Extending and Experimenting with the EvoSuite Test Generation Tool

Aidin Rasti
arast040@uottawa.ca
University of Ottawa
Ottawa, Ontario, Canada

ABSTRACT

Search-based test generation techniques are one of the effective test generation approaches. EVOSUITE is one of the whole test suite generation tools that has implemented several types of Genetic Algorithms. We compare and show the results of generating tests for ten open-source Java projects with a combination of different selection and crossover functions in EVOSUITE. We have also extended EVOSUITE and implemented the two-point crossover function.

CCS CONCEPTS

• Software and its engineering → Search-based software engineering; **• General and reference → Empirical studies**; **• Computing methodologies → Genetic algorithms**.

KEYWORDS

genetic algorithms, test generation, search-based software engineering, empirical evaluation

ACM Reference Format:
Aidin Rasti. 2020. Extending and Experimenting with the EvoSuite Test Generation Tool. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Search-Based Software Engineering (SBSE) has numerous use cases [11, 12]. One of the exciting applications of SBSE is automated unit test generation. There are several works on this subject [3, 10, 14, 22]. One of the methods of automated unit test generation is using Genetic Algorithms. EVOSUITE [10] is a novel evolutionary-based approach for whole test suit generation. There are two approaches to search based test generation, one is targeting a single criterion at a time, for example, targeting each branch individually and generating data for that particular target branch. The other approach is generating and optimizing the whole test suit at the same time while maximizing coverage and minimizing test suite size. Instead of generating tests for each target goal one at a time, it generates and optimizes a whole test suite simultaneously.

EVOSUITE is a solution in the latter category, it is a whole test suite generation solution that can target multiple criteria at the same time. EVOSUITE starts with a random set of initial solutions and uses an evolutionary approach to improve solutions until the chosen criteria are satisfied. After generating solutions it will minimize the test suite size. EVOSUITE is developed in Java. Currently, it only generates tests in the JUnit format for Java programs. EVOSUITE works by instrumenting the compiled java bytecode and collecting metrics from the runtime. EVOSUITE is designed to write tests for object-oriented programs, therefore it can comfortably generate tests for a Class, which may require a sequence of method calls.

Genetic Algorithms are used to optimize and evolve a set of initial solutions toward better fitness values. Like all search-based techniques, Genetic Algorithms also have a solution representation and fitness function defined for the problem at hand. A typical flow of a Genetic Algorithm is as follows. The first step is the initialization, one way to do this is to randomly generate initial solutions. After calculating the fitness of each solution, a few of them are selected to create offsprings. This is called the Selection step. One method to do selection is to simply choose solutions with the best fitness values. After selecting parents, offsprings are created by applying a Crossover function. A popular way of doing crossover is to simply flip parents at a point and select before/after the point for each child. For example, suppose our solution is represented as an array of numbers and we have selected the middle of the array as the flipping point. From the beginning to the middle of the first parent and from the middle to the end of the second parent goes to the first child, and vice versa for the second child. After breeding a new generation of solutions mutations operators are applied to solutions. Mutation operators depend on the problem and solution representation. This process from fitness evaluation to mutations continues until termination criteria are met or appropriate solutions are found.

EVOSUITE is a highly configurable software, users can tweak its numerous parameters to achieve the best result. Authors of EVOSUITE have implemented several variations of Genetic Algorithms. In this report, we compare a variety of different crossover and selection functions for the standard genetic algorithm implemented in EVOSUITE. We used the standard Genetic Algorithm as the base and evaluated the performance of Uniform, Single point, and Two-point crossover functions in combination with Rank, Roulette Wheel, Tournament, and Best-k selection methods. Therefore we have eight combination of configurations to evaluate. EVOSUITE does not include the Two-point crossover function and we have implemented it. We have selected ten popular and open-source Java libraries from GitHub for our tests.

The structure of this report is as follows. First, in Section 2 we introduce other empirical works related to test generation with EVOSUITE then in Section 3 we describe our testing approach,

Conference'17, July 2017, Washington, DC, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

we explain our test configurations and procedure. Also, we briefly describe each of our test subjects. Finally, in Section 5 we show the result of our evaluations.

