Diversity in Search-based Unit Test Suite Generation

Nasser M. Albunian

The University of Sheffield
nmalbunian1@sheffield.ac.uk

Abstract. Search-based unit test generation is often based on evolutionary algorithms. Lack of diversity in the population of an evolutionary algorithm may lead to premature convergence at local optima, which would negatively affect the code coverage in test suite generation. While methods to improve population diversity are well-studied in the literature on genetic algorithms (GAs), little attention has been paid to diversity in search-based unit test generation so far. The aim of our research is to study the effects of population diversity on search-based unit test generation by applying different diversity maintenance and control techniques. As a first step towards understanding the influence of population diversity on the test generation, we adapt diversity measurements based on phenotypic and genotypic representation to the search space of unit test suites.

Keywords: Search-Based Test Generation; Population Diversity; Genetic Algorithm

1 Introduction

As software testing is a time-consuming, laborious, and error-prone task, developers can choose to generate tests automatically. In the context of unit testing object oriented software, where tests are sequences of calls on a class under test (CUT), Genetic Algorithms (GAs) have been successfully applied for generating tests achieving high code coverage [7].

The success of GAs is dependent on the diversity maintained in the search population [10]. If the individuals of the population all become too similar and lack diversity, then the search may converge on a local optimum of the objective function. This reduces the effectiveness of the GA, and in the case of search-based test generation, premature convergence would imply a reduced code coverage.

To avoid premature convergence and maintain diversity in the search population, different diversity techniques have been proposed at the genotype and the phenotype levels [19]. While the problem of maintaining the diversity has been extensively investigated within different domains of evolutionary algorithms (e.g., [3]), much less is known about diversity in whole test suite generation.

The aim of our research is to investigate the effects of population diversity on the generation of unit tests for Java programs. The attempt is to see whether the diversity of the search population helps in generating test suites that are capable of achieving the search goal (e.g. higher code coverage) and, on the other hand, reducing the occurrence of premature convergence. To do so, different diversity maintenance and control techniques will be examined to determine their effects

on the population diversity with the hope of proposing techniques that improve the diversity to generate better test suites. However, as a first step towards understanding the influence of population diversity on the test generation, we adapt diversity measurements based on phenotypic and genotypic representation to the search space of unit test suites.

2 Background

2.1 Search-Based Software Testing (SBST)

Search-Based Software Testing (SBST) describes the application of meta-heuristic optimisation techniques to the automation of various software testing tasks. In particular, SBST is frequently applied to generate test data [9]. When test data is numeric then local search algorithms such as hill climbing have been used successfully; in other domains, such as unit testing, GAs are more common.

In a GA, a population of candidate solutions is gradually evolved toward an optimal solution. The algorithm typically starts with a population of random individuals that will be iteratively evolved over many generations. In each generation, the processes of natural evolution are mimicked: Every individual in the population is evaluated by a fitness function, which determines how close this individual is to the desired solution. The fitter an individual, the more likely it is selected from the current population and used for recombination using crossover and mutation operators while building the next generation of the GA population. Higher selective pressure leads to a more biased selection of parent individuals and thus less diversity. Similarly, higher mutation rate can lead to more diverse individuals.

Although studies have shown the effectiveness of GAs for test generation [7], the application domain of unit testing seems to be special. In particular, Shamshiri et al. [18] recently showed that there are cases, in particular when the code coverage based fitness function hardly provides guidance, where random search is at least as effective as a GA. Shamshiri et al. hypothesise that in these cases the GA suffers from reduced diversity compared to the random search.

In the context of generating tests for object oriented programs, a common approach lies in evolving entire sets of unit tests [6]. The representation of a solution is a test suite, which is a set of test cases [6]. Each test case is a sequence of calls on the CUT. As the ideal test suite size is not known a priori, the number of tests in a test suite and the number of statements in a test case are variable and can be changed by the search operators.

