Lancaster University

SlowCoach: Mutating Code to Simulate Performance Bugs

Nov. 2022

Yiqun Chen, Oliver Schwahn, Roberto Natella, Matthew Bradbury and Neeraj Suri

y.chen101@lancaster.ac.uk

Agenda

Lancaster University

- Motivation
- Background
- Framework
- Evaluation
- Conclusion

Motivation

Lancaster University

- Testing a program
- Correctness of the test suite?
- Mutate the code (inject faults)
- Could test suite identify mutants?

Example – Code Mutation

Lancaster University

```
- if (cond_a && cond_b) {
+ if (cond_a || cond_b) {
    do_something();
  }
```

Lancaster University

The quality of performance testing?

Background – Terminology

Lancaster University

- MT: Mutation Testing
- PMT: Performance Mutation Testing
- SUT: Software Under Test

Background – MT

Lancaster University

```
- if (cond_a && cond_b) {
+ if (cond_a || cond_b) {
    do_something();
  }
```

- Mutant – a mutated copy of source code
- Mutation Operator – a syntactic rule defining how the source code should be mutated
- Mutation Score – a score that grades the quality of the test suite, usually computed as *number of killed mutants / all mutants*

Background – PMT

Lancaster University

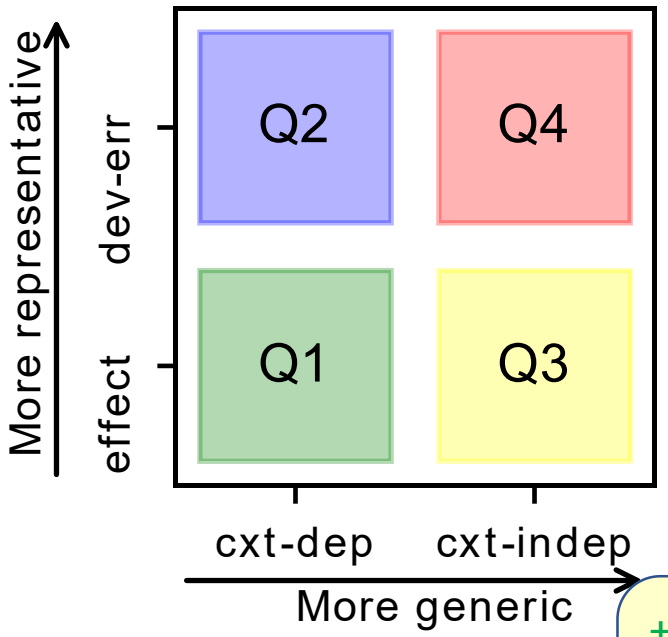
- Functional equivalence – mutants must align with the functionality of the original program.
- Context dependency – not all performance bugs can be generalized and encoded as syntactic rules. Mutation operators need extra information to effectively simulate performance bugs.

Background – Fault Models

Lancaster University

```
  for (i = 0; i < length;
       i++) {
-   if (can_go_fast()) {
+   if (0) {
      light_computation();
    } else {
      heavy_computation();
    }
  }
```