2 RELATED WORKS

One of the most comprehensive work that evaluates the performance of EVOSUITE is [4]. They have compared the performance of different test generation algorithms, such as Standard GA, Monotonic GA, DynaMOSA [18], MIO [1], etc, in EVOSUITE. Their tests include both single objective and multi-objective evaluations. They have concluded that DynaMOSA has the most covered targets in both single and multi-objective scenarios. The goal of their work is to compare different test generation algorithms, however, we want to compare the performance of different crossover and selection methods. We have used the Standard GA as our base algorithm and assessed the results under different configurations.

Another comprehensive work is the work of Arcuri and Fraser [2] which evaluates several hyperparameters of EVOSUITE and the effect of parameter tuning in test generation problems in general. They are more concerned with the effect of parameters like population size, crossover rate, probability configurations, search budget, etc.

Another work is [20] which compares the traditional single criterion approaches with multi-objective approaches available in EVOSUITE. Traditionally test generation solutions take a single criterion like branch coverage to generate unit tests. In newer whole test suite generation approaches several criteria like line coverage, branch coverage, etc can be used by algorithms like DynaMOSA [18] at the same time. These algorithms optimize for all of the targets at the same time. Their work concluded that there may be a small set of targets that may not be covered by multi-objective approaches.

One more empirical work is [19] that evaluates several seeding strategies for test generation problems. Seeding is the problem of providing values for method and class parameters when generating or modifying unit tests in search-based techniques. There are several solutions for seeding. For example, one way is to randomly use available constant values in the source code as input parameters. Another way is to observe passed parameters during runtime and use them during test generation. They have concluded that the choice of seeding strategy can impact the performance of search-based test generation solutions.

3 EXPERIMENTS

We have used the EVOSUITE test generation tool to compare the performance of several crossover and selection functions with the standard Genetic Algorithm. In total, our evaluations included 165 Java classes from various popular open-source projects and we have performed 12,240 tests.

3.1 Configurations

As mentioned earlier we have a total of eight configurations to compare. We have tested the performance of EVOSUITE by changing its crossover and selection functions. These two are denoted respectively by `crossover_function` and `selection_function` in EVOSUITE parameters. We have only used the standard GA, which is denoted by the configuration value `STANDARD_GA` in EVOSUITE. In the followed subsections we briefly describe each algorithm that we have tested.

3.1.1 Selection methods. Selection is one of the stages in a genetic algorithm. In this stage, a number of pairs of parents are selected to build offsprings. There are several selection methods available for use in genetic algorithms. We assessed four selection methods implemented in EVOSUITE, Rank Selection, Roulette Wheel, Tournament Selection, and Best-k.

- Roulette Wheel: In this method each individual is given a probability of selection equal to their fitness values, therefore, better individuals have a much higher chance of being selected. It is denoted by `ROULETTEWHEEL` value in EVOSUITE properties.
- Rank Selection: In Rank selection, parents are first sorted according to their fitness value and then each is given a probability of selection equal to their rank. Its advantage over Roulette Wheel is that it still gives individuals with a low fitness value a chance of being selected. In Roulette Wheel individuals with a high fitness value fill much more proportion of the probability space, but Rank Selection fixes this problem. It is denoted by `RANK` value in EVOSUITE properties.
- Tournament Slection: Individual chromosomes of the population are matched against each other in a tournament fashion and the winners are selected for crossover. It is denoted by `TOURNAMENT` value in EVOSUITE properties.
- Best-k: Selects the k top parents based on their fitness values. It is denoted by `BESTK` value in EVOSUITE properties.

3.1.2 Crossover methods. After choosing parents by applying a selection method, offsprings are created from parents by using a crossover function. There are several crossover functions proposed for use in Genetic Algorithms. We evaluated the performance of Uniform, Relative Single Point, and Two-point crossover functions implemented in EVOSUITE. We have extended EVOSUITE and implemented the two-point crossover method.

