TestFul: using a Hybrid Evolutionary Algorithm for Testing Stateful Systems

Matteo Miraz, Pier Luca Lanzi, Luciano Baresi[*]
Dipartimento di Elettronica e Informazione – Politecnico di Milano
piazza Leonardo da Vinci, 32
20133 - Milano, Italy
{miraz,lanzi,baresi}@elet.polimi.it

ABSTRACT

This paper introduces *TestFul*, a framework for testing stateful systems and focuses on object-oriented software. *TestFul* employs a hybrid multi-objective evolutionary algorithm, to explore the space of feasible tests efficiently, and novel quality metrics, based on both *def-use* pairs and *behavioral* coverage, to judge the quality of tests.

We compare our framework against random testing by considering the level of coverage, the size of generated tests, and the time required to generate the tests. Our preliminary results show the validity of the approach: *TestFul* outperforms random testing in most of the cases.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

Keywords

Test Generation, Evolutionary Algorithms, Stateful Systems, Object-Oriented Paradigm, Multi-Objective Optimization

General Terms

Algorithms, Design, Experimentation

1. INTRODUCTION

Writing tests manually is a tedious, expensive, and commonly underestimated activity. This is why the research community proposed several ways to generate tests automatically. Many promising works reduce the test generation to a search problem [7]. Among them, random testing [3] performs a random sequence of invocations on the system under test. Despite its simplicity, it can be as effective as other traditional approaches, revealing errors also in widespread systems [1]. Other approaches guide the search process towards the satisfaction of a goal. McMinn [7] organized them in coverage-oriented and structure-oriented. The former rewards more tests able to reach higher coverage values on the system under test. Watkins [10] targets full path coverage, but his proposal performs poorly since the search process is not guided towards new uncovered paths. The latter (e.g., [4]) prescribes the analysis of a system to identify the *target* set, which contains the structural elements that selected tests must reach (given the chosen coverage criterion). For example, full statement coverage would build a target set with all the basic blocks of the system. Then, structure-oriented approaches use a search algorithm to generate a test able to reach each target element separately. Tonella [8] applies this approach to object-oriented systems and noticed that to exercise a particular functionality of a class, the system must be in a particular state. For this reason, his proposal operates on sequences of operations able to prepare the state of objects and reach the desired execution point.

TestFul tries to improve these algorithms by considering that once a particular state is reached, it often enables several different features at the same time. There is no need to restart from scratch every time, but the current state must be retained as a valuable asset.

[*]This research has been partially funded by the European Commission, Programme IDEAS-ERC, Project 227977-SMScom.

2. TESTFUL

Structure-oriented approaches follow the *divide and conqueror* paradigm, and handle each structural element separately from the others, even when tightly related. This solution imposes unnecessary effort, especially when targeting stateful systems, whose current state plays a significant role in exercising the different features.

TestFul solves this issue by employing an evolutionary algorithm to generate tests for the entire class, without a-priori identifying interesting elements. *TestFul* works on a variable-length sequence of operations on the class under test by exploiting the variable-length version of the one-point crossover. It uses a multi-objective fitness function, that drives the SPEA2 [11] evolutionary process by means of the coverage of *def-use* pairs, the compactness of candidate results, and the coverage of behavioral aspects.

The coverage of *def-use* pairs is a common way to judge tests for object-oriented systems since it is able to spot dependencies on fields accessed in different parts of the class. As for compactness, given two tests with the same ability to exercise the system, we simply favor the shorter one. Finally, we use behavior inference techniques, like ADABU [2], to extract finite state machines that represent the behaviors of the classes under test and that tests should be able to exercise. We measure the *behavioral coverage* by applying the *all-transition* heuristic that counts the number of covered edges of the state machine.

TestFul uses this information to evolve intermediate tests by recombining their valuable building blocks. Moreover, since we target stateful systems, the fitness function uses a behavioral coverage to reward more the tests able to manage properly the state of objects.

Albeit preliminary evaluations made using this configuration of *TestFul* outperformed random test in most cases, our proposal foresees the creation of a better initial population (seeding) and also the ability to reach uncovered structural elements. Instead of starting from a randomly-generated initial population, we extract initial individuals from available tests, if they exist, or by monitoring the execution of the program. This way, the initial population contains more useful building blocks, and allows *TestFul* to obtain better results in less time.

The main problem of coverage-oriented approaches is the lack of guidance. To solve this issue, *TestFul* uses a hybrid technique able to analyze the coverage achieved by the current population, detect uncovered structural elements, and enrich individuals to cover at least one of them. Since this operation is performed during the evolution, it is possible to make it as smart as possible and select both the candidate test and the uncovered element that require less effort.

