Exploring Time-Aware Test Suite Prioritization

Mary Lou Soffa
University of Virginia

Collaborators:
Kristen Walcott
Gregory M. Kapfhammer,
Allegheny College
Regression testing

Software constantly modified
- Bug fixes
- Addition of functionality

After changes, regression testing – run test case in test suite and provide more
- Provides confidence modifications correct
- Helps find new error

Large number of test cases – continues to grow
- Weeks/months to run entire test suite
- Costs high – ½ cost of maintenance
Reducing cost regression testing

- To reduce cost, do not run all test cases – prioritize tests i.e., reorder them

Test Prioritization Techniques

- Original order
- Based on fault detection ability
- Analysis to determine what test cases affected by change and order
- Random selection – order tests randomly
- Reverse – run tests in reverse order
Example – after prioritization

But, retesting usually has a time budget – based on time, was the above order the best order?

Contribution: A test prioritization technique that intelligently incorporates the test time budget
## Fault Matrix Example

<table>
<thead>
<tr>
<th>FAULTS/TEST CASE</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T1$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$T2$</td>
<td>$\times$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T3$</td>
<td>$\times$</td>
<td></td>
<td></td>
<td>$\times$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T4$</td>
<td>$\times$</td>
<td>$\times$</td>
<td></td>
<td></td>
<td>$\times$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T5$</td>
<td></td>
<td></td>
<td>$\times$</td>
<td>$\times$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T6$</td>
<td>$\times$</td>
<td>$\times$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Given modified program, have 6 test cases

Assume a priori knowledge of faults, $f$
### Test Suite

**Faults and Time**

<table>
<thead>
<tr>
<th>Test</th>
<th>#faults</th>
<th>Time costs</th>
<th>avg faults/min</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>7</td>
<td>9</td>
<td>0.778</td>
</tr>
<tr>
<td>T2</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>T3</td>
<td>2</td>
<td>3</td>
<td>0.667</td>
</tr>
<tr>
<td>T4</td>
<td>3</td>
<td>4</td>
<td>0.75</td>
</tr>
<tr>
<td>T5</td>
<td>3</td>
<td>4</td>
<td>0.75</td>
</tr>
<tr>
<td>T6</td>
<td>3</td>
<td>4</td>
<td>0.75</td>
</tr>
</tbody>
</table>

Tests vary according to the time overhead and their ability to reveal faults.

**GOAL:** When testing, find as many faults as soon as possible.
Fault – aware Prioritization - Time limit 12 minutes

Fault based order

7 faults found in 9 minutes
Naïve time-Based prioritization

- **Original Order**

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>T2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>T4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>T5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>T6</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Naïve time based order

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>T4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>T5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>T6</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>T1</td>
<td>9</td>
<td>7</td>
</tr>
</tbody>
</table>

8 faults in 12 minutes
Average Percent Fault Detection - Based Prioritization

Original Order

- **T1**
  - Time: 9
  - APFD: 0.8
  - Faults: 7

- **T2**
  - Time: 1
  - APFD: 1.0
  - Faults: 1

- **T3**
  - Time: 3
  - APFD: 0.7
  - Faults: 3

- **T4**
  - Time: 4
  - APFD: 0.8
  - Faults: 3

- **T5**
  - Time: 4
  - APFD: 0.8
  - Faults: 3

- **T6**
  - Time: 4
  - APFD: 8.8
  - Faults: 4

APFD

- **T2**
  - Time: 1
  - APFD: 0.8
  - Faults: 1

- **T1**
  - Time: 9
  - APFD: 8.8
  - Faults: 7

- **T4**
  - Time: 4
  - APFD: 0.8
  - Faults: 3

- **T5**
  - Time: 4
  - APFD: 0.8
  - Faults: 3

- **T6**
  - Time: 4
  - APFD: 8.8
  - Faults: 4

- **T3**
  - Time: 3
  - APFD: 0.8
  - Faults: 4

7 faults in 10 minutes
Intelligent Time-aware prioritization

- **Original order**

  - **T1**
    - Time: 9
    - Faults: 7

  - **T2**
    - Time: 1
    - Faults: 1

  - **T3**
    - Time: 3
    - Faults: 4

  - **T4**
    - Time: 4
    - Faults: 3

  - **T5**
    - Time: 4
    - Faults: 3

  - **T6**
    - Time: 4
    - Faults: 3

  8 faults in 11 minutes
Comparing Test Prioritization

- Intelligent scheme performs better – finding most faults in shortest time
- Considers testing time budget and overlapping fault detection of test
- Time-aware prioritization requires heuristic solution to NP-complete
- Use genetic algorithm
  - Fitness function based on code coverage for ability to find faults and time
Infrastructure

Program Under Test (P) → Test Transformer

Coverage Calculator

Fitness Value Producer → Test Transformer

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number tuples per iteration</td>
<td>Mutation probability</td>
</tr>
<tr>
<td>Maximum # iterations</td>
<td>Addition/deletion properties</td>
</tr>
<tr>
<td>% of test suite execution time</td>
<td>Test adequacy criteria</td>
</tr>
<tr>
<td>Crossover probability</td>
<td>Program coverage weight</td>
</tr>
</tbody>
</table>

