International Journal of Science Engineering and Advance Technology

A Searchbased Multi-Objective Approach To Generate Test Suites For High Branch Coverage

M.N.V.Surekha[1], K.T.V Subbarao [2]
Department of Computer Science And Engineering
Akula Sree Ramulu institute of Engineering and Technology, prathipadu,Tadepalligudem,A.P,India
Email- [1]srksri349@gmail.com,[2]ogidi@rediffmail.com.

Abstract:
A software test consists of an input that implements the program and a definition of the expected outcome. Many techniques to automatically create inputs have been proposed over the years and today are competent to produce test suites with high code coverage. Yet the problem of the expected outcome continues and has become known as the oracle problem. To make this feasible test generation needs to intend not only at high code coverage but also at small test suites that make oracle generation as easy as possible. Coverage goals are not sovereign, not evenly difficult and sometimes infeasible. The result of test generation is therefore dependent on the order of coverage goals and how many of them are possible. To overcome this problem we propose a novel paradigm in which whole test suites are developed with the aspire of covering all coverage goals at the same time while keeping the total size as small as possible. This approach has several advantages as for example, its efficiency is not affected by the number of infeasible targets in the code. We have implemented this novel approach in the EVOSUITE tool and evaluated it to the frequent approach of addressing one goal at a time.

Keywords: Search-based software engineering, length, branch coverage, genetic algorithm, infeasible goal, collateral coverage.

Introduction:
A common approach in the literature is to generate a test case for each coverage goal e.g., branches in branch coverage and then to join them in a single test suite. Though the size of a resulting test suite is hard to expect as a test case generated for one goal may absolutely also cover any number of further coverage goals. This is usually called collateral or serendipitous coverage. In detail the order in which each goal is selected can therefore play a major role as there can be addiction among goals. Although there have been efforts to develop collateral coverage to optimize test generation to the best of our knowledge there is no decisive assessment in the literature of their effectiveness. Not all bugs lead to program crashes and not always is there a formal requirement to ensure the correctness of a software test's outcome. A common scenario in software testing is therefore that test data are generated and a tester manually adds test oracles. As this is a difficult task it is important to create small yet representative test sets and this representativeness is classically measured using code coverage.

Related Work:
Solving path limitations generated with representative implementation is a popular approach to generate test data or unit tests and active symbolic execution as an addition can conquer a number of problems by combining tangible executions with symbolic execution. This design has been implemented in tools like DART and CUTE and is also practical in Microsoft's parameterized unit testing tool PEX or in the Dsc tool. Metaheuristic search methods have been used as a substitute to figurative execution-based approaches. The request of search for test data generation can be traced as back to the 1970s where the key concepts of branch distance and approach level were introduced to help search techniques in generating the right test data.

Existing System:
A common approach is to produce a test case for each coverage goal e.g., branches in branch coverage and then to merge them in a single test suite. However the size of a resulting test suite is complicated to predict as a test case generated for one goal may absolutely also cover any number of further coverage goals.

Disadvantages:
Some targets are more complex to cover than others. Coverage goals can be infeasible such that there exists no input that would cover them even if this particular infeasible branch may be easy to detect. This is not true in general and thus targeting infeasible goals will fail and the effort would be wasted.

Proposed System:
The evaluation of a novel approach for test data generation also known as a whole test suite

generation which perks up previous approach by targeting one goal at a time. We use an evolutionary technique in which instead of evolving each test case individually we develop all the test cases in a test suite at the same time and the fitness function believes all the testing goals simultaneously.

Advantages:

We formally prove the convergence of our proposed technique. The approach defers significantly better results.

Branch Coverage:

We focus on branch coverage as test criterion while the EVOSUITE approach can be widespread to any test criterion. A program encloses control structures such as if or while statements protected by logical predicates, branch coverage necessitate that each of these predicates evaluates to true and to false. A branch is infeasible if there exists no program input that appraises the predicate such that this particular branch is executed. For ease we treat switch/case constructs such that each case is treated like an individual if condition with a true and false branch. A method without any control structures consists of only one branch and consequently we necessitate that each method in the set of methods M is executed at least once.

Bloat Control:

Bloat happens when small insignificant developments in the fitness value are attained with larger solutions. This is very characteristic in classification/regression problems. When in software testing the fitness function is just the obtained coverage then we would not imagine bloat because the fitness function would assume only a few possible values. Though when other metrics are introduced with large domains of possible values e.g., branch distance and also for example mutation impact then bloat might occur.