- Relative Single Point: This method is like the standard Single Point crossover function, however, in the Relative Single Point, the size of individuals is also taken into account. For example, If we want to select half of each parent and perform crossover, then for a chromosome of size 10, The cut point is five, but for a chromosome of size 14, the cut point is seven. This method is useful for solutions with individuals with variable size. The definition of size is dependent on the solution representation. In EVOSUITE an individual is a set of unit tests (test suite) and the size of test suites are variable.
- Uniform: In Uniform crossover function, two children exactly like each parent are cloned. Then for each gene of the offspring chromosomes it is decided by a configurable probability whether to swap it with each other or not.

3.2 Test Subjects

We have evaluated the performance of each configuration on ten open-source and popular Java libraries. In Table 1 you can see the list of projects and the version that we have performed our tests. These libraries are commonly used in java applications by developers. These are from different domains such as utility tools, mathematical functions, parsing, etc. We have avoided including libraries related to parallel or distributed applications since generating tests for such libraries may require providing unique conditions. Due to the limited computation resources, if a project had more than 20 classes, we have randomly chosen only 20 classes for test generation. In total, we have 165 classes. Our selection included a variety of simple and complex classes, the average count of methods is 18, and the average of target goals is 66. Below you can find a short description of each project directly quoted from their respective websites.

- Commons CLI: "The Apache Commons CLI library provides an API for parsing command line options passed to programs. It's also able to print help messages detailing the options available for a command line tool." [8]
- Commons Codec: "Apache Commons Codec (TM) software provides implementations of common encoders and decoders such as Base64, Hex, Phonetic and URLs." [9]
- Commons Math: "Commons Math is a library of lightweight, self-contained mathematics and statistics components addressing the most common problems not available in the Java programming language or Commons Lang." [7]
- http-request: "A simple convenience library for using a `HttpURLConnection` to make requests and access the response." [21]
- Joda Time: "Joda-Time provides a quality replacement for the Java date and time classes. Joda-Time is the de facto standard date and time library for Java prior to Java SE 8." [17]
- Joda Money: "Joda-Money provides a library of classes to store amounts of money. The JDK provides a standard currency class, but not a standard representation of money. Joda-Money fills this gap, providing the value types to represent money." [16]
- JSON Java: "The JSON-Java package is a reference implementation that demonstrates how to parse JSON documents into Java objects and how to generate new JSON documents from the Java classes." [15]
- JSoup: "Jsoup is a Java library for working with real-world HTML. It provides a very convenient API for fetching URLs and extracting and manipulating data, using the best of HTML5 DOM methods and CSS selectors." [13]
- Spatial4j: "LocationTech Spatial4j is a general purpose spatial / geospatial ASL licensed open-source Java library. Its core capabilities are 3-fold: to provide common geospatially-aware shapes, to provide distance calculations and other math, and to read and write the shapes to strings." [6]
- Eclipse Vertx: "Vert.x is a tool-kit for building Reactive applications on the JVM. Reactive application are both scalable as workloads grow, and resilient when failures arise." [5]

3.3 Test Procedure

We have tested each of the test subjects on all of the eight configurations. Given the random nature of evolutionary algorithms, we have run each configuration 10 times for each project and averaged the results. EVOSUITE implements several criteria targets. We have only used the branch coverage criteria, therefore, we compare the branch coverage values across each configuration. For each Java class, the search budget was 30 seconds. If a project has 20 classes, then given the eight configurations and ten runs, it would take approximately a maximum of 13 hours to complete. We did not change other parameters like mutation rates, crossover rates, etc, we used the default parameters set by EVOSUITE authors, as it was concluded in [2] that default parameters can achieve very good results.

We wrote a PowerShell script to automatically run the test generation process for each combination of configurations ten times. For other parameters of the GA algorithm, we used the default ones provided by EVOSUITE authors. The default population size in EVOSUITE is 50.

