Automated Search For “Good” Coverage Criteria

Position Paper

Phil McMinn  University of Sheffield
Mark Harman  University College London
Gordon Fraser  University of Sheffield
Gregory Kapfhammer  Allegheny College
Coverage Criteria: The “OK”, The Bad and The Ugly

The “OK”

- Divide up system into things to test
- Useful to generate tests on if no functional model exists
- Indicates what parts of the system are and aren’t tested
The Bad

- Not based on anything to do with faults, not even:
  - Fault histories
  - Fault taxonomies
  - Common faults
The Ugly

• Studies disagree as to which criteria are best

• Coverage or test suite size?
The Key Question of this Talk

Can we evolve “good” coverage criteria?

Coverage criteria that are better correlated with fault revelation?
Why This Might Work

• The best criterion might actually be a mix and match of aspects existing criteria

  • For example “cover the top $n$ longest d-u paths, and then any remaining uncovered branches”

• Or…
Maybe this is One Big Empirical Study using SBSE

... which aspects of which criteria and how much

less branches more

less complex d-u chains more

less basis paths more
What About Including Aspects Not Incorporated into Existing Criteria

Non functional aspects

• For example timing behaviour, memory usage
  • “Cover all branches using as much memory as possible”

Fault histories

• “Maximize basis path coverage in classes with the longest fault histories”
“Isn’t This Just Mutation Testing?”

Our criteria are more like generalised strategies

- Potentially more insightful to the nature of faults
- Cheaper to apply
  (coverage is generally easier to obtain than a 100% mutation score)

Perhaps different strategies will work best for different types of software, or different teams of software developers
How This Might Work
Fault Database

Need examples of real faults

• Defects4J

• CoREBench

• … or, just use mutation
Fitness Function

“Goodness” is correlation between greater coverage and greater fault revelation

• Needs test suites to establish
Generation of Test Suites

At least two possibilities

• Generate up front universe of test suites

• Generate specific test suites with the aim of achieving specific coverage levels of the criteria under evaluation (drawback: expensive)
Search Representation

GP Trees

AND

up to 50% branch coverage

OR

over 75% basis path coverage

maximise memory usage
Handling Bloat

GP techniques classically involve “bloat”

• Consequence: generated criteria may not be very succinct

• Various techniques could be applied to simplify the criteria, e.g. delta debugging
Overfitting

The evolved criteria may not generalise beyond the systems studied and the faults seeded

• May not be a disadvantage:
  • insights into classes of system
  • faults made by particular developers
  • … apply traditional techniques from machines learning to combat overfitting.
Summary

Our Position:
SBSE can be used to automatically evolve coverage criteria that are well correlated with fault revelation.

Over to the audience:
Is it feasible that we could do this?