Test Transformer → Test Suite

New Test suite

Test Reorder
Fitness Function

- Since fault information unknown, use method and block coverage to measure test suite potential
- Coverage is aggregated for entire test suite
- Test prioritization fitness measures
  - The percentage of $P$’s code that is covered by $T_i$
  - The time at which each test case covers code within $P$ – can use percentages of code coverage
Change the order of test cases

- Develop smaller test suites based on operators that change
  - Order
  - Test cases included

Fitness evaluation determines goodness of the changed suite.
Crossover Operator

- Vary test prioritizations by recombination at a randomly chosen crossover point
Addition and Deletion Operators

- **Operators**
  - Add operator
  - Delete operator

- **Selected Test suite**

- **Entire test suite**
Another way to add variation to create new population

Test cases are mutated –
  replaced by an unused test case
  Swap test cases if no unused test case
Experiment Goals and Design

- **Determine if the GA-produced prioritizations, on average, outperform a selected set of other prioritizations**
- **Identify overhead - time and space - associated with the creation of the prioritized test suite**
Experiments

- Block or method coverage
- Order
  - Initial order
  - Reverse order
  - Random order
  - Fault-aware prioritization
Experimental Design

- GNU/Linux Workstation – 1.80 GHz Intel Pentium and 1GB of main memory
- Used JUnit to prioritize test cases
- Seeded faults: 25%, 50%, 75% of 40 faults
- Used Emma to compute coverage criteria
- 2 Case studies
  - Gradebook
  - JDepend – traverse directories of Java class files
Test Adequacy Metrics

- **Method coverage**
  - Considered covered when entered

- **Basic block coverage**
  - A sequence of byte code instructions without any jumps or jump targets
  - Considered covered when entered

- How much of the code has been executed – used 100%
APFD Results for Block and Method Coverage

Code Coverage: Block vs. Method

11% better Gradebook
13% better JDepend
Prioritization Efficiency

User Time Overhead

Time(s)

13.8 hours

8.3 hours

Space costs insignificant

Number of Generations, Population Size
Gradebook: Intelligent vs Random

Gradebook Prioritization: GA vs. Random

APFD

(Percent of Total Time, Number of Faults)
JDdepend: Intelligent vs Random

![Graph showing JDepend Prioritization: GA vs. Random]

- **APFD**
- **GA Tuple**
- **Random Tuple**

(Percent of Total Time, Number of Faults)
Comparisons with other orders

- Experiments to compare with other types of prioritizations
- Original
- Reverse
- Fault aware (impossible to implement)
- Time aware
**APFD Metric**

Let $T$ be the test suite under evaluation, $g$ the number of faults contained in the program under test $P$, $n$ the total number of test cases, and $\text{reveal}(i,T)$ the position of the first test in $T$ that exposes fault $i$.

\[
\text{APFD}(T,P) = 1 - \frac{\sum_{i=1}^{g} \text{reveal}(i,T)}{ng} + \frac{1}{2n}.
\]
<table>
<thead>
<tr>
<th>$Pi$</th>
<th>$Fi$</th>
<th>Initial</th>
<th>Reverse</th>
<th>Fault aware</th>
<th>$GA$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>10</td>
<td>-0.6</td>
<td>-0.2</td>
<td>0.7</td>
<td>0.4</td>
</tr>
<tr>
<td>0.25</td>
<td>20</td>
<td>-0.9</td>
<td>-0.2</td>
<td>0.7</td>
<td>0.4</td>
</tr>
<tr>
<td>0.25</td>
<td>30</td>
<td>-0.9</td>
<td>-0.0</td>
<td>0.5</td>
<td>0.6</td>
</tr>
<tr>
<td>0.50</td>
<td>10</td>
<td>-0.04</td>
<td>0.1</td>
<td>0.9</td>
<td>0.7</td>
</tr>
<tr>
<td>0.50</td>
<td>20</td>
<td>-0.2</td>
<td>0.2</td>
<td>0.9</td>
<td>0.7</td>
</tr>
<tr>
<td>0.50</td>
<td>30</td>
<td>-0.3</td>
<td>0.3</td>
<td>0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>0.75</td>
<td>10</td>
<td>0.3</td>
<td>0.5</td>
<td>0.9</td>
<td>0.9</td>
</tr>
<tr>
<td>0.75</td>
<td>20</td>
<td>0.1</td>
<td>0.4</td>
<td>0.9</td>
<td>0.7</td>
</tr>
<tr>
<td>0.75</td>
<td>30</td>
<td>0.04</td>
<td>0.5</td>
<td>0.9</td>
<td>0.7</td>
</tr>
</tbody>
</table>
Results

- Comparison of
  - Original
  - Fault-aware (impossible to implement)
  - Reverse
  - Gradebook
    - 120% better than original
    - Time aware better than original
- JDepend
  - Produced better results
Technique Enhancements

- Make fitness calculation faster
  - Eliminate the majority of coverage cover overlap by reducing the test suite
  - Record coverage on a per-test basis
- Distribute execution of fitness function
- Exploit test execution histories and favor tests that have recently revealed faults
- Terminate the genetic algorithm when it achieves fitness equivalent to previous prioritizations
Conclusions and Future Work

- Contribution: a test prioritization technique that includes the testing time budget
- Time-aware prioritization can yield a 120% improvement in APFD when compared to alternative prioritizations
- Different heuristics - analysis
Paper to appear

- *International Symposium on Software Testing and Analysis (ISSTA)*
- *July, 2006*