Table 1: Test Subjects

Project	Version	Count of Classes
Commons CLI	1.4	18
Commons Codec	1.15	20
Commons Math	4.0	20
http-request	6.0	1
Joda Time	2.10.9	20
Joda Money	1.0.2	18
JSON Java	9	17
JSoup	1.13.1	17
Spatial4j	0.9	18
Eclipse Vert.x	3.9.5	16

4 EXTENDING EVOSUITE

As a part of our evaluation, we have implemented the Two-point crossover function. By default EVOSUITE contains the Single Point and Uniform crossover functions, therefore we had to implement it ourselves. EVOSUITE has an extensible architecture, we were able to easily implement this new crossover function. The new class is in the package `org.evosuite.ga.operators.crossover`, and the class name is `DoublePointCrossOver`. The algorithm is the standard two-point crossover, it chooses two random points and applies the crossover. For individuals with a size of less than 4 it fallbacks to the Single Point crossover. To configure EVOSUITE to use our new crossover function, the `crossover_function` parameter has to be set to `DOUBLEPOINT`. This parameter can be changed by either passing as a command argument or setting in the properties file.

5 EVALUATION

In order to compare the influence of various crossover and selection functions, we evaluated the performance of test subjects. In total, we ran 18,442 tests over 165 Java classes. In particular our research questions are:

RQ1) *What is the effect of different selection functions of the GA algorithm in test generation?*
RQ2) *What is the effect of different crossover functions of the GA algorithm in test generation?*
RQ3) *Will changing selection/crossover functions improve results? Are they significant?*

Table 2: For each configuration, we calculated the mean of branch coverage, the size of test suite, the total length of the statements in the test suite, mutation score, count of fitness evaluations, and the standard deviation of branch coverage across all 18,444 runs.

Configuration	Coverage	Coverage σ	Size	Length	Mutation Score	Fitness Evaluations
Best-K Selection-Two-point crossover	**85.6%**	0.24	17.56	59.38	0.45	19608.08
Best-K Selection-Single-point crossover	**85.72%**	0.23	17.56	61.06	0.46	18514.32
Best-K Selection-Uniform Crossover	82.45%	0.25	18.09	110.13	0.44	11509.66
Rank Selection-Two-point crossover	85.23%	0.23	17.31	58.8	0.45	16262.85
Rank Selection-Single-point crossover	85.3%	0.23	17.34	60.18	0.45	15776.5
Rank Selection-Uniform Crossover	81.36%	0.25	15.25	50.35	0.43	11655.96
Roulette Wheel Selection-Two-point crossover	84.32%	0.24	17.11	57.02	0.45	17787.47
Roulette Wheel Selection-Single-point crossover	84.9%	0.23	16.99	56.67	0.45	16923.51
Roulette Wheel Selection-Uniform Crossover	80.76%	0.25	15.14	47.69	0.43	10709.78
Tournament Selection-Two-point crossover	84.07%	0.24	16.98	57.32	0.45	18288.11
Tournament Selection-Single-point crossover	84.48%	0.24	17.34	58.75	0.45	16084
Tournament Selection-Uniform Crossover	80.69%	0.25	15.07	49.08	0.42	11183.36

To answer our research questions we collected the results of test generation of EVOSUITE for every run across all eight configurations. We used an *R* script to calculate a few metrics. We calculated the mean of branch coverage, the size of test suite, the total length of the statements in the test suite, mutation score, count of fitness evaluations, and the standard deviation of branch coverage. In Table 2 you can see these metrics calculated across the whole 18,442 test runs. We can observe that the Best-k selection function, with Single-point crossover has the highest mean of branch coverage (85.72%). In Fig. 1 we show the box plot of branch coverage across the whole 18,442 test runs. The performance of all configurations is almost at the same level, however, we can see in Fig. 1 that the Uniform crossover function has consistently achieved a lower branch coverage than the other two crossover methods. The Single-point and Two-point crossover functions have almost the same performance.

To further analyze the results of each configuration we have provided the box plot of branch coverage for each project, from Fig. 2 to Fig. 9. The branch coverage of EVOSUITE is variable across projects. Again we see that the Uniform crossover function impacts the performance of the GA algorithm, it is more obvious in projects like Joda Money, Vert.x, http-request, and Spatial4j. The Vert.x project is one of the most complex software in Java, We see that the variance of branch coverage in it is high. But still the fact that EVOSUITE can generate unit tests with good branch coverage for such a complex project is noteworthy.