3. PRELIMINARY RESULTS

We compared our approach against random testing [5], probably the most widely known approach for the automatic generation of tests, characterized by a blind random search of the solution space. To have a fair comparison, we used the "vanilla" version of *TestFul*, which starts from a random initial population and does not exploit hybrid techniques.

We applied both *TestFul* and random testing on a class implementing a state machine, inspired by a method present in the VIM text editor [9], and previously analyzed in [6].

We performed ten runs of six hours for each method, and the comparison is made on the average values. The results (Figure 1) are extremely encouraging: *TestFul* was able to achieve the full coverage (40 edges of the behavioral model and 100 def-use pairs) in 2 hours. Random testing after 6 hours was only able to cover 14 edges in the behavioral model (35%) and 26 def-use pairs (26%).

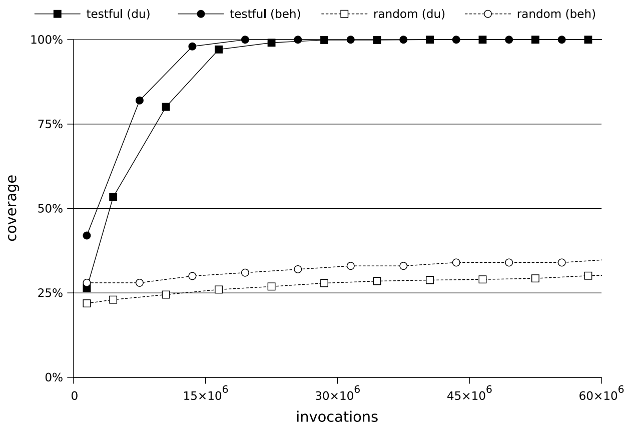

Figure 1: State machine: behavioral (circles) and def-use pairs (squares) coverage achieved by random testing (empty symbols) and *TestFul* (solid symbols). Curves are averages over ten run.

Even more interesting, the test generated by *TestFul* is extremely compact and consists of less than 500 invocations.

From this and other experiments, *TestFul* appears to explore the search space less than random testing, being able to put objects in complex states, and therefore it typically outperforms it. However, in some cases both random testing and *TestFul* failed to reach optimal coverage for classes with very complex states: for random testing, this is caused by an inner lack of guidance while, for *TestFul*, this appears to be due to a too coarse guidance —a result that confirms the importance of applying a hybrid evolutionary strategy.

4. CONCLUSIONS AND FUTURE WORK

This paper introduces *TestFul*, a novel framework for the automatic test generation for stateful systems based on a hybrid multi-objective evolutionary optimization. Compared to the state of the art in search-based test generation, *TestFul* recognizes and reuses useful state configurations to exercise different features, which is a way to save in resources.

Preliminary evaluations suggest that *TestFul* explores the search space less than random testing by exploring the space closer to current solutions. This allows *TestFul* to put objects in useful states and therefore it typically outperforms random testing on classes with complex states.

As for future work, we will keep working on the search strategy and on behavioral inference techniques by exploiting the domain knowledge. In parallel, we would like to use *TestFul* on other case studies and on other classes of stateful systems.

5. REFERENCES

[1] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer. Experimental assessment of random testing for object-oriented software. In *ISSTA*, pages 84–94, 2007.

[2] V. Dallmeier, C. Lindig, A. Wasylkowski, and A. Zeller. Mining object behavior with ADABU. In *WODA '06*, pages 17–24, New York, 2006. ACM.

[3] R. Hamlet. Random testing. In *Encyclopedia of Software Engineering*, pages 970–978. Wiley, 1994.

[4] B. Korel. Automated software test data generation. *IEEE Trans. Software Eng.*, 16(8):870–879, 1990.

[5] A. Leitner, I. Ciupa, M. Oriol, B. Meyer, and A. Fiva. Contract driven development = test driven development - writing test cases. In *ESEC/SIGSOFT FSE*, pages 425–434, 2007.

[6] R. Majumdar and K. Sen. Hybrid concolic testing. In *ICSE*, pages 416–426. IEEE Computer Society, 2007.

[7] P. McMinn. Search-based software test data generation: a survey. *Softw. Test., Verif. Reliab.*, 14(2):105–156, 2004.

[8] P. Tonella. Evolutionary testing of classes. In *ISSTA*, pages 119–128. ACM, 2004.

[9] Vim. `http://www.vim.org`.

[10] A. Watkins. The automatic generation of test data using genetic algorithms. In *Proceedings of the Fourth Software Quality Converence*, pages 300–309, 1995.

[11] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical report, ETH Zurich, May 2001.