A Novelty Search-based Test Data Generator for Object-oriented Programs

Mohamed Boussaa, Olivier Barais, Gerson Sunye and Benoit Baudry
INRIA Rennes, France
{mohamed.boussaa, olivier.barais, gerson.sunye, benoit.baudry}@inria.fr

ABSTRACT

In search-based structural testing, meta-heuristic search techniques have been frequently used to automate test data generation. In this paper, we introduce the use of novelty search algorithm to the test data generation problem based on statement-covered criterion. In this approach, we seek to explore the search space by considering diversity as the unique objective function to be optimized. In fact, instead of having a fitness-based selection, we select test cases based on a novelty score showing how different they are compared to all other solutions evaluated so far.

Keywords

Search-based Software Testing, Structural Coverage, Automated Test Data Generation, Novelty Search, Genetic Algorithm

1. INTRODUCTION

In general, manually creating test cases for testing software systems is time consuming and error-prone. Thus, the automation of this process becomes necessary. In fact, meta-heuristic search techniques such as Genetic Algorithms (GAs) are frequently used to automate the test data generation process and gather relevant test cases through the wide search space [3, 1]. These techniques are especially applied for structural white-box testing. For coverage-oriented approaches, applying Evolutionary Algorithms (EAs) to test data generation [4] has been focused on finding input data for a specific path of the program in accordance with a coverage criterion (e.g., longest path executed). The problem with coverage-oriented approaches is that search-based techniques cannot exploit the huge space of possible test data. In fact, some structures of the system may not be reached since they are executed only by a small portion of the input domain. The use of a fitness function as a coverage criterion to guide the search to detect relevant test data usually create one or more local optima to which the search may converge.

GECCO '15 July 11-15, 2015, Madrid, Spain

ACM ISBN 978-1-4503-3488-4/15/07.
DOI: http://dx.doi.org/10.1145/2739482.2764716

This process is known as *diversity loss*. Therefore, diversity maintenance in the population level is key for avoiding premature convergence. This issue is a common problem in GAs and many techniques are proposed to escape from local optima in multi-modal search space [2, 6]. However, all these alternatives use a fitness-based selection to guide the search.

In this paper, we propose the use of Novelty Search (NS) algorithm [5] to the test data generation problem. In this approach, we seek to explore the search space of possible test input values by considering diversity as the unique objective function to be optimized. In fact, instead of having a fitness-based selection, we rather select test cases based on a novelty score showing how different they are compared to all other test data evaluated so far.

The paper is organized as follows: section 2 describes the approach overview and our novelty search adaptation. Concluding remarks and future work are provided in section 3.

2. NOVELTY SEARCH FOR TEST DATA GENERATION

2.1 Approach Overview

Our test data generation framework aims to fully automate the test data generation process without requiring user intervention. In fact, we have considered our System Under Test (SUT) as a gray or semi transparent box in where the internal structure is partially known. Thus, we can design test cases through the exposed interfaces and conduct a code coverage analysis from the general structure of our target SUT. For example, within *apache.commons.math* library, Methods Under Test (MUTs) are accessed through some specific Java interfaces. Each interface belongs to a subpackage of the whole library and exhibits a set of methods. Our test data generation framework will rely on this concept to generate automatically test cases. In fact, as shown in Figure 1, starting from an input interface and a code source package, the testing framework is able to: (1) generate automatically sequences of method invocation through the input interface, (2) generate relative test data using NS, (3) execute test cases on target Classes Under Test (CUTs) and then (4) analyze code coverage within the source package. The process is iterated until a termination criterion is met (e.g., number of iterations)

In our proposal, we use the same logic as GAs to implement the NS algorithm. However, we make some changes and we add new settings to optimize the test data generation. In fact, instead of using a fitness function to evaluate

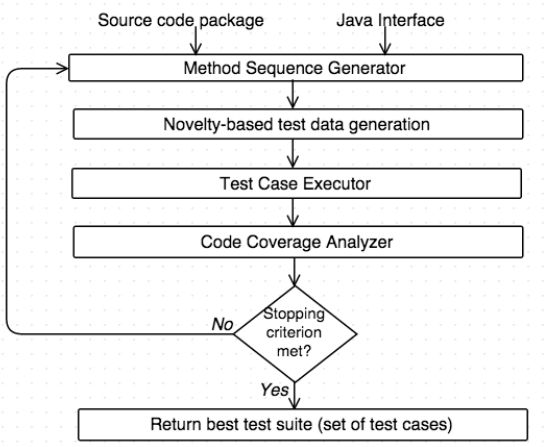

Figure 1: Approach overview

generated test cases, we define a new measure of novelty to maximize. We replace, as well, the fitness-based selection (for fittest test cases) with a novelty-based one. This may favor the diversity of generated test data. Finally, we add an archive that acts as a memory of previously generated test cases. This archive is used to calculate the novelty metric. Otherwise, we keep the same settings as GAs, namely the crossover and mutation operators.

