

Developing a simple mutation analyzer for the Major Mutation Framework

Group 52

Divyesh Harit, Rishi Mody, Thuan Binh and
Utkarsh Srivastava

Project Goals

- Implement Mutation Analyzer as a standalone Java program
- Input: Set of mutants and a test suite.
- Output: Mutation score with a summary of mutant kill information

Major Mutation Framework: An overview

- A mutation analysis tool comprising of
 - A compiler-integrated mutator
 - Mutation analyzer(execute test suite on mutants)
- Enables efficient and scalable mutation analysis
- Enables for both strong and weak mutation analysis

Input specification

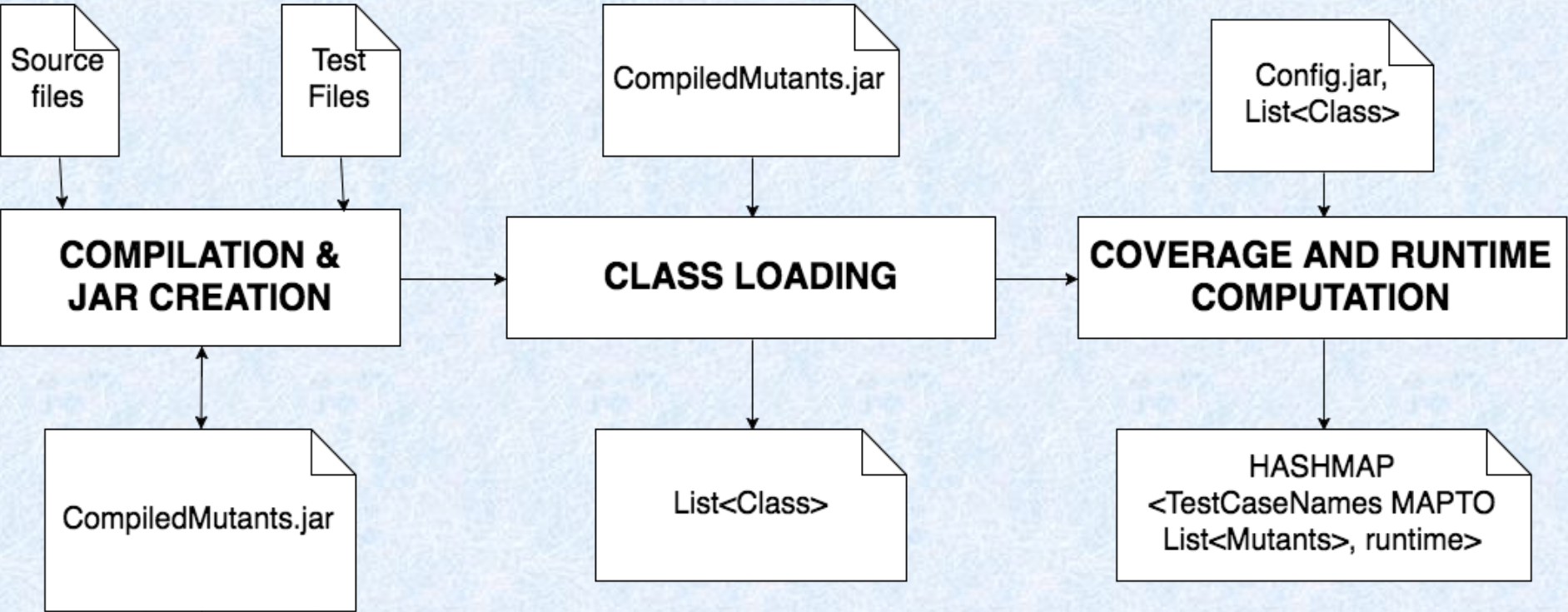
- Build provided source and test files using Major javac and XMutator flag(generates and embeds mutants in code)
- Create a JAR file of the resulting .class files
- Our Mutation Analyzer takes two program arguments:
 - Path to the created JAR
 - Path to the mutants.log file generated upon build

Output specification

- A GUI reporting information about
- Mutation kill rate
 - Summary of killed mutants
 - List of mutant ids not killed
 - Runtime of the code