The fitness function used to guide the search is based on code coverage. One of the most common coverage criteria in practice is branch coverage [6]. The overall fitness value of a test suite is the sum of normalised branch distance values (i.e., values estimating the distance to conditions evaluating to true and false), so that a test suite with 100% branch coverage has a fitness value of 0 [6].

2.2 Measuring Population Diversity

Maintaining population diversity during the evolution of EAs is widely believed to be crucial for avoiding premature convergence. Diversity measures are intended to quantify the variety of population's individuals in the base of structural or

behavioural levels. These levels differ among different domains [19], e.g., the structure of an individual in the case of genetic programming (GP) is not similar to the one with other evolutionary algorithms (EAs).

In general, there are three different levels of diversity measurement [19]: Genotype level, Phenotype level, and Composite measures. The genotypic diversity measures the structural (i.e., syntactic) differences among the individuals of a population. In contrast, the phenotypic diversity is based on the behavioural (i.e., semantic) differences in the population's individuals. Finally, composite measures are a combination of genotypic and phenotypic measures. The genotypic and phenotypic diversity measures have been intensively applied for GPs, but have seen less attention in other EAs [19].

2.3 Maintaining Population Diversity

There are several techniques that have been used to maintain the diversity of a population during the evolution process. Črepinšek et al. [19] classified these techniques into non-niching and niching techniques.

The purpose of niching techniques is to alleviate the effects of genetic drift by segmenting the population into subpopulations to locate multiple optimal solutions [15]. Fitness sharing is the most popular and well-known approach among the niching techniques [15]. It aims to find multiple peaks in the solution space, where each subpopulation around a peak represents a niche where individuals share the same resource (i.e., fitness value). The idea behind fitness sharing is to decrease the value of the resource that is shared by the individuals of a niche when the number of individuals is high. In contrast, the resource value will be increased when there are few individuals in a niche, which gives these individuals higher probability to be selected for next generations.

On the other hand, the non-niching techniques include different approaches such as increasing population size, changing selection pressure, or applying replacement restrictions [19].

However, there are other diversity techniques (e.g., dynamic adaptation of crossover and mutation rates [14]) that rely on the feedback that is provided by the population diversity measures to steer the evolution towards better exploration and exploitation of the search space. These techniques are known by the diversity control techniques [19].

2.4 Diversity in Test Generation

In the context of test generation, there have been several studies on generating diverse test cases [16]. These studies adapt different evolutionary aspects to be based on diversity of tests. The aim of these studies is for the tests within the final test suite to be diverse, rather than the individuals in the search population (i.e., diverse test suites).

One aspect that can be modified to be based on diversity is fitness function. For example, Feldt et al. [4] modified the fitness function of their evolutionary algorithm to measure the fitness of a test case based on its similarity to other test cases. The measure is based on information distance (i.e., information about the actual execution of a test) between each two test cases.

Another approach that modifies the fitness function in a GA to be based on the diversity of black-box string test cases is presented by Shahbazi et al. [17].

The fitness function, in this case, measures the diversity of test cases as the distance between every test case and its nearest test case (i.e., higher fitness value indicates more diverse test cases). The authors examined different string distance functions including Levenshtein, Hamming, Cosine, Manhattan, Cartesian, and LSH distance functions and found that the LSH distance function performs better in measuring the distance.

Alshraideh et al. [2] defined a diversity measure that is used to rank individuals in the population (i.e., an individual with the highest distance from other individuals receives better rank). Their measure is enriched to their approach that directs the search when the fitness function at any test goal can not find scarce test inputs.

The selection of parents during the evolution is another aspect that can be adapted to be based on tests diversity. An approach that is applied by Palomba et al. [11] incorporated two metrics into the Many-Objective Genetic Algorithm (MOSA) within the selection mechanism as a secondary objective. One metric is to reduce the test coupling (i.e., higher diversity) and the other is to increase the test cohesion (i.e., lower test length).

Panichella et al. [13] proposed a novel many-objective GA that targets all branches in the software under test as different objectives to be optimised simultaneously. To maintain better diversity among test cases, the authors applied the crowding distance in the selection scheme; a test case has a higher probability to be selected if it has a higher distance from the other test cases.

