Post-Print Version
DOI: http://doi.acm.org/10.1145/1569901.1570253

An Adaptive Strategy for Improving the Performance of Genetic Programming-based Approaches to Evolutionary Testing

José Carlos Bregieiro Ribeiro
Polytechnic Institute of Leiria
Morro do Lena, Alto do Vieiro
Leiria, Portugal
jose.ribeiro@estg.ipleiria.pt

Mário Alberto Zenha-Rela
University of Coimbra
CISUC, DEI, 3030-290,
Coimbra, Portugal
mzrela@dei.uc.pt

Francisco Fernández de Vega
University of Extremadura
C/ Sta Teresa de Jornet, 38
Mérida, Spain
fcofdez@unex.es

ABSTRACT

This paper proposes an adaptive strategy for enhancing Genetic Programming-based approaches to automatic test case generation. The main contribution of this study is that of proposing an adaptive Evolutionary Testing methodology for promoting the introduction of relevant instructions into the generated test cases by means of mutation; the instructions from which the algorithm can choose are ranked, with their rankings being updated every generation in accordance to the feedback obtained from the individuals evaluated in the preceding generation. The experimental studies developed show that the adaptive strategy proposed improves the algorithm's efficiency considerably, while introducing a negligible computational overhead.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing tools (e.g., data generators, coverage testing)*

General Terms

Verification

Keywords

Evolutionary Testing, Search-Based Software Engineering, Genetic Programming, Adaptive Evolutionary Algorithms

1. INTRODUCTION

Evolutionary Algorithms (EAs) are general – yet powerful – methods for search and optimization. However, if knowledge about a problem is available, a bias can be introduced directly into the problem so as to remove (or penalize) undesirable candidate solutions and improve the efficiency of the search. Unfortunately, *a priori* knowledge about the intricacies of the problem is frequently unavailable, but this does not necessarily prevent introducing an appropriate specific bias into an evolutionary problem; for many tasks, it is possible to dynamically adapt aspects to anticipate the regularities of the environment and improve solution optimization or acquisition speed [1]. Adaptive Evolutionary Algorithms (AEAs) are distinguished by their dynamic manipulation of selected parameters or operators during the course of evolving a problem solution [2].

The focus of our on-going research is on generating test data for the unit-testing of Object-Oriented (OO) programs using EAs. Our approach to Evolutionary Testing (ET) involves representing and evolving test cases using the Strongly-Typed Genetic Programming (STGP) paradigm. Test case quality evaluation involves instrumenting the object under test, and executing it using the generated test cases – with the intention of collecting trace information with which to derive coverage metrics. The goal is that of finding a test set that achieves full structural coverage of the test object [4]. The main contribution of this study is that of proposing a novel adaptive approach to ET.

2. AN ADAPTIVE STRATEGY TO ET

In STGP-based ET approaches the Method Call Sequences (MCSs) that compose test cases are encoded as STGP trees. Each tree subscribes to a Function Set, which must be specified beforehand and defines the STGP nodes legally permitted in the tree; the Function Set thus contains the set of instructions from which the algorithm can choose, and defines the constraints involved in MCS construction. The Function Set definition process is, however, heavily problem-specific and usually falls on the users' hands; there are good reasons to suppose that there is no one strategy that will be optimal for all ET problem domains.

The novel strategy proposed is based on the concept of dynamically adapting the Function Set's constraints selection probabilities. During an evolutionary run, it is possible to perceive that the introduction of certain instructions should be favoured, e.g. because they had been less prone to throw runtime exceptions and their introduction will likely contribute to test case feasibility [5], or simply because they had been used less frequently and their inclusion would promote diversity and allow trying out different objects and states in the search for full structural coverage.

The main process by which new genetic material is introduced during the evolutionary search is by means of the mutation operator. Whenever mutation occurs, a new (sub)tree must be created; we propose employing Luke's Strongly-typed Probabilistic Tree Creation 2 (PTC2) algorithm [3] to perform this task, so as to take advantage of the built-in feature that allows assigning probabilities to the selection of

Copyright is held by the author/owner(s).
GECCO'09, July 8–12, 2009, Montréal Québec, Canada.
ACM 978-1-60558-325-9/09/07.

Table 1: Percentage of runs to attain full coverage.

Test Object	adaptive	static
`Vector`	***80%***	***77%***
`BitSet`	***56%***	***45%***

constraints. What's more, we have modified this algorithm in order to be able to dynamically update the constraints' probabilities during the evolutionary run. Namely, the constraints' selection probabilities are recalculated every generation as detailed in the following Section.

