EvoSuiteDSE at the SBST 2021 Tool Competition

Ignacio Manuel Lebrero Rial
Department of Computer Science
University of Buenos Aires, Argentina
ilebrero@dc.uba.ar

Juan P. Galeotti
Department of Computer Science
University of Buenos Aires, Argentina
jgaleotti@dc.uba.ar

***Abstract*—EvoSuite is a search-based tool with a dynamic symbolic execution module (EvosuiteDSE) that automatically generates executable unit tests for java code (JUnit tests). This paper summarizes the results and experiences of EvoSuiteDSE's participation at the ninth unit testing competition at SBST 2021, where EvoSuiteDSE achieve an overall score of 47.14 in its first participation on the competition.**

I. Introduction

Automated unit test generation can support developers and testers, produce regression test suites, and is an enabler for dynamic program analyses. The annual unit test generation contest aims to foster research and development of automated unit test generators. This paper describes the results obtained by the EvoSuite's DSE module's [1] test generation tool in this competition. EvosuiteDSE uses dynamic symbolic execution to systematically explore different execution paths of a given program by recollecting constraints about an execution, modifying them to express a new not explored execution path and generating new inputs using a SMT-solver to solve the new path constraints. In the 9th instance of the competition at the International Workshop on Search-Based Software Testing (SBST) 2021, Evosuite achieved and overall score of 47.14, which was the lowest among the competition and baseline tools [2]. This paper describes the results obtained by EvoSuiteDSE in this competition.

TABLE I
Classification of the EvoSuiteDSE unit test generation tool

Prerequisites	
Static or dynamic	Dynamic testing at the Java class level
Software Type	Java classes
Lifecycle phase	Unit testing for Java programs
Environment	All Java development environments
Knowledge required	JUnit unit testing for Java
Experience required	Basic unit testing knowledge
Input and Output of the tool	
Input	Bytecode of the target class and dependencies
Output	JUnit 4 test cases
Operation	
Interaction	Through the command line
User guidance	Manual verification of assertions for functional faults
Source of information	http://www.evosuite.org
Maturity	Mature research prototype, under development
Technology behind the tool	Dynamic Symbolic Execution / Generational Search
Obtaining the tool and information	
License	Lesser GPL V.3
Cost	Open source
Support	None

II. EvoSuiteDSE

Evosuite is a Java search-based unit test generation tool [3] which contains a dynamic symbolic execution module that can be used as both an enhancer for search-based generation [4] or for running a pure dynamic symbolic execution approach. Table 1 summarizes the features of EvoSuite in the standard format of the SBST unit testing competition: Given just the Java classpath containing all compiled dependencies and the name of a class under test, EvoSuiteDSE automatically generates a set of JUnit test cases aimed at maximizing code coverage. EvoSuiteDSE can only be used on the command line at the moment.

Throughout its development, two exploration algorithms were evaluated, starting with a depth-first search when experimented on enhancing the search-based algorithm [4].

Recently, the DSE module was rewritten in order to decouple it from the existing evolutionary framework found in the tool and to enable the tool to use a pure concolic approach [1]. In this work, the exploration algorithm was changed to a generational search one (In the same fashion as SAGE[5] but using incremental branch coverage instead of incremental block coverage) and two encoding of uni-dimensional arrays support were added to the engine. The first array encoding uses arrays theory while the second one, which was named *lazy variables*, models each entry in the array as an independent symbolic element using uninterpreted functions. Memory wise, the *lazy variable* implementation should perform better, as symbolic expressions are created on demand as the array is accessed and single variables should be easier to solve than arrays theory expressions (this approach results in the loose of completeness due to the fact that the array access index's symbolic value is concretized when used). Currently, the engine can reason about integers (int, long, short, boolean, byte and char), reals (float and double), partially about strings and arrays using the Z3 or CVC4 SMT-solvers. In addition, the engine partially supports a few dynamic language features found in recent versions of the JDK that use the `Invokedynamic` bytecode instruction, such as lambdas and string concatenation.

After the search has used up the available search budget (or alternatively, has achieved 100% coverage of all coverage objectives), EvoSuite applies various post-search optimizations aimed to improve the readability of the generated tests [3], [6], such as minimization and addition of test assertions using mutation analysis[7]. It also checks all generated tests for compile errors (which may be the result of bugs in EvoSuite) or flakiness caused by non-determinism in the class under test, not covered by EvoSuite's extensive instrumentation.

As EvoSuiteDSE is in an early development stage, it has been preliminary evaluated [1] and several mayor features remain future work, such as complex objects support.

III. Tool setup

The configuration for EvoSuiteDSE for the 2021 competition is largely based on Evosuite's default values since these have been tuned extensively for non-DSE purposes [8]. The exploration algorithm used was generational search [5] and the arrays implementation was *lazy variables*. On the solver side, CVC4 was used.

