Evosuite at the SBST 2021 Tool Competition

Vogl, Sebastian; Schweikl, Sebastian; Fraser, Gordon; Arcuri, Andrea; Campos, José; Panichella, Annibale

DOI
10.1109/SBST52555.2021.00012

Publication date
2021

Document Version
Final published version

Published in
2021 IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST)

Citation (APA)
https://doi.org/10.1109/SBST52555.2021.00012

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.
EVO SUITE at the SBST 2021 Tool Competition

Sebastian Vogl*, Sebastian Schweikl*, Gordon Fraser†, Andrea Arcuri‡, Jose Campos§, and Annibale Panichella§

*University of Passau, Germany
{sebastian.vogl,sebastian.schweikl,gordon.fraser}@uni-passau.de
†Kristiania University College and Oslo Metropolitan University, Norway
andrea.arcuri@kristiania.no
‡University of Lisbon, Portugal
jcmcampos@fc.ul.pt
§Delft University of Technology, Netherlands
A.Panichella@tudelft.nl

Abstract—EVO SUITE is a search-based unit test generation tool for Java. This paper summarises the results and experiences of EVO SUITE’s participation at the ninth unit testing competition at SBST 2021, where EVO SUITE achieved the highest overall score.

I. INTRODUCTION

The annual unit test generation competition aims to drive and evaluate progress on unit test generation tools. Every year, a benchmark of Java classes is selected, and different test generation tools are evaluated in terms of the code coverage and mutation scores they can achieve on these classes. In the 2021 edition of the competition, six tools were applied on a set of 98 classes. Details about the procedure of the competition, the technical framework, and the benchmark classes can be found in the competition report [9]. EVO SUITE achieved an overall score of 292.05, which was the highest among the competing and baseline tools.

II. ABOUT EVO SUITE

EVO SUITE [2] uses evolutionary search to automatically generate test suites for Java classes. It takes as input the name of a target class as well as the classpath describing where the compiled bytecode of the class as well as its dependencies are located. A basic static analysis extracts information about the relevant classes as well as their constructors, methods, and fields. The bytecode is instrumented while classes are loaded, such that EVO SUITE can produce execution traces for test executions, and to avoid test flakiness by replacing non-deterministic calls with deterministic, mocked versions. EVO SUITE then applies meta-heuristic search algorithms to automatically produce a set of JUnit test cases aimed at maximising code coverage.

Originally, EVO SUITE was implemented to optimise entire test suites with respect to their overall code coverage [4]. The latest search algorithm, which is now used by default, is the Dynamic Many-Objective Sorting Algorithm (DynaMOSA) search algorithm [7] which operates at the test case level. The genetic encoding of test cases in EVO SUITE consists of a target class as well as calls on constructors or methods. Standard evolutionary search operators (e.g., selection, crossover, mutation) are adapted for this representation. EVO SUITE supports multiple different coverage criteria that can be optimised simultaneously. The core fitness functions in EVO SUITE are based on traditional heuristics for code coverage, such as the branch distance and the approach level (see [4] for more details). Further fitness functions are based on mutation testing [5] as well as other basic criteria [10].

To improve the readability of the generated tests and to avoid test smells [8], EVO SUITE applies post-processing once the available search budget has been exhausted [2, 3]. In particular, minimisation is used to remove redundant tests and statements, and a minimised set of assertions is added using mutation analysis [6]. Finally, EVO SUITE compiles and executes each test individually to avoid compilation errors (which may be the result of bugs in EVO SUITE) or flakiness caused by non-determinism in the class under test.

III. COMPETITION SETUP

The most recent version of EVO SUITE (1.1.0) was used in the competition. As of this version, the default search algorithm in EVO SUITE is DynaMOSA [7]. No new features were added in this version, but support for Java 9+ was added, several bugs were fixed, and a major refactoring was applied to improve the code quality with respect to the use of Java generics internally. Several smaller bugfixes were applied in the run-up to the competition, which are now included in the current development version on GitHub.

The configuration of EVO SUITE used in the 2021 competition is identical to the configuration used in prior years, and largely based on its tuned default values [1] and default coverage criteria [10]. Similar to prior competitions, the test minimisation step was omitted as the competition score does not reward test minimality, and the minimisation takes substantial computational effort. Similarly, we included all regression assertions rather than filtering them with mutation analysis to save time. Same as in previous competitions, we allocated 50% of the overall time set by the competition organisers for the search, and distributed the other 50% equally to the remaining phases (e.g., assertion generation, compile-check, flakiness-check).
IV. RESULTS

With an overall score of 292.05, EVOSUITE achieved the highest score of all tools in the competition. As described in the competition report [9], the score is calculated on the subset of 25 classes for which all tools produced tests in some of the runs. Figure 1 summarises the results of EVOSUITE in terms of the mean coverage for the 25 classes used in the competition. While the coverage increased from the 30 seconds to the 120 seconds search budget (mean line coverage of 65.7% vs. 77.1%, respectively), the additional run of 300 seconds actually shows a slight decrease (mean line coverage of 73.9%). The likely reason for this result is that EVOSUITE crashed more frequently when given more time. For example, for 76 runs of EVOSUITE on the 25 classes no test cases were generated. In most cases we found that the JVM process running EVOSUITE on the competition machine crashed with an OutOfMemoryException, which possibly could have been avoided with a more generous allocation of memory.

The coverage ratios are generally higher than the mutation scores; while this is largely because it is simply more difficult to achieve a high mutation score, the mutation score is also negatively affected by classes with flakiness or failing tests (in which case the mutation score reported is 0% by definition). Furthermore, in many cases the execution with the PIT mutation tool reported an error during class loading, which may be caused by configuration errors or clashes between EVOSUITE’s and PIT’s bytecode instrumentation.

A final observation is that all 25 Java classes considered in the competition results were taken from the Guava project, which is notorious for its complex use of Java Generics—which is a feature EVOSUITE still struggles to handle effectively. In particular, in 99 of all runs on the 25 competition classes EVOSUITE did not manage to move beyond the initial generation, which tends to happen specifically when EVOSUITE struggles to correctly instantiate complex, nested generic types. This is currently a focus point of structural improvements and refactorings in EVOSUITE’s codebase.

V. CONCLUSIONS

This paper reports on the participation of the EVOSUITE test generation tool in the ninth SBST Java Unit Testing Tool Contest. On average, EVOSUITE achieved 71% branch coverage, 77% line coverage, and a mutation score of 53%, using a search budget of 120 seconds on the 25 classes considered for the competition. Overall, this results in a score of 292.05, which is the highest score of all tools in the competition.

To learn more about EVOSUITE, visit our Web site:

http://www.evosuite.org

Acknowledgments: Many thanks to all the contributors to EVOSUITE. This project has been supported by EPSRC project EP/N023978/2.

REFERENCES