Q2 Example



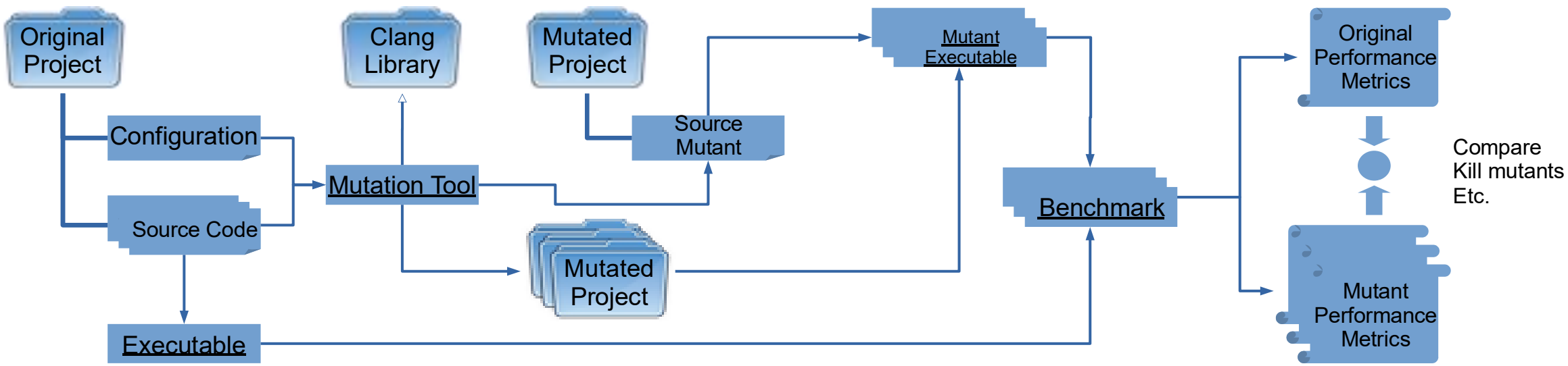

```
  for (int i = 0; i < 1024;
       i++) {
-   if (some_cond(i)) break;
    do_something();
  }
```

Q4 Example

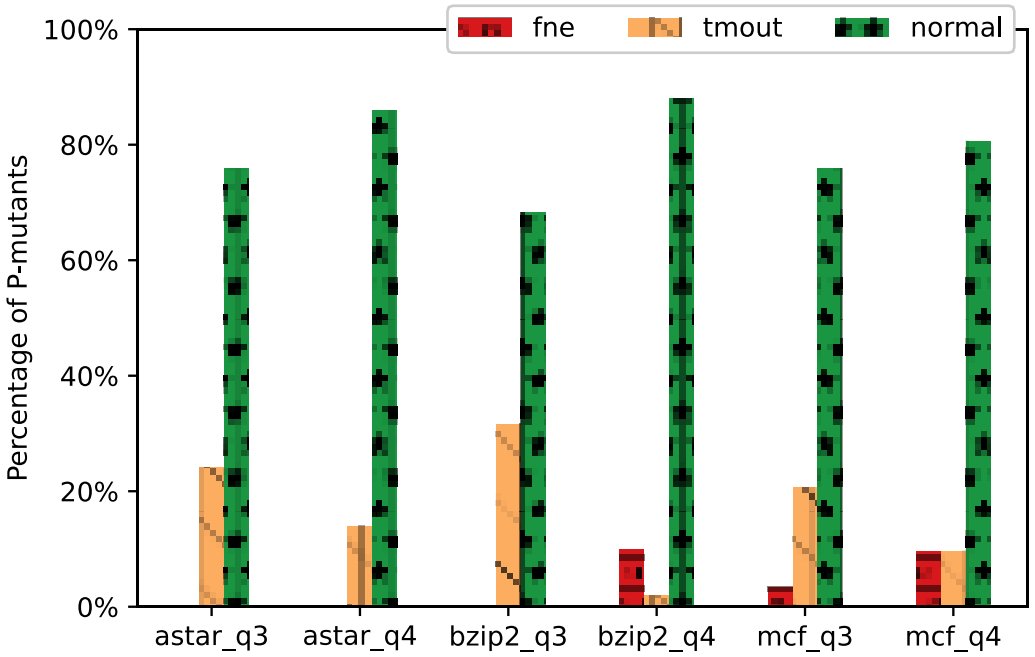
```
+ volatile int sum = 0, foo[ARR_LEN];
+ for(int i = 0; i < foo_len; i++) {
+   sum += foo[i];
+ }
```

(Q1 &)Q3 Example

Framework


Original Project
Clang Library
Mutated Project
Mutant Executable
Original Performance Metrics
Configuration
Source Mutant
Mutation Tool
Source Code
Benchmark
Compare
Kill mutants
Etc.
Mutated Project
Mutant Performance Metrics
Executable

