Allegheny College Summer Research Symposium 2007
Using Call Trees for Test Suite Reduction and Prioritization
Adam M. Smith

Adam M. Smith

adammatthewsmith@gmail.com

Department of Computer Science
Allegheny College
http://cs.allegheny.edu

Gregory M. Kapfhammer

gkapfham@allegheny.edu

Regression Testing Challenges

- Software development is complex.
- To ensure correctness developers write **unit tests** for a particular section of code.
- **Regression testing***:* Each time new functionality is added to the project the new tests are run in addition to the old ones in order to check for regression in the project.
- As software systems grow in size, the size of the **test suite** also increases.
- Regression testing is very important to the correctness of the software, but a developer cannot afford to wait a long time for a test suite to run.

Related Work

- **Reduction**: Removing test cases from the test suite.
- **Prioritization**: Reordering the test suite so that tests that cover more of the program are executed before tests that cover less.
- **Code Coverage**: Common code coverage metrics include statement coverage, branch coverage, block coverage, and method coverage.
- **Coverage Requirement**: A single unit of a coverage metric.
- A test is rated by the specific requirements that it covers.

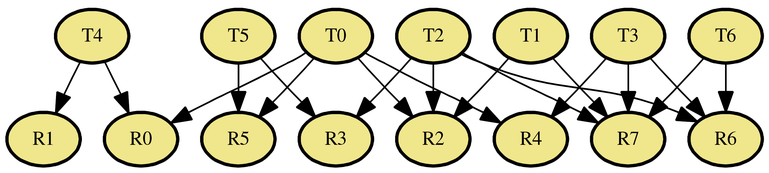

Figure 1: Test Suite: {T0, … , T6} are tests and {R0, … , R7} are requirements. Arrows represent coverage.

- The coverage of the tests can be examined to see where there is **coverage overlap**.
- Some tests can be eliminated from the test suite without changing the total coverage.

Call Trees

- The tool captures the dynamic behavior of a program as a call tree.
  - **Call Tree**: tree-based representation of the method calls of a program's execution during testing.

- Each **path** in the call tree from the root node to a leaf node is considered a requirement.
- The presented algorithms are implemented using these call tree paths as requirements.

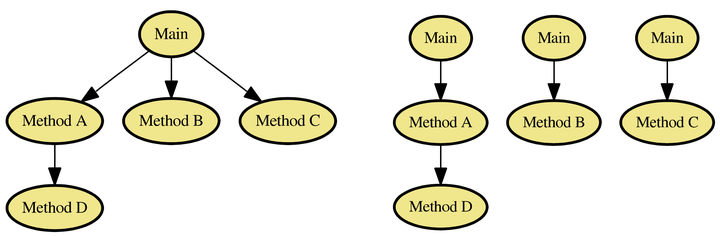

Figure 2: Call Trees and Call Tree Paths: On the left, an example of a call tree; the Main method calls Methods A, B, and C, and Method A calls Method D. On the right, the individual call tree paths of the tree on the left.

Reduction Techniques

- Four algorithms were implemented for this tool: **Traditional Greedy**, **2-Optimal**, **Harrold, Gupta, Soffa**, and **Delayed Greedy**.

Traditional Greedy Algorithm

- Tests covering more requirements than other tests are desired.
- Choose the tests that cover the most requirements.

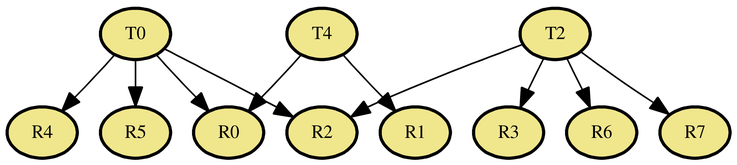

Figure 3: The results of running the traditional greedy algorithm on the data in Figure 1.

2-Optimal Algorithm

- 2-Optimal is a step towards brute force search.
- Compare every pair to every other pair of tests.
- Generalizes to K-Way.

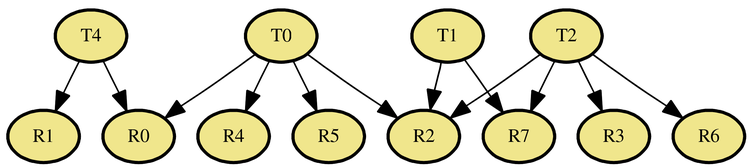

Figure 4: The results of running the 2-Optimal greedy algorithm on the data in Figure 1.

Harrold, Gupta, Soffa Algorithm

- Every requirement must be covered in order to maintain coverage.
- For requirements that are covered by the fewest tests, those tests have a high probability of being chosen.
- First choose from the tests that are more likely to be chosen, and then choose the tests that are less likely.

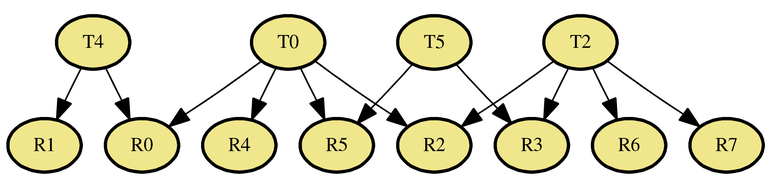

Figure 5: The results of running the Harrold, Gupta, Soffa algorithm on the data in Figure 1.

Delayed Greedy Algorithm

- A test whose set of covered requirements is a subset of another test's set of covered requirements does not need to be considered.
- A requirement whose set of covering tests is a superset of another requirement's set of covering tests does not need to be considered.
- If a requirement is only covered by one test, choose that test.
- If there are no requirements covered by only one test, greedily choose a test based on coverage.

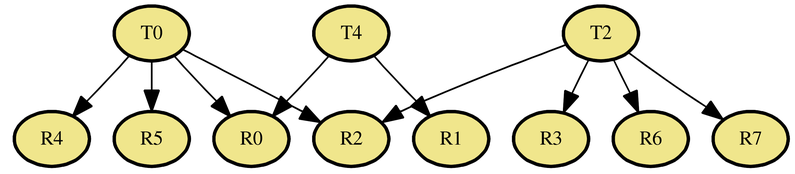

Figure 6: The results of running the delayed greedy algorithm on the data in Figure 1.

Prioritization Methods

- Each reduction algorithm produces a subset of the original test suite.
- The reduction algorithm can be repeated on the set of tests that did not get chosen. The output from the repeated execution of the algorithm can be added to the output from the previous.
- By running the reduction algorithm on the residual tests until every test has been chosen, a reordering of the test will be obtained that covers all of the requirements faster than the original test suite.

Discussion

- Given Figure 1 as input, the algorithms produced their respective figures.
- Each algorithm can produce different results for the same input.
- **Metrics for Reduction:** test suite size, execution time, and the amount of overlap.
- **Metric for Prioritization:** the ratio of the cumulative coverage of a prioritized test suite and an optimal test suite.
- Reduction and prioritization save developers time and money.

Contributions and Future Work

- **A Tool Paper:** *Test Suite Reduction and Prioritization with Call Trees,* was published at *The 22nd IEEE/ACM International Conference on Automated Software Engineering.*
- **The Tool:** The source code will be released by spring 2008.
- **Senior Thesis Research:**
  - Empirically evaluate the efficiency and effectiveness of the algorithms using both real world and synthetic test suites.
  - Compare the existing algorithms to search-based methods that use genetic algorithms, simulated annealing, and hill climbing.
  - Incorporate test costs into the test suite reduction and prioritization methods.