

Search-based testing for Java

Antonio Vélez Estévez, Universidad de Cádiz.



UCA

Universidad
de Cádiz

This work is licensed under a “CC BY-NC-SA 3.0” license.



Introduction

Goals

Search-based testing in detail

Open Challenges in Search-Based Software Testing

EvoSuite SBST Tool

How EvoSuite works?

Experiments

Conclusions

Working with EvoSuite

Generating and executing tests with EvoSuite

Coverage Analysis

References

What is search-based software testing?

Search-Based Software Testing is the use of a meta-heuristic optimizing search technique, such as a Genetic Algorithm, to automate or partially automate a testing task.

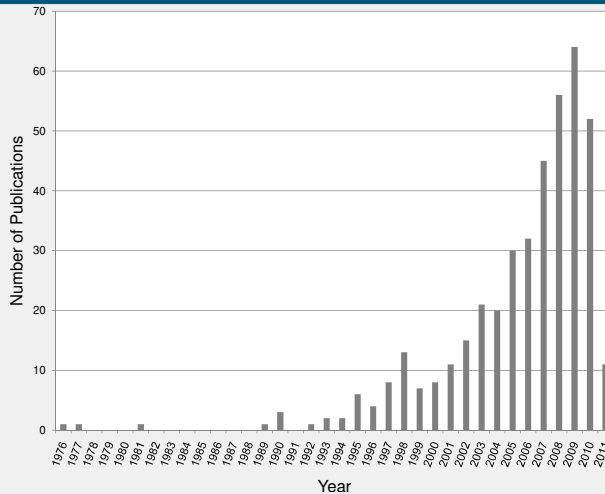
Which testing tasks have the community automated with SBST?

Functional testing, temporal testing, integration testing, regression testing, stress testing, mutation testing, test prioritisation, interaction testing, state machine testing and excepcion testing.

The first paper...

It appeared in 1976 and describes a technique for generating test data consisting of floating-point inputs. In this paper the test data were obtained by executing a version of the software under test, with these executions being guided toward to the required test data through the use of a 'cost function'. Inputs that were 'closer' to executing a desired path were rewarded with lower cost values whilst inputs with the higher ones were discarded.

P. McMinn, "Search-Based Software Testing: Past, Present and Future," 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, Berlin, 2011, pp. 153-163.



- Transform testing software problems to a search-based problems.
- Find the best solution from a potentially infinite search space, regarding a time limit and a fitness function.

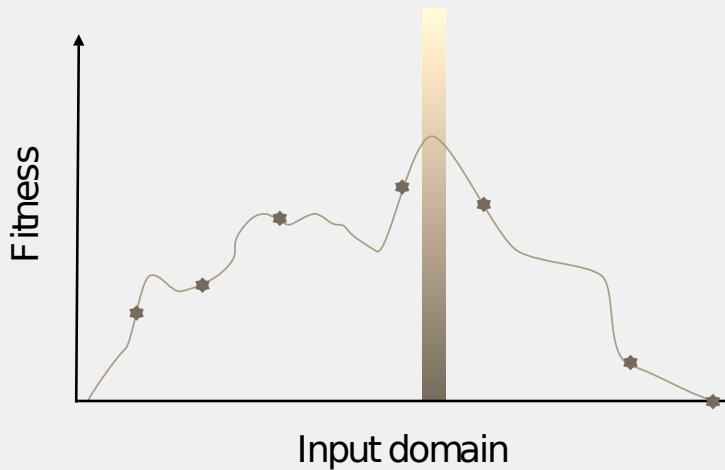
In order to apply search-based technique to a testing problem, we need to fulfill two requirements.

- **Representation:** the candidate solutions for the problem need to be encoded in order they can be manipulated by the search algorithm.
- **Fitness function:** A fitness function guides the search to the areas of the search domain by evaluating the possible solutions. It is problem specific.

We can solve the searching with several algorithms:

- Hill climbing.
- Random search.
- Genetic algorithm.
- Simulated Annealing.

Fitness function (Source: DOI 10.1109/ICSTW.2011.100)



- The Oracle problem. It is easy to detect system crashes but other oracles are more difficult.
- Incorporating human knowledge and interaction into search.
- Add learning approaches to search.