Design choices

- Separation of Concerns: Every individual function and associated utilities have been divided into appropriate package structures by focusing distinction on the following three phases:
 - Pre-pass to record run time and coverage information per mutant for a test case
 - Prioritization of a test suite based on above information
 - Running test case on every mutant to analyze mutant kill information
- Composite pattern for output view
- Iterator pattern for executing functions per test class



Run a script to build the code using javac compiler form "MAJOR"and create a single jar of all source files and generated mutants

From the given JAR in the path provided, extract all the file names that contain substring "Test" and load them using a class loader

Create a hashmap of test case names per test class mapped to list of mutant IDs they cover and their runtime information

HASHMAP
(Result of
previous step)

Prioritized map,
Info Tracker
&
Mutants.log file

Mutants kill
summary

**PRIORITIZER / INFO
TRACKER**

MUTATION ANALYSIS

OUTPUT

Prioritized map
< Test class, List<SortedTestCaseNames>>
&
Info Map<Test class, Set<Mutant ids
covered>>

Mutants kill
summary

Visualization
of this info.
(UI)

Use the hashmap from previous step to produce a prioritized map of test case names based on increasing runtime, and no. of mutants covered. Also keeps a track of ids of mutants covered by every class, so as to optimize on mutation analyzer.

Find total no. of mutants from mutants.log. Run prioritized test cases leveraging information from Info-Tracker on each mutant. Whenever a test case fails, it returns a failed status by breaking the execution for that mutant
Status code 0: Not killed
Status Code 1: Killed
Status Code 2: Killed with Timeout

Display mutants killed, mode of mutants killed (assertion, timeout, etc), list of mutants not killed, total runtime of entire code

Code Structuring



Libraries and Utilities

- InputStreamReader: Read contents from mutants.log file
- URLClassLoader: To load contents of mutants.jar file
- JarEntry: TO iterate over elements in the JAR
- JunitCore, Request, Result: To create a request, run and record the results of a test case in a test file
- JFrame: GUI output
- ExecutorService, Future, Callable: For handling time outs and Thread Execution.

Data Structures

- MutationInfoClass: Holding information from Pre-Pass phase
- Mutation AnalyserOutput: Hold Mutant kill information to display output.
- Map<String, List<String>> Prioritized Tests: Class name v/s ordered test cases.
- Map<String, Set<Integer>> MutationInfoPerClass: Class name v/s Ids of Covered mutants.

Optimizations


- Using coverage and runtime information to prioritize test case execution in each test class to reduce overall runtime of the code
- A mutant is set to be killed as soon as a test case for that iteration fails. This saves time by not calling further unnecessary test case executions.
- Using executor service with Timeout to handle test cases stuck in an infinite loop

Limitations

- The code takes coverage and runtime into consideration but does not cater to state infection through weak-mutation analysis
- JAR as an input is essential as per design.
- To identify test class files in the JAR, we assume that the substring “Test” is present in the .class file being analyzed.

