EvoSuite at the SBST 2019 Tool Competition

Campos, José; Panichella, Annibale; Fraser, Gordon

DOI
10.1109/SBST.2019.00017

Publication date
2019

Document Version
Final published version

Published in
2019 IEEE/ACM 12th International Workshop on Search-Based Software Testing (SBST)

EvoSuite at the SBST 2019 Tool Competition

José Campos
University of Washington
Seattle, USA
jmcampos@uw.edu

Annibale Panichella
Delft University of Technology
Delft, Netherlands
a.panichella@tudelft.nl

Gordon Fraser
Chair of Software Engineering II,
University of Passau
Passau, Germany
gordon.fraser@uni-passau.de

ABSTRACT

EvoSuite is a search-based tool that automatically generates executable unit tests for Java code (JUnit tests). This paper summarises the results and experiences of EvoSuite's participation at the seventh unit testing competition at SBST 2019, where EvoSuite achieved the highest overall score (255.43 points) for the sixth time in seven editions of the competition.

ACM Reference Format:
José Campos, Annibale Panichella, and Gordon Fraser. 2019. EvoSuite at the SBST 2019 Tool Competition. In *SBST'19: IEEE/ACM 12th International Workshop on Search-Based Software Testing, May 27, 2019, Montreal, Canada.* ACM, New York, NY, USA, 4 pages.

1 INTRODUCTION

The annual unit test generation competition aims to drive and evaluate progress on unit test generation tools. In the 7th instance of the competition at the International Workshop on Search-Based Software Testing (SBST) 2019, several JUnit test generation tools competed on a set of 38 open-source Java classes. This paper describes the results obtained by the EvoSuite test generation tool [8] in this competition. Details about the procedure of the competition, the technical framework, and the benchmark classes can be found in [21]. In this competition, EvoSuite achieved an overall score of 255.43, which was the highest among the competing and baseline tools.

2 ABOUT EVOSUITE

EvoSuite [8] is a search-based tool [12] that uses a genetic algorithm to automatically generate test suites for Java classes. Given the name of a target class and the full Java classpath (i.e., where to find the compiled bytecode of the class under test and all its dependencies), EvoSuite automatically produces a set of JUnit test cases aimed at maximizing code coverage. EvoSuite can be used on the command line, or through plugins for popular development tools such as IntelliJ, Eclipse, or Maven [2].

For this edition of the competition, EvoSuite was configured with *Dynamic Many-Objective Sorting Algorithm* (DynaMOSA) as a search algorithm. DynMOSA [23, 24] is a recently developed many-objective algorithm that incrementally optimizes multiple coverage criteria at the same time. More specifically, different testing coverage requirements (e.g., branches) are treated as distinct, contrasting search objectives in a many-objective paradigm. Search objectives are computed using standard heuristics for code coverage, such as the branch distance and the approach level (see [12] for more details). Coverage requirements are prioritized during the search according to their structural dependencies in the control dependency graph. The search is initialized with the coverage requirements (e.g., branches) positioned higher in the hierarchy; then, the remaining requirements are incrementally inserted in later generations when their dominator requirements are covered.

Table 1: Classification of the EvoSuite 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, and plugins for IntelliJ, Maven and Eclipse
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	Search-based testing / many-objective optimization
Obtaining the tool and information	
License	Lesser GPL V.3
Cost	Open source
Support	None
Does there exist empirical evidence about	
Effectiveness and Scalability	See [12, 13]

DynaMOSA evolves test cases [23], which correspond to the chromosomes for the search. Each test case consists of variable-length sequences of Java statements (e.g., primitive statements and calls on the class under test). A population of randomly generated test cases is evolved using search operators (e.g., selection, crossover and mutation) to iteratively produce test cases that are closer to satisfy the target coverage requirements. The final test suite is formed by the shorter test cases encountered during the search, and that satisfy the coverage requirements. These tests are stored in an external data structure, called *archive* [22, 23].

EvoSuite can be configured to satisfy for multiple coverage criteria at the same time, and the default configuration combines branch coverage with mutation testing [14] and other basic criteria [25]. Once the search is completed, EvoSuite applies various post-search optimizations aimed to improve the readability of the generated tests. For example, tests are minimized and test assertions are selected using mutation analysis [18]. For more details on post-process optimization we refer to [8, 9].

In the past, the effectiveness of EvoSuite has been evaluated on open source as well as industrial software in terms of code coverage [7, 13, 23, 27], fault finding effectiveness [1, 29], and effects on developer productivity [17, 26] and software maintenance [30]. EvoSuite has a longstanding record of success at the unit testing tool competition, having ranked second in the third edition of the competition [15] and first in all the other editions [10, 11, 16, 19, 20].

