we have to select at line, a maximum number of successive generations are produced out of initial population. Test cases in the current population are executed, possibly covering some of the previously uncovered targets. The targets still to be covered are updated, and if the currently selected target (t) has been covered, the most internal loop is exited and a new target is selected.

4. Evolution of test analysis:

The generated test suites are based on without prioritization and with prioritization. A new characteristic is said to be killed by a test suite if the test performance reports a failure. The percentage of killed characteristics shows the fault detection ability of the test suite. For a given program and given test generation, the two test suites generated with and without prioritization have the same tests and thus the same fault detection ability.

IV. PROPOSED SYSTEM

The hybrid approach method is used in proposed system. This hybrid approach combines search based techniques and dynamic symbolic execution technique for generating test data. This test data covers all branch statements. The accuracy of code coverage is increased. Minimum test

cases are generated with regular time interval. Show the following proposed system architecture diagram.

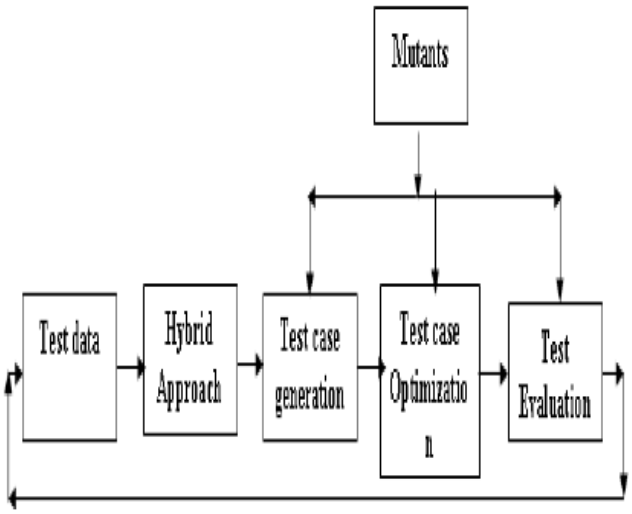

Fig.2 Proposed architecture

Genetic algorithm is used for prioritizing and optimizing the test cases. The test case priority is based on maximum coverage level of branch statements. Show the pseudo code for genetic algorithm.

Begin
T ← 0
Initialize P (t)

While (not termination condition)
Evaluate P (t)
Select P (t+1) from p (t)
Crossover P (t+1)
Mutate P (t+1)
T ← t+1
End while
End procedure

A. HYBRID APPROACH

The hybrid approach combine search based technique and dynamic symbolic method technique. This method is used to generate test input data. The test input data covers all branch statements including infeasible branch statement. The SBST technique with genetic algorithm generates test input data in following way.