Reference: M. B. Cohen, "The Maturation of Search-Based Software Testing: Successes and Challenges," 2019 IEEE/ACM 12th International Workshop on Search-Based Software Testing (SBST), Motreal, QC, Canada, 2019, pp. 13-14.

*"**EvoSuite** is a search-based tool that optimizes whole test suites towards satisfying a coverage criterion."*

G. Fraser and A. Arcuri, "Evolutionary Generation of Whole Test Suites," in International Conference On Quality Software (QSI), Los Alamitos, CA, USA, 2011, pp. 31-40.

- Generation of JUnit 4 tests for the selected classes.
- Optimization of different coverage criteria, like lines, branches, outputs and mutation testing
- Tests are minimized: only the ones contributing to achieve coverage are retained.
- Generation of JUnit asserts to capture the current behavior of the tested classes.
- Tests run in a sandbox to prevent potentially dangerous operations.
- Virtual file system.
- Virtual network.

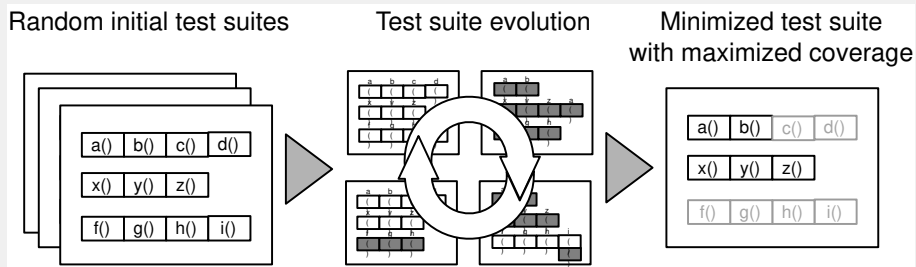


Figure: EvoSuite process (Picture extracted from the previously cited article)

1. A set of test suites is randomly generated.
2. That set is evolved using evolutionary search (genetic algorithm) towards satisfying a chosen coverage criterion.
3. The best resulting test suite is minimized.


```
import java.math.BigInteger;
class Karatsuba {
    private final static BigInteger ZERO = new BigInteger("0");
    public static BigInteger karatsuba(BigInteger x, BigInteger y) {
        int N = Math.max(x.bitLength(), y.bitLength());
        if (N <= 2000) return x.multiply(y);
        N = (N / 2) + (N % 2);
        BigInteger b = x.shiftRight(N);
        BigInteger a = x.subtract(b.shiftLeft(N));
        BigInteger d = y.shiftRight(N);
        BigInteger c = y.subtract(d.shiftLeft(N));
        BigInteger ac = karatsuba(a, c);
        BigInteger bd = karatsuba(b, d);
        BigInteger abcd = karatsuba(a.add(b), c.add(d));
        return ac.add(abcd.subtract(ac).subtract(bd).shiftLeft(N)).add(bd.shiftLeft(2*N));
    }
}
```

Authors: Robert Sedgewick and Kevin Wayne.

<https://introcs.cs.princeton.edu/java/99crypto/Karatsuba.java.html>

- 5 tests are generated.
- 3 tests are proving the arithmetic of the algorithm itself.
- 2 tests are proving the creating of the Karatsuba object and passing nulls as parameters to the method.

- Automatic tests are OK but they are difficult to manage.
- Better coverage does not imply high quality testing.
- Search-Based Software Testing is a complementary technique.
- It is a good technique to generate efficient and minimized tests for Object-Oriented Software.

In order to use EvoSuite with Maven we need the following:

Requirements

- Java Development Kit ≥ 8 . `sudo apt install openjdk-8-jre-headless`.
- Maven. `sudo apt-get install maven`.
- A text editor, **vi**, **emacs**, etc.
- (Optional but easiest) Eclipse e IntelliJ Idea have plugins to integrate EvoSuite.

To use EvoSuite the easiest way is to use Maven. To do so, we first will create a simple project.