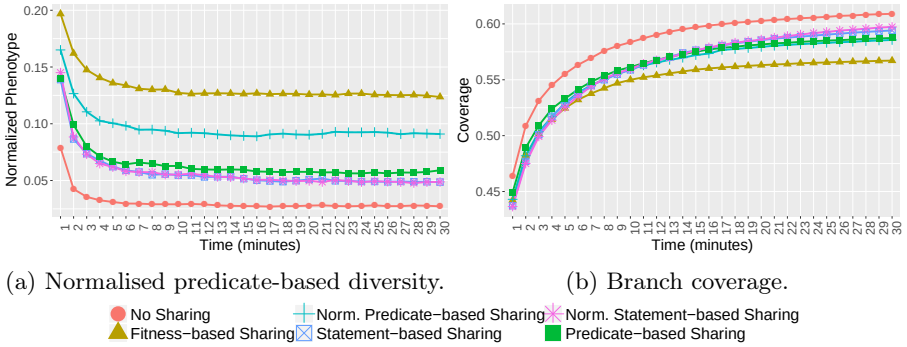
3 Preliminary Work

Our goal is to analyse how test suite generation for Java programs is influenced by test case diversity, and whether maintaining diversity during the search can lead to better coverage results. We therefore started by applying three diversity measure techniques based on the phenotypic and genotypic levels. We measure the phenotypic diversity based on the fitness entropy and test execution traces, and we define a genotypic measurement based on the syntactic representation of test suites.

To maintain population diversity, we investigated the influence of five selection mechanisms (i.e., Roulette wheel, Tournament selection size 2 and size 7, and Rank selection bias 1.2 and bias 1.7) and five configurations of fitness sharing (i.e., using fitness values, predicate distance, normalised predicate distance, statement distance, and normalised statement distance to define niches) on the diversity of the generated test suites.

To run our experiments, we used EvoSuite (version 1.0.3) with a "vanilla" configuration [5] with only branch coverage as target criterion, no test archive, and search budget of 30 minutes. We ran EvoSuite 10 times on 347 complex classes from the DynaMOSA study [12].

The results show that for selection mechanisms, the main constant across all types of diversity is that the roulette wheel selection leads to higher increase in the population diversity. If we compare tournament selection with tournament sizes 2 and 7, then the larger tournament size is expected to lead to less diversity, and indeed this is confirmed by the entropy measure. For rank selection, the higher rank bias of 1.7 results in less diversity than a bias of 1.2, as expected.



(a) Normalised predicate-based diversity. (b) Branch coverage.

Fig. 1: Diversity throughout the evolution using sharing algorithms.

On the other hand, when comparing the five configurations of fitness sharing with the baseline without fitness sharing, the fitness sharing shows clear increase in diversity; fitness sharing based on fitness values shows the largest increase across all metrics. The normalised versions of the metrics are usually close to the non-normalised, except when normalised predicate-distance is used to drive fitness sharing, in which case it results in larger increase, as shown in Figure 1a.

Figure 1b shows the average code coverage over time. Interestingly, all applications of fitness sharing lead to lower code coverage. Indeed the configuration that results in the highest diversity (fitness-based sharing) results in the lowest code coverage. Our conjecture for this observation is that diversity increases length. The likely reason for this lies in the variable size representation used in test suite generation, where higher diversity leads to larger individuals, and thus more expensive fitness evaluations.

4 Future Work

As our preliminary results indicate that increasing population diversity leads to an increase in the length of individuals rather than improving the coverage, we will therefore consider devising diversity metrics for test suites that are not influenced by length. We will further investigate alternative ways to increase population diversity, for example by modifying the mutation operators. In particular, we anticipate potential by making mutations informed about the state of the search (e.g., which methods are not covered yet) rather than random.

The adapted diversity measures can be extended and used with other configurations such as calculating the overall predicate diversity based on the maximum/minimunm value of the resultd distance values. We will also extend our current phenotypic measure that is based on the execution profile of predicates in the CUT to be based on branches of each predicate and, as an alternative, based on the state of the object under test.