Mutation:

The mutation operator for test suites is more complex than that used for crossover because it works at both the test suite and test case levels. To assess the fitness of a test suite it is essential to carry out all its test cases and gather the branch information. During the search on average only one test case is distorted in a test suite for each generation. This means that re-executing all test cases is not required as the coverage information can be carried over from the previous execution.

Random Test Cases:

Sampling a test case at chance means that each possible test case in the search space has a nonzero possibility of being sampled and these prospects are independent. In other words the probability of sampling a specific test case is stable and it does not depend on the test cases sampled so far. When a test case representation is compound and it is of variable length it is frequently not likely to sample test cases with uniform distribution i.e., each test case having the same probability of being sampled.

The Evosuite Tool:

EVOSUITE works on the byte-code level and collects all essential information for the test cluster from the byte-code via Java Reflection. This means that it does not need the source code of the SUT and in principle is also appropriate to other languages that compile to Java byte-code such as Scala or Groovy, for example. Note that we also believe branch coverage at the byte-code level. Because high-level branch statements in Java e.g., predicates in loop conditions are changed into simpler statements similar to atomic if statements in the byte-code. EVOSUITE is capable to handle all language constructs. Furthermore EVOSUITE treats each case of a switch/case creates like an individual if-condition. The number of branches at byte-code level is thus frequently larger than at source code level as intricate predicates are compiled into simpler byte-code instructions.

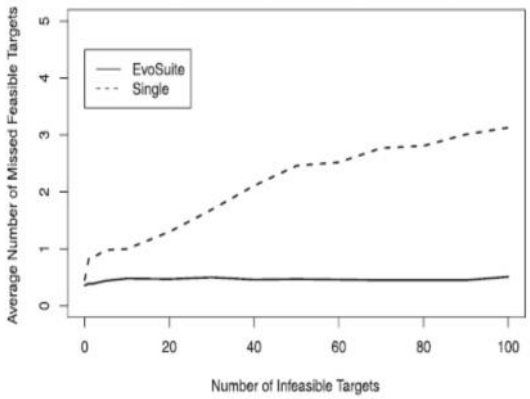
Algorithms Used:

Algorithm 1. The genetic algorithm applied in EVOSUITE

```
1  current_population ← generate random population
2  repeat
3     Z ← elite of current_population
4     while |Z| ≠ |current_population| do
5        P1, P2 ← select two parents with rank selection
6        if crossover probability then
7           O1, O2 ← crossover P1, P2
8        else
9           O1, O2 ← P1, P2
10       mutate O1 and O2
11       fP = min(fitness(P1), fitness(P2))
12       fO = min(fitness(O1), fitness(O2))
13       lP = length(P1) + length(P2)
14       lO = length(O1) + length(O2)
```

15 T_B = best individual of *current_population*
16 if $f_O < f_P \vee (f_O = f_P \wedge l_O \leq l_P)$ then
17 for O in $\{O_1, O_2\}$ do
18 if $length(O) \leq 2 \times length(T_B)$ then
19 $Z \leftarrow Z \cup \{O\}$
20 else
21 $Z \leftarrow Z \cup \{P_1 \text{ or } P_2\}$
22 else
23 $Z \leftarrow Z \cup \{P_1, P_2\}$
24 *current_population* $\leftarrow Z$
25 until solution found or maximum resources spent

Experimental Results:



It shows how EVOSUITE is unaware to the introduced infeasible branches while the presentation of the single branch strategy humiliates. To give more soundness to this analysis we also applied a Kruskal-Wallis test to confirm the contact of the number of infeasible targets on the number of missed i.e., uncovered feasible branches. The differences in coverage look so small that even if superior number of runs might decrease the p-value of the test then the differences would be so little to be of little practical interest anyway

Enhancement:

Previous work applicable only Object oriented Software. Future research on this approach could be applicable to procedural software and Android Apps and Iphone apps. Eventually improve the Performance of this tool and minimize the test cases length and it should be readable. Research on software testing produces many innovative automated techniques, but because software testing is by necessity incomplete and approximate, any new technique faces the challenge of an empirical assessment. Scientific advance is typically demonstrated using a set of examples that represent a particular problem addressed by the technique. However, demonstrating scientific advance is not necessarily the same as demonstrating practical value: A technique that works well on small, artificial problems might not scale up to the complexity of real systems. Ideally, one would use large "real-world" case studies to minimize the threats to external validity when evaluating research tools.