3. TECHNICAL APPROACH

Let the *constraint selection ranking* of constraint c in generation g be identified as ρ_c^g. Then, ρ_c^g is updated, at the beginning of each generation, as follows.

$$\rho_c^g = \rho_c^{g-1} - \lambda_c^{g-1} - \sigma_c^{g-1} - \gamma_c^{g-1} \quad (1)$$

The *runtime exceptions caused factor* λ is equal to the number of runtime exceptions thrown by instructions corresponding to constraint c dividing by the total number of trees (i.e., MCSs). This factor's main purpose is that of penalizing the ranking of constraints corresponding to instructions that have caused runtime exceptions to be thrown in the preceding generation.

The *runtime exceptions caused by ancestors factor* σ is equal to the sum of the inverses of the ancestry levels ($2 =$ parent, $3 =$ grandparent, ...) of the ancestors of constraint c that threw runtime exceptions. This factor's main purpose is that of penalizing the ranking of constraints which have participated in the composition of sub-trees (i.e., sub-MCSs) that have caused runtime exceptions to be thrown in the preceding generation.

The *constraint diversity factor* γ is equal to the deviation between the number of times constraint c was used and the average number of times all constraints were selected for inclusion in the MCSs of the preceding generation, dividing by the deviation range (i.e., the maximum deviation minus the minimum deviation). This factor's main purposes are those of allowing constraints to recover their ranking if they have been being used infrequently, and penalizing the ranking of constraints which have been selected too often.

The constraints' selection probabilities correspond to the normalized values of the ρ factors.

4. EXPERIMENTAL STUDIES

The adaptive technique described in the previous Sections was embedded in eCrash tool [4], and a series of empirical studies were executed with the objective of assessing its contribution to enhance the test case generation procedure. The Java `Vector` and `BitSet` classes (JDK 1.4.2) were used as test objects, and 20 runs were executed for each of the 67 public methods of these classes; half of these runs were performed employing the adaptive strategy, and half using a "static" approach for comparison purposes.

The results clearly indicated that the test case generation procedure's success rate was improved by the inclusion of the adaptive methodology. Regarding the overall average number of runs attaining full structural coverage, the adaptive strategy enhanced results by 3% for the `Vector` class and by 11% for the `BitSet` class (Table 1). In terms of the percentage of runs to attain full structural coverage per method, the adaptive strategy outperformed the static approach in 28.4% of the methods tested, whereas the latter only surpassed the former in 5.9% of the situations. What's more, the adaptive strategy allowed attaining full structural coverage for 4 methods in which the success rate had been of 0% using the non-adaptive strategy – which indicates that this strategy is specially suited for overcoming some difficult state problems. In terms of speed, the overhead introduced by embedding the adaptive strategy into the evolutionary algorithm was a mere 0.19%.

5. CONCLUSIONS

The main contribution of this work is that of proposing a dynamic adaptation strategy for promoting the introduction of relevant instructions into the generated test cases by means of mutation. The adaptive ET strategy proposed obtains feedback from the individuals produced and evaluated in the preceding generation, and dynamically updates the selection probability of the constraints defined in the Function Set so as to encourage the selection of interesting genetic material while promoting diversity and test case feasibility. This methodology allows mitigating the negative effects of including a large number of entries into the Function Set; also, it allows a higher degree of freedom when defining it, by minimizing the impact of redundant, irrelevant or erroneous choices. The experimental studies implemented indicate a considerable improvement in the algorithm's efficiency when compared to its static counterpart, while introducing a negligible overhead. Future work involves developing strategies for dynamically adapting other aspects of our EA, such as the operators' selection probabilities.

6. ACKNOWLEDGEMENTS

This work has been partially supported by project TIN2007-68083-C02 (Spanish Ministry of Education and Culture, Non-Hierarchical Network Evolutionary System Project – NoHNES).

7. REFERENCES

[1] P. J. Angeline. Adaptive and self-adaptive evolutionary computations. In *Computational Intelligence: A Dynamic Systems Perspective*, pages 152–163. IEEE Press, 1995.

[2] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3:124–141, 1999.

[3] S. Luke. Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation*, 4(3):274–283, Sept. 2000.

[4] J. C. B. Ribeiro. Search-based test case generation for object-oriented java software using strongly-typed genetic programming. In *GECCO '08: Proceedings of the 2008 GECCO Conference Companion on Genetic and Evolutionary Computation*, pages 1819–1822, New York, NY, USA, 7 2008. ACM.

[5] J. C. B. Ribeiro, M. Z. Rela, and F. F. de Vega. A strategy for evaluating feasible and unfeasible test cases for the evolutionary testing of object-oriented software. In *AST '08: Proceedings of the 3rd International Workshop on Automation of Software Test*, pages 85–92, New York, NY, USA, 2008. ACM.

View publication stats