In addition, we aim to apply a generic modification of the search algorithms that improves the diversity such as adapting the Novelty search algorithm [8] to fit our diversity metrics and applying the Cellular form of GAs [1] to generate test suites.

References

1. Alba, E., Dorronsoro, B.: Cellular Genetic Algorithms. Operations Research/Computer Science Interfaces Series, Springer US (2009)
2. Alshraideh, M., Bottaci, L., Mahafzah, B.A.: Using program data-state scarcity to guide automatic test data generation. Software Quality Journal 18(1), 109–144 (2010)
3. Burke, E., Gustafson, S., Kendall, G., Krasnogor, N.: Advanced population diversity measures in genetic programming. In: International Conference on Parallel Problem Solving from Nature. pp. 341–350. Springer Berlin Heidelberg (2002)
4. Feldt, R., Torkar, R., Gorschek, T., Afzal, W.: Searching for cognitively diverse tests: Towards universal test diversity metrics. In: Software Testing Verification and Validation Workshop, 2008. ICSTW'08. pp. 178–186. IEEE (2008)
5. Fraser, G., Arcuri, A.: Evosuite: automatic test suite generation for object-oriented software. In: Proc. of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. pp. 416–419. ACM (2011)
6. Fraser, G., Arcuri, A.: Whole test suite generation. Software Engineering, IEEE Transactions on 39(2), 276–291 (2013)
7. Fraser, G., Arcuri, A.: A Large-Scale Evaluation of Automated Unit Test Generation Using EvoSuite. ACM Trans. Softw. Eng. Methodol. 24(2), 8:1–8:42 (Dec 2014)
8. Lehman, J., Stanley, K.O.: Exploiting Open-Endedness to Solve Problems Through the Search for Novelty. In: ALIFE. pp. 329–336 (2008)
9. McMinn, P.: Search-based software testing: Past, present and future. In: Software testing, verification and validation workshops (icstw), 2011 ieee fourth international conference on. pp. 153–163. IEEE (2011)
10. Morrison, J., Oppacher, F.: Maintaining Genetic Diversity in Genetic Algorithms Through Co-evolution. In: Proceedings of the 12th Biennial Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence. pp. 128–138. AI '98, Springer-Verlag, London, UK, UK (1998)
11. Palomba, F., Panichella, A., Zaidman, A., Oliveto, R., De Lucia, A.: Automatic test case generation: what if test code quality matters? In: Proceedings of the Int. Symposium on Software Testing and Analysis. pp. 130–141. ACM (2016)
12. Panichella, A., Kifetew, F., Tonella, P.: Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets. IEEE Transactions on Software Engineering (2017)
13. Panichella, A., Kifetew, F.M., Tonella, P.: Reformulating branch coverage as a many-objective optimization problem. In: Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on. pp. 1–10. IEEE (2015)
14. Pellerin, E., Pigeon, L., Delisle, S.: Self-adaptive parameters in genetic algorithms. pp. 53–64. International Society for Optics and Photonics (2004)
15. Sareni, B., Krahenbuhl, L.: Fitness Sharing and Niching Methods Revisited. Trans. Evol. Comp 2(3), 97–106 (Sep 1998)
16. Shahbazi, A.: Diversity-Based Automated Test Case Generation. Ph.D. thesis, University of Alberta (2015)
17. Shahbazi, A., Miller, J.: Black-Box String Test Case Generation through a Multi-Objective Optimization. IEEE Transactions on Software Engineering 42(4) (2016)
18. Shamshiri, S., Rojas, J.M., Fraser, G., McMinn, P.: Random or Genetic Algorithm Search for Object-Oriented Test Suite Generation? In: Proc. of the Conference on Genetic and Evolutionary Computation. pp. 1367–1374. ACM (2015)
19. Črepinšek, M., Liu, S.H., Mernik, M.: Exploration and Exploitation in Evolutionary Algorithms: A Survey. ACM Comput. Surv. 45(3), 35:1–35:33 (Jul 2013)