The dataset of collected metrics from EVOSUITE, calculated metrics, and the used *R* script is available in the code repository.

5.1 RQ1

As we observed, the selection functions do not play a significant role in Genetic Algorithms for automated test generation. We can't see a significant difference in performance in runs across the eight test configurations. The Best-k selection function has the highest branch coverage with the average of 85.72% across all 18,442 tests. Regardless of the crossover function in Table 2 we see that it achieves higher coverage than other selection functions. We also see that the Tournament selection function has the lowest branch coverage. Overall the branch coverage value does not vary much among various selection functions.

5.2 RQ2

As we observed, the crossover function can impact the branch coverage. The results of Single-point crossover and Two-point crossover functions are almost identical. But, the Uniform crossover function has the worst performance and consistently performs worse than the other two. The highest average of branch coverage for the Uniform Crossover function was 82.45%, which is three percent lower than the other two crossover functions. Also, the lowest branch coverage for it was 80.69% when it was used with Tournament

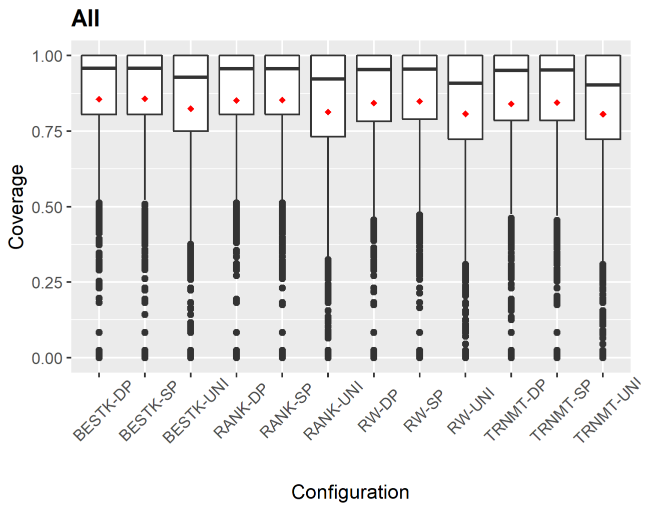

Figure 1: Box plot of branch coverage across the whole 18,441 tests

Selection function. It is noticeable that the average count of fitness evaluations of Uniform Crossover function was significantly lower than others in all setups. While the average fitness evaluation count of other crossover functions was more than 15,000, the highest value for Uniform crossover was 11,655. It is not clear to us that whether this observation is due to an implementation bug in EVOSUITE or the nature of Uniform function. However, its lower coverage values may be attributed to its algorithm. At each step, the Uniform function mixes much more genes than the other two functions, this act may decrease the branch coverage of better individuals.

5.3 RQ3

Except for the fact that the Uniform crossover function made results worse, we don't think that choice of selection and crossover functions can impact the results. Other factors such as the choice of fitness function and the main algorithm (we used the Standard GA) has much more effect as was evaluated by other studies [4].

5.4 Threats to Validity

We were able to study over a small sample of open-source Java projects and our empirical evaluations are prone to External Validity. However, even with our small sample subjects, a few patterns were emerging.

6 CONCLUSION

To study the effect of several selection and crossover functions on automated test generation with Genetic Algorithms, we performed 18,442 tests over 165 Java classes from 10 popular open-source projects. We employed the Standard Genetic algorithm implemented in EVOSUITE and measured the impact of Uniform, Single-point, and Two-point crossover, and Rank, Tournament, Best-k, Roulette Wheel selection functions. We discovered that the Uniform crossover function can result in poorer branch coverage. Also, we saw that the Best-k selection function yields the highest branch coverage. Nevertheless, the impact of various selection functions is small and other factors such as fitness function should be configured.