For our NS adaptation, we have to define the number of iterations N, the population size and the minimum coverage value. This latter defines the threshold of covered statements that should be reached. Test cases that exceed this threshold are automatically added to the set of relevant test cases. As well, we have to define a novelty threshold T that defines the threshold for how novel a test suite has to be before it is added to the archive. In addition, we define a maximum size limit for the archive L and a k number that will be used in calculating the novelty metric.

2.2 Novelty Metric

The Novelty metric expresses the sparseness of a test suite. It measures its distance to all other test cases in the current population and to all test cases that were discovered in the past (i.e., test cases in the archive). This measure expresses how unique the test suite is. We can quantify the sparseness of a set of test cases as the average distance to the k-nearest neighbors. The distance between two test suites is computed as a Manhattan distance between the input parameter values of all methods tested in the test suite. A candidate solution represents the set of methods signatures declared in the interface. Thereby, we represent this solution as a vector where each dimension has a method name, a list of parameters types and a list of test data (the genotype) Formally, we define this distance between two solutions as follows :

$$distance(t1, t2) = \sum_{i=1}^{m} \sum_{j=1}^{p} |t1(M_i, P_j) - t2(M_i, P_j)| \quad (1)$$

where $t1$ et $t2$ are two selected test suites (solutions), m is the number of methods composing the test suite, p is the number of parameters composing a method. The couple (M_i, P_j) returns the j^{th} parameter value of the i^{th} method M relative to a test suite ($t1$ or $t2$). Since we are using Java primitive types for test data such as floats, integers, doubles, etc, it is easy to calculate this distance for numerical parameters values. However, for string data types we use the Levenshtein Algorithm[1] to measure the strings distance.

To measure the sparseness of a test suite, we will use the previously defined distance to compute the average distance of a test suite to its k-nearest neighbors. In this context, we define the novelty metric of a particular solution as follows:

$$NM(S) = \frac{1}{k} \sum_{i=1}^{k} distance(S, \mu_i) \quad (2)$$

where μ_i is the i^{th} nearest neighbor of the solution S within the population and the archive of novel individuals. Finally, we have normalized the resulted novelty distance in the range [0-100].

3. CONCLUSION AND FUTURE WORK

In this paper, we proposed a new adaptation of the NS algorithm to test data generation problem. Using the NS approach is clearly a divergent evolutionary technique, inspired by natural evolution's drive to novelty that directly rewards novel behaviors instead of progress towards a fixed objective.

As a future work, we aim to conduct an empirical evaluation of our NS approach by comparing it to fitness-based and random approaches. In addition, we can optimize our approach by adding diversity as an addition goal to a traditional objective driven approach to form a multi-objective optimization problem. Finally, since we are testing gray-box systems, we can apply this approach, as well, for black-box testing. In this case, we will be able to measure some non-functional properties such as memory usage and CPU consumption.

4. REFERENCES

[1] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege. A systematic review of the application and empirical investigation of search-based test case generation. *Software Engineering, IEEE Transactions on*, 36(6):742–762, 2010.

[2] S. Das, S. Maity, B.-Y. Qu, and P. N. Suganthan. Real-parameter evolutionary multimodal optimizationâĂŤa survey of the state-of-the-art. *Swarm and Evolutionary Computation*, 1(2):71–88, 2011.

[3] M. Harman and P. McMinn. A theoretical and empirical study of search-based testing: Local, global, and hybrid search. *Software Engineering, IEEE Transactions on*, 36(2):226–247, 2010.

[4] K. Lakhotia, P. McMinn, and M. Harman. Automated test data generation for coverage: Haven't we solved this problem yet? In *Testing: Academic and Industrial Conference-Practice and Research Techniques, 2009. TAIC PART'09.*, pages 95–104. IEEE, 2009.

[5] J. Lehman and K. O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, pages 329–336, 2008.

[6] H. M. Pandey, A. Chaudhary, and D. Mehrotra. A comparative review of approaches to prevent premature convergence in ga. *Applied Soft Computing*, 24:1047–1077, 2014.

[1] http://www.levenshtein.net/