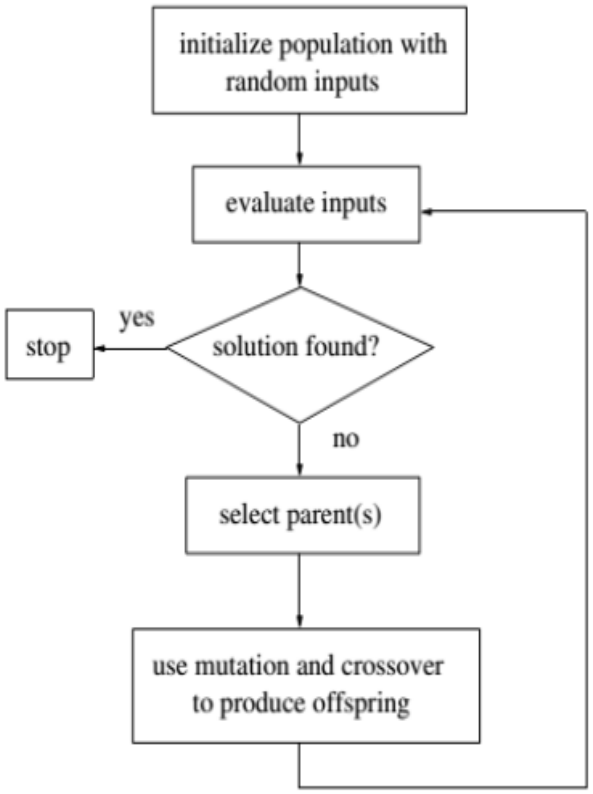

Fig.3 Flowchart for test data generation

The dynamic symbolic execution is used to automatically generate test inputs to achieve high code coverage. The dynamic symbolic execution execute given input data on program under test and at same time collect symbolic constraints obtained from predicates in branch statements. DSE is performed iteratively on the program under to increase code coverage. DSE uses a search strategy to flip a branching node in the path. Flipping a branching node in a path constructs a new path that shares the prefix to the node with the old path, but then deviates and takes a different path. Such a flipped path is feasible is checked by building a constraint system. If a constraint solver can determine that the constraint system is satisfiable within the available resources, DSE generates a new test input that will execute along the flipped path and achieve additional code coverage. DSE is able to generate a set of test inputs that achieve high code coverage.

V. CONCLUSION

Our approach achieves higher coverage than pure search requiring less repeat, and also same or higher coverage. At the same time, the use of search techniques [1] allows us to easily change the search objective. Finally the combination of search-based test generation with dynamic symbolic performance and make the best such as testability transformation or local search will further improve the achieved coverage when compared to the before approach

REFERENCES

[1]. Jon Edvardson, “A survey on Automatic Test Data Generation”.

[2]. Mohammad Alshraideh and Leonardo Bottaci, “Search based software test data generation for string data using program specific search operators,” Softw. Test. Verif. Reliab. pp. 175-203, 2006.

[3]. Andrea Arcuri, “A Theoretical and Empirical Analysis of the Role of Test Sequence Length in Software Testing for Structural Coverage, ” IEEE Trans.Software Eng., vol.38, no., pp. 497-519, 2012.

[4]. Shaukat Ali and Hadi Hemmmati, “A Systematic Review of the application and Empirical Investigation of Search-Based Test Case Generation,” IEEE Trans. Software Eng., vol.36 no. 6, pp. 742-761, 2010.

[5]. Gordon Frasor and Andrea Arcuri, “EvoSuite: Automatic Test Suite Generation for Object-Oriented Software”

[6]. Wiiem visser, “Test Inout Generation with Java PathFinder,” NASA Research center Moffett Field.

[7]. Mehrshad Khosravini, saadatPour Mozafari and Mohammad Mehdi Ebadzadeh, “Coverage Analysis of Quantum Genetic Algorithm”.

[8]. Jan Malburg, “Combining Search-based and Constrained-based Testing,” Saarland university.

[9]. A.Arcuru and L.Briand,”A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering,” Proc. 33rd Int’l Conf. Software Eng., pp. 1 10. 2011.

[10]. A.Arcuri, M.Z.Iqbal and Briand,”Random Testing: Theoretical Results and Practtical Implications,” IEEE trans. Software Eng., vol.38 no.2 pp.258-277, 2011

[11]. Arthur Baars, Mark Harman, Youssef Hassoun,kiran lakhotia, “Symbolic Search- Based Technique”.

[12]. T.Prem and T.Ravi, “Optimization of Test Cases by Prioritization,” Journal of computer science, pp. 972-980, 2013.

[13]. Rupa Kommineni, Vaibhu Ahlawat and anjaneyulu, “Functional Test Suite Minimization using Genetic aligorithm,” infosys labs briefings no.2 vol.11 2013.

[14]. Lingming Zhang, Tao Xie, Lu Zhang, Hong mei, “Test Generation via Dynamic Symbolic Execution for Mutation Testing”.

[15]. Gordon Fraser and Andreas Zeller, “Mutation-Driven Generation of Unit Tests and Oracles,” ISSA ’10. 2010.

[16]. L.Baresi, P.L, Lanzi and M.Miraz, “Testful: An Evolutionary Test Approach for Java,” Proc IEEE Int’l Conf. Software Testing, Verification and Validation, pp. 185-194, 2010.

[17]. P.McMinn, “Search-Based Software Test Data Generation: A Survey,” Software Testing. Verification and Reliability, vol.14, no.2 pp. 105-156, 2006.

[18]. M. Harrman and P.McMinn. “A Theoretical and Empirical Study of Search Based Testing: Local, Global, and Hybrid Search,” IEEE Trans. Software Eng., vol. 63, no. 2, pp. 226-247, 2010.

[19]. M.Harman, L.Hu, R.Hierons, J.Wegener, H.Sthamer, “Testability Transformation,” IEEE Trans. Software Eng,. Vol. 30, no.1, pp. 3-16,2004.