3 COMPETITION SETUP

The configuration of EvoSuite for the 2019 competition is largely based on its default values since these have been tuned extensively [4]. We used the default set of coverage criteria [25] (e.g., line coverage, branch coverage, branch coverage by direct method invocations, weak mutation testing, output coverage, exception coverage). EvoSuite uses an archive of solutions [27] to keep the search focused on uncovered goals, iteratively discarding covered goals and storing the tests that covered them. After a certain percentage of the search budget has passed and with a certain probability, EvoSuite starts using mock objects (relying on Mockito) instead of actual class instances [3] for dependencies; only branches that cannot be covered without mocks lead to tests with mock objects. As in the previous instance of the competition, we continue to use frequency-based weighted constants for seeding [28] and support Java Enterprise Edition features [5]. EvoSuite is actively maintained, therefore several bug fixes and minor improvements have been applied since the last instance of the competition, in particular regarging non-determinism and flaky tests [6], as well as new types of test assertions (e.g., related to arrays and standard container classes).

Like in previous instances of the competition, we enabled the post-processing step of test minimization—not for efficiency reasons, but because minimized tests are less likely to break. To reduce the overall time of test generation we included all assertions rather than filtering them with mutation analysis [18], which is computationally expensive. The use of all assertions has a negative impact on readability, but this is not evaluated as part of the SBST contest.

Four time budgets were used to call each tool: 10, 60, 120 and 240 seconds. We used the same strategy used in previous competition (e.g., [16]) to distribute the overall time budget onto the different phases of EvoSuite (e.g., initialisation, search, minimization, assertion generation, compilation check, removal of flaky tests). That is, 50% of the time was allocated to the search, and the rest was distributed equally to the remaining phases.

4 BENCHMARK RESULTS

On the whole, the performance of EvoSuite was in line with previous results, although EvoSuite seemed to fail on a notably large number of classes. In particular, EvoSuite failed to produce any test suites for benchmarks DUBBO-2, SPOON-169, SPOON-25, and WEBMAGIC-4, and generally struggled with most AUTHZFORCE benchmarks (highlighted in grey in Table 2).

A closer look at the benchmarks DUBBO-2 and WEBMAGIC-4 reveals that there were missing dependencies in the competition setup [21] (similar to cases in previous years [20]), which explains why EvoSuite was incapable of generating any test: When a dependency of the class under test is missing, then EvoSuite intentionally aborts with an error to inform the user of the configuration error. While this is desirable behavior during regular usage, arguably in the scope of the competition, EvoSuite could try to ignore such errors and try to generate tests nevertheless, as it might still be possible to cover code even with some missing dependencies. For the benchmark FASTJSON-3 and a 10s time budget, EvoSuite generated some test suites, but due to a bug in the competition infrastructure, no data was collected.

For the benchmarks SPOON-169 and SPOON-25, EvoSuite either failed to generate a compilable test suite or crashed due to a bug in DynaMOSA. Although EvoSuite's minimization could help at reducing the number of uncompilable / flaky test cases, for SPOON-169 all minimizations attempts ran out of time.

For the AUTHZFORCE benchmarks, although EvoSuite managed to generate some test suites, most executions for data collection ended with a failure. Specifically, for these classes, an exception was raised for test execution for *all* tools in the competition. However, while this exception was not problematic for other tools, it caused an inconsistent state of the JVM which led to the `@Before` setup methods of EvoSuite to fail (the `@Before` method was not able to configure EvoSuite's custom security manager because it was never correctly removed). As a consequence, all test executions failed. We were not able to reproduce the error in the competition infrastructure, and thus cannot conjecture about the causes. The same problem also occurred for SPOON-253. While in principle it would be possible to extend EvoSuite to try to ignore such errors and keep on executing tests, as explained it is EvoSuite's philosophy that the user should be made aware of configuration errors, and hence such an extension would only serve to tune EvoSuite for the competition.

In general, EvoSuite often struggles with a time budget as low as 10 seconds. For example, Table 2 reports "-" for benchmarks FASTJSON-1, FASTJSON-3, FESCAR-41, FESCAR-42, and SPOON-105. Surprisingly, for benchmarks SPOON-155, SPOON-16, SPOON-20, and SPOON-211, EvoSuite successfully generated test suites for small-time budgets, but it failed to do it for large time budgets. For all, EvoSuite crashed due to a bug in the DynaMOSA algorithm which had not been revealed by our previous experiments.

Table 2: Detailed results of EvoSuite on the SBST benchmark classes.