Code screenshots

File Edit View Search Tools Documents Help

Open ▾ 

ant clean
ant compile
cd bin
jar cf ../../Jars/Mutants.jar *

Using javac from "MAJOR"
and XMutator



```
nts.log (-workspace_new/utkarsh1404/...)
1:LVR:0:POS:triangle.Triangle@classify:20:0 |==> 1
2:LVR:0:NEG:triangle.Triangle@classify:20:0 |==> -1
3:ROR:==(int,int):==(int,int):triangle.Triangle@classify:20:a <= 0 |==> a <= 0
4:ROR:==(int,int):==(int,int):triangle.Triangle@classify:20:a <= 0 |==> a == 0
5:ROR:==(int,int):TRUE(int,int):triangle.Triangle@classify:20:a <= 0 |==> true
6:LVR:0:POS:triangle.Triangle@classify:20:0 |==> -1
7:LVR:0:NEG:triangle.Triangle@classify:20:0 |==> 1
8:ROR:==(int,int):==(int,int):triangle.Triangle@classify:20:b <= 0 |==> b <= 0
9:ROR:==(int,int):==(int,int):triangle.Triangle@classify:20:b <= 0 |==> b == 0
10:ROR:==(int,int):TRUE(int,int):triangle.Triangle@classify:20:b <= 0 |==> true
11:COR:||(boolean,boolean):!(boolean,boolean):triangle.Triangle@classify:20:a <= 0 || b <= 0 |==> a <= 0 || b <= 0
12:COR:||(boolean,boolean):!(boolean,boolean):triangle.Triangle@classify:20:a <= 0 || b <= 0 |==> a <= 0 || b <= 0
13:COR:||(boolean,boolean):!(boolean,boolean):triangle.Triangle@classify:20:a <= 0 || b <= 0 |==> a <= 0 || b <= 0
14:COR:||(boolean,boolean):!(boolean,boolean):triangle.Triangle@classify:20:a <= 0 || b <= 0 |==> a <= 0 || b <= 0
15:LVR:0:POS:triangle.Triangle@classify:20:0 |==> 1
16:LVR:0:NEG:triangle.Triangle@classify:20:0 |==> -1
17:ROR:==(int,int):==(int,int):triangle.Triangle@classify:20:c <= 0 |==> c <= 0
18:ROR:==(int,int):==(int,int):triangle.Triangle@classify:20:c <= 0 |==> c == 0
19:ROR:==(int,int):TRUE(int,int):triangle.Triangle@classify:20:c <= 0 |==> true
20:COR:||(boolean,boolean):!(boolean,boolean):triangle.Triangle@classify:20:a <= 0 || b <= 0 || c <= 0 |==> (a <= 0 || b <= 0 || c <= 0) |==> (a <= 0 || b <= 0 || c <= 0)
21:COR:||(boolean,boolean):!(boolean,boolean):triangle.Triangle@classify:20:a <= 0 || b <= 0 || c <= 0 |==> (a <= 0 || b <= 0 || c <= 0) |==> (a <= 0 || b <= 0 || c <= 0)
22:COR:||(boolean,boolean):!(boolean,boolean):triangle.Triangle@classify:20:a <= 0 || b <= 0 || c <= 0 |==> (a <= 0 || b <= 0 || c <= 0) |==> (a <= 0 || b <= 0 || c <= 0)
23:COR:||(boolean,boolean):!(boolean,boolean):triangle.Triangle@classify:20:a <= 0 || b <= 0 || c <= 0 |==> (a <= 0 || b <= 0 || c <= 0) |==> (a <= 0 || b <= 0 || c <= 0)
24:LVR:0:POS:triangle.Triangle@classify:23:0 |==> -1
25:LVR:0:NEG:triangle.Triangle@classify:23:0 |==> 1
26:ROR:==(int,int):==(int,int):triangle.Triangle@classify:24:a == b |==> a == b
27:ROR:==(int,int):==(int,int):triangle.Triangle@classify:24:a == b |==> a == b
28:ROR:==(int,int):FALSE(int,int):triangle.Triangle@classify:24:a == b |==> false
29:LVR:0:POS:triangle.Triangle@classify:25:1 |==> 0
30:LVR:0:NEG:triangle.Triangle@classify:25:1 |==> -1
31:AOR:==(int,int):==(int,int):triangle.Triangle@classify:25:trian + 1 |==> trian % 1
32:AOR:==(int,int):==(int,int):triangle.Triangle@classify:25:trian + 1 |==> trian % 1
33:AOR:==(int,int):==(int,int):triangle.Triangle@classify:25:trian + 1 |==> trian % 1
34:AOR:==(int,int):==(int,int):triangle.Triangle@classify:25:trian + 1 |==> trian % 1
35:ROR:==(int,int):==(int,int):triangle.Triangle@classify:27:a == c |==> a == c
36:ROR:==(int,int):==(int,int):triangle.Triangle@classify:27:a == c |==> a == c
37:ROR:==(int,int):FALSE(int,int):triangle.Triangle@classify:27:a == c |==> false
38:LVR:0:POS:triangle.Triangle@classify:28:2 |==> 0
39:LVR:0:NEG:triangle.Triangle@classify:28:2 |==> -2
40:AOR:==(int,int):==(int,int):triangle.Triangle@classify:28:trian + 2 |==> trian % 2
41:AOR:==(int,int):==(int,int):triangle.Triangle@classify:28:trian + 2 |==> trian % 2
42:AOR:==(int,int):==(int,int):triangle.Triangle@classify:28:trian + 2 |==> trian % 2
43:AOR:==(int,int):==(int,int):triangle.Triangle@classify:28:trian + 2 |==> trian % 2
44:ROR:==(int,int):==(int,int):triangle.Triangle@classify:30:b == c |==> b == c
45:ROR:==(int,int):==(int,int):triangle.Triangle@classify:30:b == c |==> b == c
46:ROR:==(int,int):FALSE(int,int):triangle.Triangle@classify:30:b == c |==> false
47:LVR:0:POS:triangle.Triangle@classify:31:3 |==> 0
48:LVR:0:NEG:triangle.Triangle@classify:31:3 |==> -3
49:AOR:==(int,int):==(int,int):triangle.Triangle@classify:31:trian + 3 |==> trian % 3
50:AOR:==(int,int):==(int,int):triangle.Triangle@classify:31:trian + 3 |==> trian % 3
51:AOR:==(int,int):==(int,int):triangle.Triangle@classify:31:trian + 3 |==> trian % 3
52:AOR:==(int,int):==(int,int):triangle.Triangle@classify:31:trian + 3 |==> trian % 3
53:LVR:0:POS:triangle.Triangle@classify:33:0 |==> 1
54:LVR:0:NEG:triangle.Triangle@classify:33:0 |==> -1
55:ROR:==(int,int):==(int,int):triangle.Triangle@classify:33:trian == 0 |==> trian == 0
56:ROR:==(int,int):==(int,int):triangle.Triangle@classify:33:trian == 0 |==> trian == 0
57:ROR:==(int,int):FALSE(int,int):triangle.Triangle@classify:33:trian == 0 |==> false
58:AOR:==(int,int):==(int,int):triangle.Triangle@classify:34:a + b |==> a + b
```

