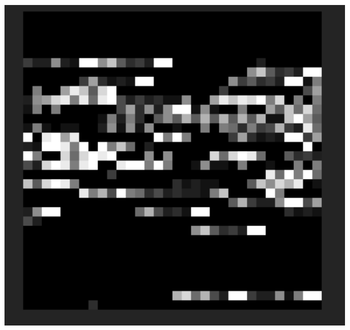
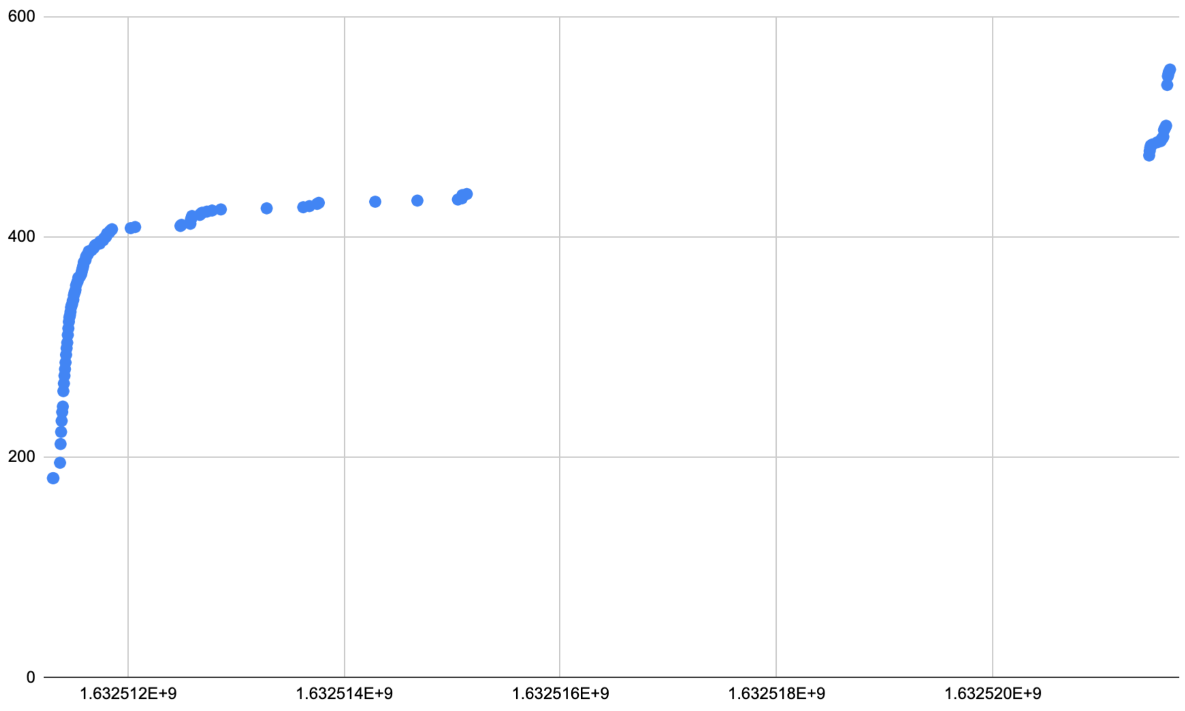
1. I seeded AFL with most of my manual tests and the AFL is a superset of such test cases. The total run time of AFL is around 4 hours and 43 minutes. During this time, AFL created 522 images and the final coverage of the tests comes out to be 37.08%.
2. The image on the left is the AFL version of the image on the right.



It is an interesting image because the image successfully captured the shape and outlines of the original image. In addition, it outlines the image with more fuzzy pixels.

3. The scatterplot shows that the path increases very quickly at the beginning and then at some point it starts to slow down and thus the scatter plot becomes smoother because the execution time for the path becomes longer.



4. For the second subject program, the assessment of manually created test cases of JSoup indicates 0 branch coverage for all classes in package “examples” and an average branch coverage between 80% and 90% for the rest of classes. However, the report of EvoSuite-created test cases indicates all packages have branch coverage, including the examples package. In addition, the tests created by EvoSuite seem to be more extreme in terms of the coverage rate, since some branches have coverage below

50% while some other branches can be 100% covered. The average branch coverage of EvoSuite-created test cases is similar to that of manually created tests.

5. In several cases, the EvoSuite produced tests have higher coverage than human tests. For example, the EvoSuite produced tests have 100% coverage for class “ParseError” but the human made tests only have 54% at mean time. After carefully analysis of the source codes as well as two versions of test codes, I found human tests only test about two out of four constructors of the ParseError Class. The EvoSuite strictly follows the test coverage logic and generates ParseError Objects by all four kinds of constructors thus winning the coverage competition for this class. In this case, the human’s lack of tests may due to developer’s subjective idea that only some constructors work is fine because the ParseErrors in the program are rarely triggered by the other two constructors. The quality of the test created by EvoSuite is good because it exhausts all possibilities a ParseError can be generated, and the use of junit also ensures the readability codes, therefore the test failure can notice the developer of a bug.
6. Sometimes EvoSuite produced tests have lower coverage than the human created tests. For example, in the class “FormElement”, the branch coverage for the human test is 84% but for EvoSuite created one is only 48%. The main reason for EvoSuite tests’ poor performance in this class is the lack of coverage for the function “formData()”, which is a complex function with multiple if statements to retrieve data from html. My hypothesis for limited test cases created by EvoSuite is that the input of “formData()” has so many constrains and need to be preprocessed, so that EvoSuite has no idea of creating an effective test case. Developers on the other hand, know what the form and constrains of the input html, so they can handcraft a long html String deliberately for further testing.
7. One strength of AFL is that it can create a lot of test cases and paths by fuzzing the mutated tests. However, the weakness is also very obvious: it takes a lot of time to get good results. One of the strengths of EvoSuite is that it is very effective for classes with multiple short functions and overloaded functions with relatively simple input data form, but when the input data is complex and may have constraints or need preprocessing, EvoSuite is unable to generate effective further tests for this function, which is its weakness. The usability of the AFL and Evosuite is similar. They both generated many useful test cases. AFL created many test cases based on the mutated test inputs, while it labeled them with unique IDs and source file IDs in their generated file names. It helps the software engineers to quickly locate the wanted test cases and compare them with the original ones. With some limited test input, AFL helps us to reach a good test coverage. For Evosuite, it is also able to create decent test results and cover many branches that many human engineers normally could not cover easily under most situations, with some limitations because it is not capable of covering every branch. As for efficiency, Evosuite seems to be a faster tool than AFL. Personally I think both AFL and Evosuite are very good tools for human testers. It would be almost impossible for humans to manually create lots of test cases within a short period of time. These tools really help us to improve the efficiency and quality, which means that it reduces the cost of software testing, which is what companies love to see. I would use AFL personally for some projects such as machine learning and computer vision, mostly because AFL really helps to create lots of test images, which might be hard for us to do manually.

EvoSuite is a good tool, especially for Java programs. However, it is more useful on large scale based programs. Therefore for many of my personal projects I may not use it, but maybe in the future when I am working in the industry with large projects I would use them as one of the testing tool options.