Benchmark	Java Class	Branch Coverage				Mutation Score			
		10s	60s	120s	240s	10s	60s	120s	240s
AUTHZFORCE-1	org.ow2.authzforce.core.pdp.impl.PdpBean	0.0%	16.7%	0.0%	0.0%	0.0%	14.4%	0.0%	0.0%
AUTHZFORCE-11	org.ow2.authzforce.core.pdp.impl.func.LogicalNOfFunction	0.0%	0.0%	0.0%	0.0%	4.8%	4.8%	0.0%	0.0%
AUTHZFORCE-27	org.ow2.authzforce.core.pdp.impl.func.MapFunctionFactory	75.0%	0.0%	0.0%	0.0%	75.0%	0.0%	0.0%	0.0%
AUTHZFORCE-32	org.ow2.authzforce.core.pdp.impl.func.SubstringFunction	0.0%	0.0%	0.0%	0.0%	7.7%	0.0%	0.0%	0.0%
AUTHZFORCE-33	org.ow2.authzforce.core.pdp.impl.SchemaHandler	55.6%	0.0%	0.0%	0.0%	57.6%	0.0%	0.0%	0.0%
AUTHZFORCE-48	org.ow2.authzforce.core.pdp.impl.policy.PolicyVersions	0.0%	12.1%	34.8%	60.6%	0.0%	16.7%	37.3%	69.6%
AUTHZFORCE-5	org.ow2.authzforce.core.pdp.impl.CloseableAttributeProvider	20.5%	0.0%	0.0%	0.0%	27.4%	0.0%	0.0%	0.0%
AUTHZFORCE-52	org.ow2.authzforce.core.pdp.impl.PepActionExpression	91.7%	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%
AUTHZFORCE-63	org.ow2.authzforce.core.pdp.impl.combining.DPUnlessPDCombiningAlg	10.0%	3.3%	0.0%	0.0%	5.6%	1.9%	0.0%	0.0%
AUTHZFORCE-65	org.ow2.authzforce.core.pdp.impl.combining.FirstApplicableCombiningAlg	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
DUBBO-2	com.alibaba.dubbo.common.utils.ReflectUtils	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
FASTJSON-1	com.alibaba.fastjson.parser.JSONLexerBase	-	23.0%	24.9%	31.6%	-	7.7%	8.8%	6.5%
FASTJSON-3	com.alibaba.fastjson.parser.DefaultJSONParser	-	13.4%	20.5%	25.7%	-	9.4%	14.1%	19.1%
FESCAR-1	com.alibaba.fescar.core.protocol.transaction.BranchReportRequest	97.9%	95.8%	96.9%	99.0%	97.7%	97.7%	98.5%	100.0%
FESCAR-12	com.alibaba.fescar.core.rpc.netty.RpcServerHandler	12.5%	87.5%	87.5%	87.5%	50.0%	100.0%	100.0%	100.0%
FESCAR-18	com.alibaba.fescar.core.protocol.MergedWarpMessage	61.9%	48.8%	56.0%	42.9%	81.2%	60.9%	68.1%	51.4%
FESCAR-23	com.alibaba.fescar.core.protocol.MergeResultMessage	63.1%	78.6%	64.3%	83.3%	78.6%	97.0%	79.2%	98.8%
FESCAR-25	com.alibaba.fescar.core.rpc.netty.RmMessageListener	37.5%	62.5%	62.5%	50.0%	22.2%	22.2%	22.2%	18.5%
FESCAR-36	com.alibaba.fescar.core.protocol.RegisterRMRequest	100.0%	92.9%	80.1%	83.3%	100.0%	93.8%	81.2%	83.0%
FESCAR-37	com.alibaba.fescar.core.rpc.RpcContext	52.9%	89.2%	91.2%	91.2%	71.9%	96.4%	94.8%	95.8%
FESCAR-41	com.alibaba.fescar.core.rpc.netty.RmRpcClient	-	2.0%	2.0%	2.0%	-	2.4%	2.4%	2.4%
FESCAR-42	com.alibaba.fescar.core.rpc.DefaultServerMessageListenerImpl	-	16.1%	27.4%	27.9%	-	16.7%	27.8%	29.2%
FESCAR-7	com.alibaba.fescar.core.rpc.netty.MessageCodecHandler	37.8%	78.4%	82.8%	87.1%	37.9%	92.8%	95.2%	98.3%
OKIO-1	okio.Buffer	27.4%	57.0%	62.3%	67.9%	13.2%	20.2%	16.3%	18.5%
OKIO-4	okio.RealBufferedSource	19.2%	37.6%	51.3%	76.2%	13.1%	28.8%	40.1%	45.3%
SPOON-105	spoon.support.compiler.jdt.PositionBuilder	-	22.1%	16.4%	3.8%	-	27.7%	23.1%	5.4%
SPOON-155	spoon.reflect.visitor.filter.AllMethodsSameSignatureFunction	12.0%	4.2%	2.5%	0.0%	19.0%	10.7%	4.3%	5.0%
SPOON-16	spoon.reflect.path.CtElementPathBuilder	34.0%	34.3%	0.0%	2.0%	45.5%	43.9%	0.0%	4.0%
SPOON-169	spoon.reflect.visitor.ImportScannerImpl	-	0.0%	0.0%	0.0%	-	0.0%	0.0%	0.0%
SPOON-20	spoon.support.reflect.reference.CtLocalVariableReferenceImpl	33.3%	20.0%	0.0%	0.0%	40.3%	33.3%	0.0%	0.0%
SPOON-211	spoon.reflect.path.impl.CtRolePathElement	8.1%	19.2%	0.0%	5.1%	27.8%	36.8%	0.0%	17.7%
SPOON-25	spoon.pattern.internal.ValueConvertorImpl	-	-	0.0%	0.0%	-	-	0.0%	0.0%
SPOON-253	spoon.pattern.internal.parameter.MapParameterInfo	29.7%	50.0%	0.0%	0.0%	34.8%	50.0%	0.0%	0.0%
SPOON-32	spoon.MavenLauncher	12.5%	10.4%	10.4%	12.5%	6.7%	5.6%	5.6%	6.7%
SPOON-65	spoon.support.DefaultCoreFactory	9.0%	49.8%	20.0%	5.7%	1.2%	2.3%	1.2%	2.0%
WEBMAGIC-1	us.codecraft.webmagic.model.PageModelExtractor	28.0%	27.6%	28.8%	31.3%	50.4%	50.0%	51.3%	54.3%
WEBMAGIC-4	us.codecraft.webmagic.Page	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
ZXING-10	com.google.zxing.qrcode.encoder.Encoder	70.1%	82.3%	82.9%	84.0%	63.0%	76.1%	72.6%	72.8%
Average		17.3%	17.2%	14.8%	15.5%	19.9%	16.9%	13.9%	14.6%