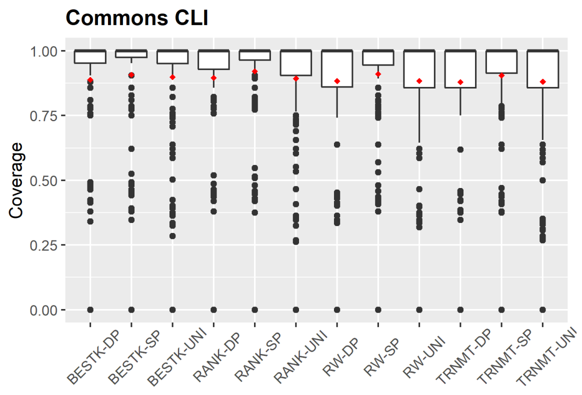

Figure 2: Box plot of branch coverage Commons CLI library

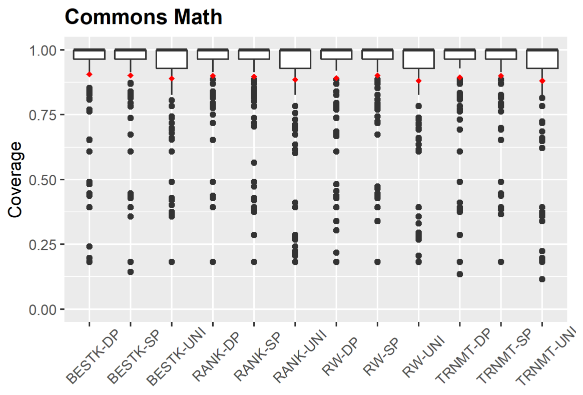

Figure 3: Box plot of branch coverage Commons Math library

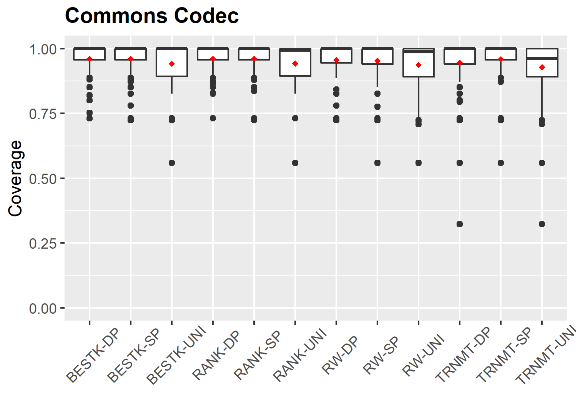

Figure 4: Box plot of branch coverage Commons Codec library

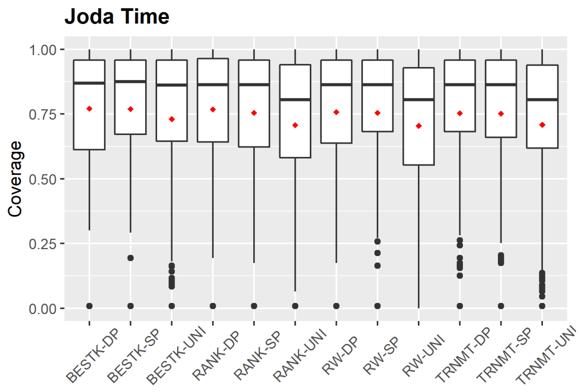

Figure 5: Box plot of branch coverage Joda Time library

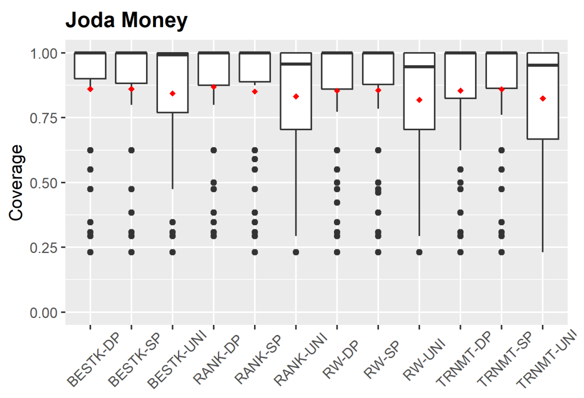

Figure 6: Box plot of branch coverage Joda Money library

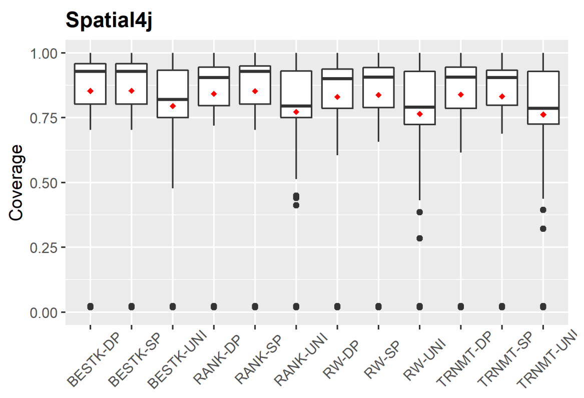

Figure 9: Box plot of branch coverage Spatial4j library