Command to generate a simple project with Maven

```
mvn -B archetype:generate \  
  -DarchetypeGroupId=org.apache.maven.archetypes \  
  -DgroupId=es.uca.muiisc.psi \  
  -DartifactId=evosuitepresentacion
```

This will generate a folder with a **pom.xml** file inside it that we would have to modify to add the dependencies needed.

```
.  
evosuitepresentacion/  
pom.xml  
  src/  
    main/es/uca/muiisc/psi/  
      App.java  
    test/es/uca/muiisc/psi/  
      AppTest.java
```

We add the dependency with JUnit which is needed by EvoSuite in order automatically generate the unit tests.

Needed dependencies pom.xml

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

We also add the **evosuite-standalone-runtime** dependency to be able to run the tests.

Needed dependencies pom.xml

```
<dependencies>
  [...]
  <dependency>
    <groupId>org.evosuite</groupId>
    <artifactId>evosuite-standalone-runtime</artifactId>
    <version>1.0.6</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```


To use EvoSuite without worrying about the **classpath**, we will add the plugin **evosuite-maven-plugin**.

Needed plugins pom.xml

```
<build>
  <plugins>
    <plugin>
      <groupId>org.evosuite.plugins</groupId>
      <artifactId>evosuite-maven-plugin</artifactId>
      <version>1.0.6</version>
    </plugin>
  </plugins>
</build>
```

In order to be able to automatically download the plugin we would have to add the repository.

Needed plugin repository pom.xml

```
<pluginRepositories>
  <pluginRepository>
    <id>EvoSuite</id>
    <name>EvoSuite Repository</name>
    <url>http://www.evosuite.org/m2</url>
  </pluginRepository>
</pluginRepositories>
```

To generate the tests, we will need some code to try out first. So, in order to try something different than what already exists in EvoSuite's webpage, we have created a simple reservation system for a hostel.

We have to download its files and place them on to `./src/main/es/uca/muiisc/psi` directory. To download the project we could run the following command:

```
wget https://github.com/avleze/material-evosuite/archive/master.zip
```

To generate the tests with EvoSuite, we will execute the following command with Maven:

```
mvn evosuite:generate
```

This command will be executable thanks to the plugin included in the file **pom.xml** . The test generated by EvoSuite will be stored in the absolute path:

```
$PROJECT_ROOT$/.evosuite/best-tests/$PACKAGE_NAME$
```

Whereas:

- **\$PROJECT_ROOT\$** is the path to the root of the project created with maven.
- **\$PACKAGE_NAME\$** is the name of the package where we will find the class from wich we want to see the generated tests.

To include the tests generated by EvoSuite in the Maven project, we need to write the following command:

```
maven evosuite:export
```

Next, we can execute the tests using the command **mvn test**.

To proceed with the coverage analysis of the tests generated by EvoSuit, we can use the following plugins

Plugins

- Jacoco.
- Cobertura.
- Clover.
- PIT.






We choose **cobertura** for its easy instalation and configuration.

pom.xml

```
<plugins>
  [...]
  <plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>cobertura-maven-plugin</artifactId>
    <version>2.7</version>
  </plugin>
</plugins>
```

Once we have exportated and executed the test we can analyze the coverage following these two steps:

- `mvn cobertura:cobertura`, we ask **cobertura** to start analyzing.
- `mvn cobertura:dump-datafile`, we ask **cobertura** to show the results on screen.

-  Y. Jia, M. B. Cohen, M. Harman, and J. Petke, “Learning combinatorial interaction test generation strategies using hyperheuristic search,” in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, pp. 540–550, May 2015.
-  P. McMinn, “Search-based software testing: Past, present and future,” in *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, pp. 153–163, March 2011.
-  G. Fraser and A. Zeller, “Mutation-driven generation of unit tests and oracles,” *IEEE Transactions on Software Engineering*, vol. 38, pp. 278–292, March 2012.
-  G. Fraser and A. Arcuri, “Evolutionary generation of whole test suites,” in *2011 11th International Conference on Quality Software*, pp. 31–40, July 2011.
-  M. Harman, Y. Jia, and Y. Zhang, “Achievements, open problems and challenges for search based software testing,” in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pp. 1–12, April 2015.

- <http://www.evosuite.org/documentation/tutorial-part-1/> [Accessed on 17-12-2019].
- <http://www.evosuite.org/documentation/tutorial-part-2/> [Accessed on 17-12-2019].
- <http://www.evosuite.org/documentation/measuring-code-coverage/> [Accessed on 18-12-2019].