Name: Main

Main (x)= Arguments

JRE

Classpath

Source

Program arguments:

"/home/utkarsh1404/workspace_new/Jars/Mutants.jar" "/home/utkarsh1404/workspace_new/triangle_HW/mutants.log"

Variables...

Total no. of Mutants killed due to TimeOut : 0

Problems @ Javadoc Declaration Console JUnit

Main [Java Application] /home/utkarsh1404/Downloads/eclipse-installer/jre/bin/java (Dec 13, 2016, 12:09:46 AM)

-----Starting Mutation Analysis-----

STEP 1 : Pre-Pass (Collecting Mutation coverage and run-time info for each test case)

STEP 2 : Performing Test Suite Prioritization

STEP 3 : Performing Strong Mutation Analysis

Total no. of mutants = 139
100 mutants analysed!!

Total no. of Un-Killed Mutants : 9
Un-Killed Mutant Ids : [11, 55, 72, 83, 94, 104, 120, 127, 136]

Run Time of code (in seconds) : 1.106

Sample Run on Real World Data Set: Numerics4J

- Xmutator Flag: Set to “ALL”
- Total number of Mutants generated: 16708

The image shows a screenshot of a Java Swing application window titled "Major Mutation Framework : Mutation Analyser". The window has a dark gray title bar with standard OS controls (close, maximize, minimize). The main content area is white and contains several lines of black text reporting mutation test results. At the bottom center, there is a label for the execution time. A horizontal scrollbar is visible at the very bottom of the window, indicating that the text content exceeds the width of the frame.

Major Mutation Framework : Mutation Analyser

Total Number of Mutants Present : 16708 Total no. of Mutants killed due to TimeOut : 228 Total no. of Un-Killed Mutants : 5971
Total Number of Mutants Killed : 10737
Mutation Score : 64.26262868087144%

Run Time of code (in seconds) : 828.046