Consequently, even though DynaMOSA is known to increase the coverage, EvoSuite achieves substantially in general, this increase was countered with some missing datapoints. As always, the competition was helpful in improving the robustness of the tool, by revealing this and other bugs.

5 CONCLUSIONS

This paper reports on the participation of the EvoSuite test generation tool in the 7th SBST Java Unit Testing Tool Contest. With an overall score of 255.43 points, EvoSuite achieved 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 funded by the EPSRC project "GREATEST" (EP/N023978/2), and by the National Research Fund, Luxembourg (FNR/P10/03).

REFERENCES

[1] M. M. Almasi, H. Hemmati, G. Fraser, A. Arcuri, and J. Benefelds, "An industrial evaluation of unit test generation: Finding real faults in a financial application," in *ACM/IEEE Int. Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 263–272.

[2] A. Arcuri, J. Campos, and G. Fraser, "Unit Test Generation During Software Development: EvoSuite Plugins for Maven, IntelliJ and Jenkins," in *IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE Computer Society, 2016, pp. 401–408.

[3] A. Arcuri, G. Fraser, and R. Just, "Private api access and functional mocking in automated unit test generation," in *IEEE Int. Conference on Software Testing, Verification and Validation (ICST)*, 2017, pp. 126–137.

[4] A. Arcuri and G. Fraser, "Parameter tuning or default values? An empirical investigation in search-based software engineering," *Empirical Software Engineering (EMSE)*, pp. 1–30, 2013.

[5] ——, "Java enterprise edition support in search-based junit test generation," in *Int. Symposium on Search Based Software Engineering*. Springer, 2016, pp. 3–17.

[6] A. Arcuri, G. Fraser, and J. P. Galeotti, "Automated unit test generation for classes with environment dependencies," in *IEEE/ACM Int. Conference on Automated Software Engineering (ASE)*. ACM, 2014, pp. 79–90.

[7] J. Campos, Y. Ge, N. Albunian, G. Fraser, M. Eler, and A. Arcuri, "An empirical evaluation of evolutionary algorithms for unit test suite generation," *Information and Software Technology*, vol. 104, pp. 207 – 235, 2018.

[8] G. Fraser and A. Arcuri, "EvoSuite: Automatic test suite generation for object-oriented software." in *ACM Symposium on the Foundations of Software Engineering (FSE)*, 2011, pp. 416–419.