Evaluation – Functional Equivalence



Equivalence in output

Legends:

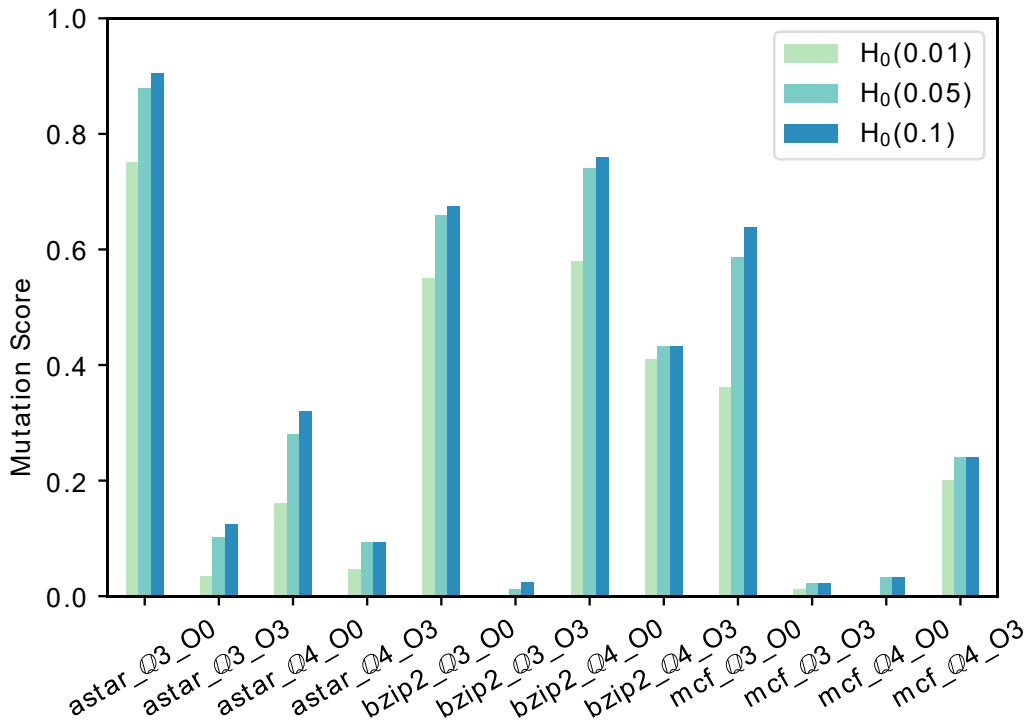
- Fne = functionally deviated mutants
- Tmout = timeouted mutants
- Normal = normal mutants

Evaluation – Mutation Score

Lancaster University

- Mutation score
  - Execution time of context independent mutants
  - Repeat 30 times
  - P_m is the performance of the mutant
  - P_b is the performance of the original program (baseline)
  - KS-test on execution time (H_0: $P_m \leq P_b$)
- Case study on Q2 (context dependent) mutants