Conclusion:

We have revealed that optimizing whole test suites toward a coverage criterion is better to the traditional approach of targeting one coverage goal at a time. In our experiments this results in considerably better overall coverage with smaller test suites. While we have determined on branch coverage in this paper the answer also take over to other test criteria. As a result the capability to keep away from being misled by infeasible test goals can assist in overcoming similar problems in other criteria for example the equivalent mutant problem in mutation testing. Even though the results accomplished with EVOSUITE already shows that whole test suite generation is greater to single target test generation there is sufficient opportunity to further improve our EVOSUITE prototype.

References:

[1] S. Ali, L. Briand, H. Hemmati, and R. Panesar-Walawege, "A Systematic Review of the Application and Empirical Investigation of Search-Based Test-Case Generation," IEEE Trans. Software Eng., vol. 36, no. 6, pp. 742-762, Nov./Dec. 2010.

[2] M. Alshraideh and L. Bottaci, "Search-Based Software Test Data Generation for String Data Using Program-Specific Search Operators: Research Articles," Software Testing, Verification, andReliability, vol. 16, no. 3, pp. 175-203, 2006.

[3] L. Araujo and J. Merelo, "Diversity through Multiculturality: Assessing Migrant Choice Policies in an Island Model," IEEE Trans. Evolutionary Computation, vol. 15, no. 4, pp. 1-14, Aug. 2011.

[4] A. Arcuri, "It Really Does Matter How You Normalize the Branch Distance in Search-Based Software Testing," Software Testing, Verification and Reliability, http://dx.doi.org/10.1002/stvr.457, 2011.

[5] A. 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. 3, pp. 497-519, May/ June 2011.
[6] A. Arcuri and L. Briand, "Adaptive Random Testing: An Illusion of Effectiveness?" Proc. ACM Int'l Symp. Software Testing and Analysis, 2011.
[7] A. Arcuri 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,
[8] A. Arcuri and G. Fraser, "On Parameter Tuning in Search Based Software Engineering," Proc. Third Int'l Conf. Search Based Software Eng., pp. 33-47, 2011.
[9] A. Arcuri, M.Z. Iqbal, and L. Briand, "Black-Box System Testing of Real-Time Embedded Systems Using Random and Search-Based Testing," Proc. 22nd IFIP Int'l Conf. Testing Software and Systems, pp. 95-110, 2010,
[10] A. Arcuri, M.Z. Iqbal, and L. Briand, "Random Testing: Theoretical Results and Practical Implications," IEEE Trans. Software Eng., vol. 38, no. 2, http://doi.ieeecomputersoc..org/ 10.1109/TSE.2011.121, pp. 258-277, Mar./Apr. 2011.
[11] A. Arcuri and X. Yao, "Search Based Software Testing of Object- Oriented Containers," Information Sciences, vol. 178, no. 15, pp. 3075-3095, 2008.
[12] A. Baars, M. Harman, Y. Hassoun, K. Lakhotia, P. McMinn, P. Tonella, and T. Vos, "Symbolic Search-Based Testing," Proc. IEEE/ ACM 26th Int'l Conf. Automated Software Eng., 2011.
[13] 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.
[14] B. Baudry, F. Fleurey, J.-M. Je´ze´quel, and Y. Le Traon, "Automatic Test Cases Optimization: A Bacteriologic Algorithm," IEEE Software, vol. 22, no. 2, pp. 76-82, Mar./Apr. 2005.
[15] C. Csallner and Y. Smaragdakis, "JCrasher: An Automatic Robustness Tester for Java," Software Practice and Experience, vol. 34, pp. 1025-1050, 2004.

Mrs.M.N.V.Surekha is a student of Akula Sree Ramulu institute of Engineering & Technology, Tadepalligudem. Presently she is pursuing her M.Tech [Computer Science and Engineering] from this college and she received her M.C.A from Akula Sree Ramulu institute of Engineering & Technology, Tadepalligudem. to JNT University, Kakinada in the year 2010. Her area of interest includes Computer Networks and Object oriented Programming languages, all current trends and techniques in Computer Science.

Prof. K.T.V Subbarao, well known Author and teacher received M.Tech (CSE) and working as Principal, Akula SreeRamulu institute of Engineering and Technology, He is an active member of ISTE. He has 12 years of teaching experience in various engineering colleges. To his credit couple of publications both national and international conferences/journals. His area of interest includes cryptography and network security, Distributed databases, Operating systems and other advances in computer Applications.