The test minimization step used by EvoSuite is computationally expensive, but it was enabled in the competition because minimized tests are less likely to expose flakiness. However, we aimed to reduce the post-processing time by including all regression assertions rather than filtering them with mutation analysis [7]. While this makes test cases less readable and potentially more brittle with respect to future changes in the software under test, neither of these aspects is evaluated as part of the SBST contest.

EvoSuite uses different phases (e.g., initialization, search, minimization, assertion generation, compilation check, removal of flaky tests). Like in previous competitions where the search-based configuration was presented, we allocated 50% of the overall time set by the competition organizers for the search, and distributed the other 50% equally to the remaining phases.

IV. Benchmark results

Table 2 summarizes the results achieved by EvoSuiteDSE on the competition classes for the *GUAVA* project and search budgets. In general, EvoSuiteDSE did not perform well compared to the other tools. This is due to the fact that in its current release, EvoSuiteDSE can only create test cases that directly invoke the static methods of the target classes. And whenever a non primitive data type or string is required, the only value passed by EvoSuiteDSE is null. Interestingly, all search budgets (30s, 120s and 300s) performed exactly the same (except for *com.google.common.math.Quantiles* which resulted in 0 line, branch and mutation coverage when running for 30s), this may suggest that the exploration space reachable by the tool is relatively small given the current elements that the engine can reason about.

TABLE II
Detailed results of EvoSuiteDSE on the SBST benchmark's GUAVA project classes. All classes have an implicit prefix "com.google.common"

Java Class	Coverage(%)		Mutation Score(%)
	Line	Branch	
base.CharMatcher	10.11	0	9.04
base.FinalizableReferenceQueue	0	0	0
base.Stopwatch	21.15	4.16	12.76
cache.CacheStats	0	0	0
collect.MapMaker	0	0	0
collect.MinMaxPriorityQueue	0	0	0
collect.Ordering	0	0	0
collect.Range	9.09	5.10	7.43
collect.TreeRangeMap	3.90	0	1.58
escape.Escapers	21.42	0	18.18
graph.EndpointPair	6.25	0	0
hash.HashCode	44.18	53.57	46.96
hash.Hashing	39.02	23.33	42.5
io.ByteSource	5.68	0	7.69
io.ByteStreams	14.28	8.33	16.12
io.Files	27.55	23.07	32.5
io.MoreFiles	3.84	0	4.05
math.PairedStatsAccumulator	0	0	0
math.Quantiles	6.18	2.94	7.14
net.MediaType	58.45	2.70	11.84
primitives.Shorts	0	0	0
reflect.TypeResolver	0	0	0
reflect.TypeToken	0	0	0
util.concurrent.MoreExecutors	8.02	5.55	14.92
util.concurrent.Striped	23.33	20	24.24

V. Conclusion

This paper reports on the participation of the EvoSuiteDSE test generation tool in the 9th SBST Java Unit Testing Tool Contest. Given it's premature state and with an overall score of 47.14, EvoSuiteDSE seemed to perform the worst on the CUTs selected for this year. In light of the benchmark results, implementing complex objects support seems to be the next step for the tool as a significant performance improvement is suggested.

References

[1] I. M. Lebrero Rial, "Ejecución simbólica dinámica en evosuite: Estudio e implementación. master's thesis. department of computer science, faculty of exact and natural sciences, university of buenos aires," 2020.

[2] S. Panichella, A. Gambi, F. Zampetti, and V. Riccio, "Sbst tool competition 2021," in *International Conference on Software Engineering, Workshops, Madrid, Spain, 2021*. ACM, 2021.

[3] G. Fraser and A. Arcuri, "Evosuite: Automatic test suite generation for object-oriented software," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 416–419. [Online]. Available: https://doi.org/10.1145/2025113.2025179

[4] J. P. Galeotti, G. Fraser, and A. Arcuri, "Improving search-based test suite generation with dynamic symbolic execution," in *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, 2013, pp. 360–369.

[5] P. Godefroid, M. Y. Levin, and D. Molnar, "Sage: Whitebox fuzzing for security testing," *Queue*, vol. 10, no. 1, p. 20–27, Jan. 2012. [Online]. Available: https://doi.org/10.1145/2090147.2094081

[6] G. Fraser and A. Arcuri, "Evosuite: On the challenges of test case generation in the real world," *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pp. 362–369, 2013.

[7] G. Fraser and A. Zeller, "Mutation-driven generation of oracles and unit tests," *IEEE Transactions on Software Engineering*, vol. 38, no. 2, pp. 278–292, Mar. 2012.

[8] A. Arcuri and G. Fraser, "Parameter tuning or default values? an empirical investigation in search-based software engineering," *Empirical Software Engineering*, vol. 18, 06 2013.