Investigating Data-Flow Coverage of Classes Using Evolutionary Algorithms

Konstantinos Liaskos
University of Strathclyde, Glasgow
Livingstone Tower, 26 Richmond St.
Glasgow G1 1XH, UK
+44 (0)141 548 3590
konstantinos.liaskos@cis.strath.ac.uk

Marc Roper
University of Strathclyde, Glasgow
Livingstone Tower, 26 Richmond St.
Glasgow G1 1XH, UK
+44 (0)141 548 2956
marc.roper@cis.strath.ac.uk

Murray Wood
University of Strathclyde, Glasgow
Livingstone Tower, 26 Richmond St.
Glasgow G1 1XH, UK
+44 (0)141 548 3390
murray.wood@cis.strath.ac.uk

ABSTRACT
It is not unusual for a software development organization to expend 40% of total project effort on testing, which can be a very laborious and time-consuming process. Therefore, there is a big necessity for test automation. This paper describes an approach to automatically generate test-data for OO software exploiting a Genetic Algorithm (GA) to achieve high levels of data-flow (d-u) coverage. A proof-of-concept tool is presented. The experimental results from testing six Java classes helped us identify three categories of problematic test targets, and suggest that in the future full d-u coverage with a reasonable computational cost may be possible if we overcome these obstacles.

Categories and Subject Descriptors
D.2.5 [**Software Engineering**]: Testing and Debugging – *Testing tools*

General Terms
Verification

Keywords
Object-oriented testing, evolutionary algorithms, automatic test-case generation, data-flow coverage

1. INTRODUCTION
Search-based test case generation techniques are well researched for procedural software [1], but relatively little research has been done in the area of OO testing. Our aim is to build on existing work [2] and investigate the applicability of GAs to the problem of automated testing with d-u as the adopted coverage criterion.

2. THE PROBLEM
Data-flow coverage has the advantage of requiring very thorough testing, because the paths reported have direct relevance to the way the program handles data. In this context, data-flow coverage is particularly appropriate for OO testing, where classes combine data and behavior in the form of methods that access and manipulate that data. Two drawbacks of this coverage criterion are the high complexity and the fact that it does not include decision coverage.

3. OUR APPROACH
The class under test is analyzed prior to the execution of the GA in order to identify the test targets (d-u pairs). Our research focuses on the design of the fitness function (every statement of the d-u pair is considered a partial target, for which traditional fitness computation is utilized) and the implementation of the GA. JUnit is used to execute the resulting test cases, and various coverage tools are used to measure different types of coverage.

4. EXPERIMENTAL EVALUATION
Six classes (same as in [2]) from the standard Java library were used. Statement, branch and d-u coverage were measured and compared with each other. D-u coverage is relative high, but lower than branch or statement coverage in the case of classes with high cyclomatic complexity. Lower coverage, higher computational cost, and the large number of test cases can be explained by the fact that the number of test goals in our case is much higher compared to branch coverage. The analysis of our experimental results helped us identify three categories of problematic test targets: equivalent (i.e. d-u pairs that correspond to the same code structure), subsequent (i.e. the satisfaction of a test target is strongly related with another), and unrealistic.

5. CONCLUSIONS AND FUTURE WORK
The major contribution of this paper is the utilization of data-flow as the coverage criterion. Our experimental results suggest that in the future full d-u coverage with a reasonable computational cost may be possible if we manage to address the recorded problematic test targets. To this end, the first step is to examine the use of a modified fitness function that incorporates information from both partial targets of a d-u pair simultaneously, and then investigate the exploitation of an AIS algorithm with a path-oriented affinity computation mechanism to better guide the search.

6. REFERENCES
[1] McMinn, P. Search-based Software Test Data Generation: a Survey. *Journal of Software Testing, Verifications, and Reliability*, vol. 14, no. 2, pp. 105-156, June 2004.

[2] Tonella, P. Evolutionary Testing of Classes. *In Proceedings of the ACM SIGSOFT International Symposium of Software Testing and Analysis*, Boston, MA, July 2004, pp. 119-128.