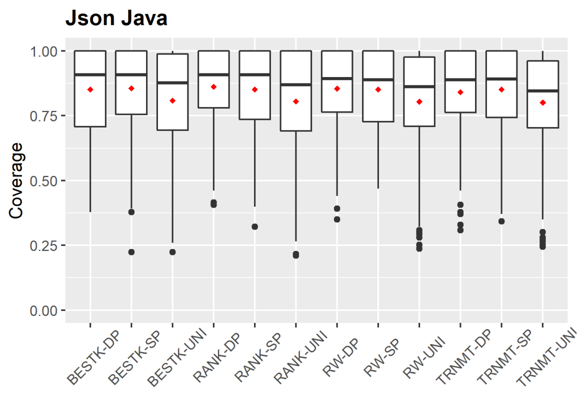

Figure 7: Box plot of branch coverage JSON Java library

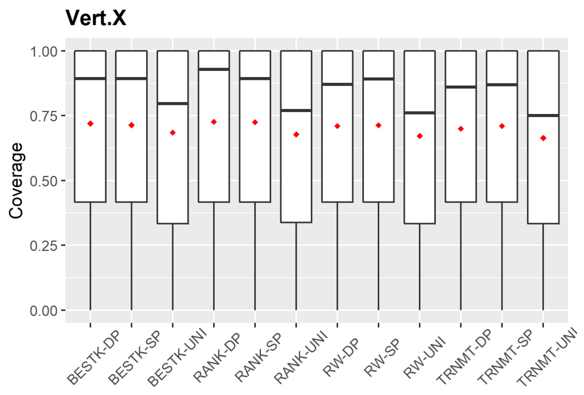

Figure 10: Box plot of branch coverage Vert.x framework

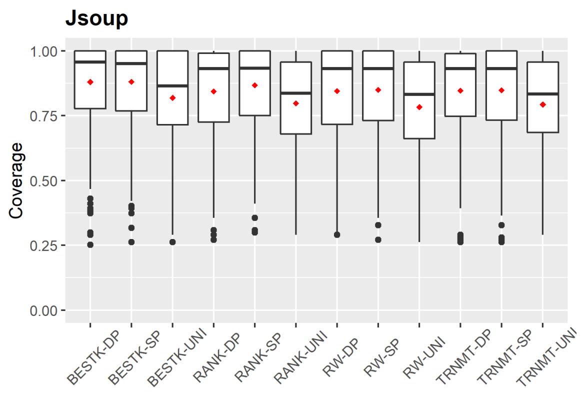

Figure 8: Box plot of branch coverage Jsoup library

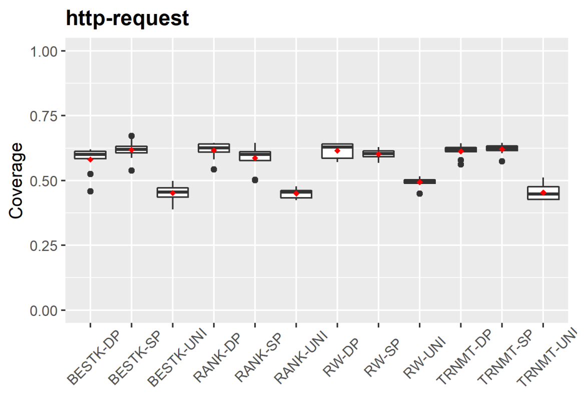

Figure 11: Box plot of branch coverage http-request library

REFERENCES

[1] Andrea Arcuri. 2017. Many Independent Objective (MIO) Algorithm for Test Suite Generation. *Lecture Notes in Computer Science* (2017), 3–17. https://doi.org/10.1007/978-3-319-66299-2_1