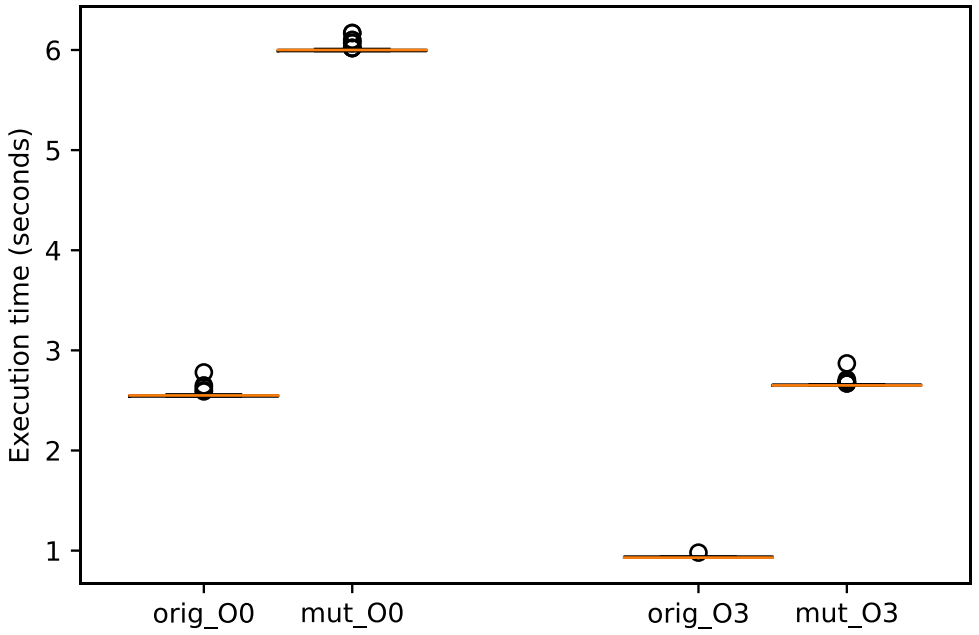
Evaluation – Mutation Score (Context Independent)



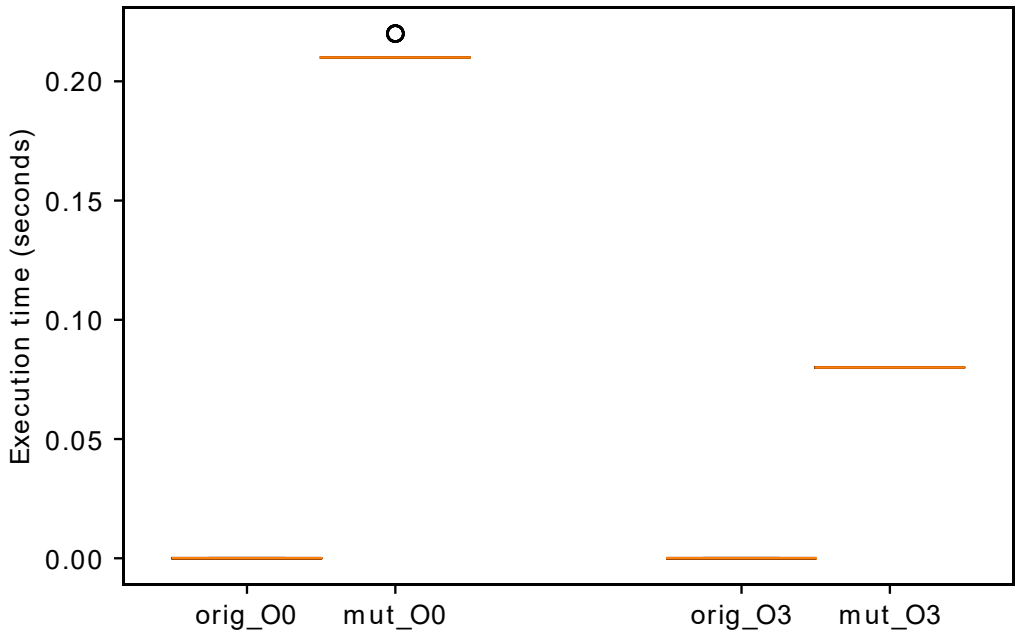
- `astar`, `bzip2` and `mcf`
- O0: not optimized
- O3: fully optimized
- Score ranges from 0 – 1
- Optimization affects mutation scores

Evaluation – Case Study (Context Dependent)

Lancaster University



Q2-A



Q2-B

Conclusion

Lancaster University

- Proposed a classification of four different fault models
- A methodology to use PMT to evaluate test suites
- Ability to generate and inject context dependent performance bugs

Lancaster University

Thank you for attending, any questions?