[9] ——, "EvoSuite: On the challenges of test case generation in the real world (tool paper)," in *IEEE Int. Conference on Software Testing, Verification and Validation (ICST)*, 2013, pp. 362–369.

[10] ——, "EvoSuite at the SBST 2013 tool competition," in *Int. Workshop on Search-Based Software Testing (SBST)*, 2013, pp. 406–409.

[11] ——, "EvoSuite at the second unit testing tool competition." in *Fittest Workshop*. Springer, 2013, pp. 95–100.

[12] ——, "Whole test suite generation," *IEEE Transactions on Software Engineering*, vol. 39, no. 2, pp. 276–291, 2013.

[13] ——, "A large-scale evaluation of automated unit test generation using EvoSuite," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 24, no. 2, pp. 8:1–8:42, 2014.

[14] ——, "Achieving scalable mutation-based generation of whole test suites." *Empirical Software Engineering (EMSE)*, vol. 20, no. 3, pp. 783–812, 2015.

[15] ——, "EvoSuite at the SBST 2015 tool competition," in *Int. Workshop on Search-Based Software Testing (SBST)*. IEEE Press, 2015, pp. 25–27.

[16] ——, "EvoSuite at the SBST 2016 tool competition," in *Int. Workshop on Search-Based Software Testing (SBST)*. ACM, 2016, pp. 33–36.

[17] G. Fraser, M. Staats, P. McMinn, A. Arcuri, and F. Padberg, "Does automated unit test generation really help software testers? a controlled empirical study," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 24, no. 4, pp. 23:1–23:49, 2015.

[18] G. Fraser and A. Zeller, "Mutation-driven generation of unit tests and oracles," *IEEE Transactions on Software Engineering (TSE)*, vol. 28, no. 2, pp. 278–292, 2012.

[19] J. C. Gordon Fraser, José Miguel Rojas and A. Arcuri, "EvoSuite at the SBST 2017 tool competition," in *Int. Workshop on Search-Based Software Testing (SBST)*. IEEE Press, 2017, pp. 39–41.

[20] J. M. R. Gordon Fraser and A. Arcuri, "EvoSuite at the SBST 2018 tool competition," in *Int. Workshop on Search-Based Software Testing (SBST)*. IEEE Press, 2018, pp. 34–37.

[21] F. Kifetew, X. Devroey, and U. Rueda, "Java Unit Testing Tool Competition - Seventh Round," in *Int. Workshop on Search-Based Software Testing (SBST)*, 2019.

[22] A. Panichella, F. M. Kifetew, and P. Tonella, "Reformulating Branch Coverage as a Many-Objective Optimization Problem," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, April 2015, pp. 1–10.

[23] ——, "Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets," *IEEE Transactions on Software Engineering*, vol. 44, no. 2, pp. 122–158, Feb 2018.

[24] A. Panichella, F. M. Kifetew, and P. Tonella, "Incremental control dependency frontier exploration for many-criteria test case generation," in *International Symposium on Search Based Software Engineering*. Springer, 2018, pp. 309–324.

[25] J. M. Rojas, J. Campos, M. Vivanti, G. Fraser, and A. Arcuri, "Combining multiple coverage criteria in search-based unit test generation," in *Search-Based Software Engineering*. Springer, 2015, pp. 93–108.

[26] J. M. Rojas, G. Fraser, and A. Arcuri, "Automated unit test generation during software development: A controlled experiment and think-aloud observations," in *ACM Int. Symposium on Software Testing and Analysis (ISSTA)*. ACM, 2015, pp. 338–349.

[27] J. M. Rojas, M. Vivanti, A. Arcuri, and G. Fraser, "A Detailed Investigation of the Effectiveness of Whole Test Suite Generation," *Empirical Software Engineering (EMSE)*, vol. 22, no. 2, pp. 852–893, 2016.

[28] A. Sakti, G. Pesant, and Y.-G. Guéhéneuc, "Instance generator and problem representation to improve object oriented code coverage," *IEEE Transactions on Software Engineering*, vol. 41, no. 3, pp. 294–313, 2015.

[29] S. Shamshiri, R. Just, J. M. Rojas, G. Fraser, P. McMinn, and A. Arcuri, "Do automatically generated unit tests find real faults? an empirical study of effectiveness and challenges," in *IEEE/ACM Int. Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 201–211.

[30] S. Shamshiri, J. M. Rojas, J. P. Galeotti, N. Walkinshaw, and G. Fraser, "How do automatically generated unit tests influence software maintenance?" in *IEEE Int. Conference on Software Testing, Verification and Validation (ICST)*, 2018, to appear.