[2] Andrea Arcuri and Gordon Fraser. 2013. Parameter tuning or default values? An empirical investigation in search-based software engineering. *Empirical Software Engineering* (2013). https://doi.org/10.1007/s10664-013-9249-9

[3] Kamel Ayari, Salah Bouktif, and Giuliano Antoniol. 2007. Automatic Mutation Test Input Data Generation via Ant Colony. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* (London, England) *(GECCO ’07)*. Association for Computing Machinery, New York, NY, USA, 1074–1081. https://doi.org/10.1145/1276958.1277172

[4] José Campos, Yan Ge, Nasser Albunian, Gordon Fraser, Marcelo Eler, and Andrea Arcuri. 2018. An empirical evaluation of evolutionary algorithms for unit test suite generation. *Information and Software Technology* 104 (2018), 207 – 235. https://doi.org/10.1016/j.infsof.2018.08.010

[5] Eclipse Foundation. 2020. Eclipse Vert.x. https://vertx.io/

[6] Eclipse Foundation. 2020. LocationTech Spatial4j. https://projects.eclipse.org/projects/locationtech.spatial4j/

[7] The Apache Software Foundation. 2016. Commons Math. https://commons.apache.org/proper/commons-math/

[8] The Apache Software Foundation. 2019. Commons CLI. https://commons.apache.org/proper/commons-cli/

[9] The Apache Software Foundation. 2020. Commons Codec. https://commons.apache.org/proper/commons-codec/

[10] G. Fraser and A. Arcuri. 2011. Evolutionary Generation of Whole Test Suites. In *2011 11th International Conference on Quality Software*. 31–40. https://doi.org/10.1109/QSIC.2011.19

[11] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. 2012. Search-Based Software Engineering: Trends, Techniques and Applications. *ACM Comput. Surv.* 45, 1, Article 11 (Dec. 2012), 61 pages. https://doi.org/10.1145/2379776.2379787

[12] Mark Harman, Phil McMinn, Jerffeson Teixeira de Souza, and Shin Yoo. 2012. *Search Based Software Engineering: Techniques, Taxonomy, Tutorial.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1–59. https://doi.org/10.1007/978-3-642-25231-0_1

[13] Jonathan Hedley. 2020. jsoup: Java HTML Parser. https://jsoup.org/

[14] B. Korel. 1990. Automated software test data generation. *IEEE Transactions on Software Engineering* 16, 8 (1990), 870–879. https://doi.org/10.1109/32.57624

[15] Sean Leary. 2020. stleary/JSON-java: A reference implementation of a JSON package in Java. https://github.com/stleary/JSON-java/

[16] Joda Org. 2019. Joda Money. https://www.joda.org/joda-money/

[17] Joda Org. 2020. Joda Time. https://www.joda.org/joda-time/

[18] A. Panichella, F. M. Kifetew, and P. Tonella. 2018. Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets. *IEEE Transactions on Software Engineering* 44, 2 (2018), 122–158. https://doi.org/10.1109/TSE.2017.2663435

[19] José Miguel Rojas, Gordon Fraser, and Andrea Arcuri. 2016. Seeding strategies in search-based unit test generation. *Software Testing, Verification and Reliability* (2016), n/a–n/a. https://doi.org/10.1002/stvr.1601

[20] José Miguel Rojas, Mattia Vivanti, Andrea Arcuri, and Gordon Fraser. 2016. A detailed investigation of the effectiveness of whole test suite generation. *Empirical Software Engineering* (2016), 1–42. https://doi.org/10.1007/s10664-015-9424-2

[21] Kevin Sawicki. 2020. kevinsawicki/http-request: Java HTTP Request Library. https://github.com/kevinsawicki/http-request/

[22] Paolo Tonella. 2004. Evolutionary Testing of Classes. *SIGSOFT Softw. Eng. Notes* 29, 4 (July 2004), 119–128. https://doi.org/10.1145/1013886.1007528