Using Synthetic Coverage Information to Evaluate Test Suite Prioritizers

Gregory M. Kapfhammer†

Department of Computer Science
Allegheny College, Pennsylvania, USA
http://www.cs.allegheny.edu/~gkapfham/

Chennai Mathematical Institute, February 2008

† In Conjunction with Mary Lou Soffa, Kristen Walcott (UVa/CS)
Suvarshi Bhadra, Joshua Geiger, Adam Smith, Gavilan Steinman, Yuting Zhang (Allegheny/CS)

Featuring images from Embroidery and Tapestry Weaving, Grace Christie (Project Gutenberg)
Using Synthetic Coverage Information to Evaluate Test Suite Prioritizers
The Challenge of Software Testing

I shall not deny that the construction of these testing programs has been a major intellectual effort: to convince oneself that one has not overlooked “a relevant state” and to convince oneself that the testing programs generate them all is no simple matter. The encouraging thing is that (as far as we know!) it could be done.


Additional Challenge: empirically evaluating the efficiency and effectiveness of software testing techniques
**Important Contributions**

- **Synthetic Coverage Generators**
- **Detailed Experimental Results**

A comprehensive framework that supports the empirical evaluation of regression test suite prioritizers.

Using Synthetic Coverage Information to Evaluate Test Suite Prioritizers
Testing isolates defects and establishes a confidence in the correctness of a software application.
What is a Test Case?

Overview

- Test suite executor runs each test case **independently**
- Each test invokes a method within the program and then compares the **actual** and **expected** output values
Overview

- Structural **adequacy criteria** focus on the coverage of nodes, edges, paths, and definition-use associations
- Instrumentation **probes** track the coverage of test requirements
Finding the Overlap in Coverage

- $R_j \rightarrow T_i$ means that requirement $R_j$ is **covered by** test $T_i$
- $T = \langle T_2, T_3, T_6, T_9 \rangle$ covers **all** of the test requirements
- Include the **remaining** tests so that they can **redundantly** cover the requirements
Overview

- Prioritization **re-orders** the tests so that they cover the requirements more effectively.

- Researchers and practitioners need to determine whether the **prioritized** test suite is better than the **original** ordering.
Using Synthetic Coverage Information to Evaluate Test Suite Prioritizers
Using Real World Applications

Regression Test Suite

Program Under Test

It is difficult to **systematically** study the efficiency and effectiveness trade-offs because coverage overlap **varies**
Using Synthetic Coverage Information to Evaluate Test Suite Prioritizers

$\text{Prioritize to } \textbf{increase} \text{ the CE of a test suite } CE = \frac{\text{Actual}}{\text{Ideal}} \in [0, 1]$
### Characterizing a Test Suite

#### Test Information

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Cost (sec)</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>5</td>
<td>✓ ✓</td>
</tr>
<tr>
<td>$T_2$</td>
<td>10</td>
<td>✓ ✓ ✓</td>
</tr>
<tr>
<td>$T_3$</td>
<td>4</td>
<td>✓ ✓ ✓ ✓ ✓</td>
</tr>
</tbody>
</table>

Total Testing Time = 19 seconds

#### Formulating the Metrics

$CE$ considers the **execution time** of each test while $CE_u$ assumes that all test cases execute for a **unit cost**
## Coverage Effectiveness Values

### Calculating $CE$ and $CE_u$

<table>
<thead>
<tr>
<th>Ordering</th>
<th>CE</th>
<th>$CE_u$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1 T_2 T_3$</td>
<td>.3789</td>
<td>.4</td>
</tr>
<tr>
<td>$T_1 T_3 T_2$</td>
<td>.5053</td>
<td>.4</td>
</tr>
<tr>
<td>$T_2 T_1 T_3$</td>
<td>.3789</td>
<td>.5333</td>
</tr>
<tr>
<td>$T_2 T_3 T_1$</td>
<td>.4316</td>
<td>.6</td>
</tr>
<tr>
<td>$T_3 T_1 T_2$</td>
<td>.5789</td>
<td>.4557</td>
</tr>
<tr>
<td>$T_3 T_2 T_1$</td>
<td>.5789</td>
<td>.5333</td>
</tr>
</tbody>
</table>

### Observations

- Including test case costs does impact the CE metric
- Depending upon the characteristics of the test suite, we may see $CE = CE_u$, $CE > CE_u$, or $CE < CE_u$
Does this result **generalize** to other applications?
Presentation Outline

1. Challenges and Solutions
2. Regression Testing Techniques
3. Conducting Empirical Studies
4. Generating Synthetic Coverage
5. Empirical Evaluation
6. Future Work
7. Conclusions
Test Suites and Requirements

**Regression Test Suite**

\[ T = \langle T_1, \ldots, T_n \rangle \]
\[ T_i \in T \]

**Test Requirements**

\[ R = \{ R_1, \ldots, R_m \} \]
\[ R_j \in R \]

covers\( (i) \) denotes the set of requirements that \( T_i \) covers

coveredby\( (j) \) denotes the set of test cases that cover \( R_j \)

**Goal:** automatically generate a synthetic regression test suite \( T \) that covers the requirements in \( R \)
Coverage Overlap Metrics

\[
NCO(i, k) = (R \setminus \text{covers}(i)) \cap (R \setminus \text{covers}(k))
\]

\[
NCO(1, 2) = \{R_7, R_8, R_9\}
\]

\[
JCO(i, k) = \text{covers}(i) \cap \text{covers}(k)
\]

\[
JCO(1, 2) = \{R_2, R_3, R_4\}
\]

\[
TCO(i, k) = NCO(i, k) \cup JCO(i, k)
\]

\[
TCO(1, 2) = \{R_2, R_3, R_4, R_7, R_8, R_9\}
\]
Standard Coverage Generation

**Generation Procedure**

- Guarantee that **each requirement** is covered by a test case and that **all tests** cover at least one requirement.

- **Balance** the coverage information according to the **cardinality** of either the $covers(i)$ or the $coveredby(j)$ sets.
Configuring the Standard Generator

<table>
<thead>
<tr>
<th>Number</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tests (n)</td>
<td>10</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>Requirements (m)</td>
<td>(2 \times n)</td>
<td>(5 \times n)</td>
<td>(10 \times n)</td>
</tr>
<tr>
<td>Coverage Points (p)</td>
<td>((n \times m)/5)</td>
<td>((n \times m)/3)</td>
<td>((n \times m)/2)</td>
</tr>
</tbody>
</table>

Generating Coverage

- Configuration **sss** generates 10 tests, 20 requirements, and 40 coverage points.
- Configuration **iii** generates 100 tests, 1000 requirements, and 50,000 coverage points.
- For all of the above configurations, the generation procedure consumes less than one second of execution time.
“Greedy Fooling” Coverage Generation

The **greedy** test prioritizer iteratively selects test cases according to the (coverage / cost) ratio

**Goal**: generate coverage and timing information that will **fool** the greedy technique into creating $T' = \langle T_n, \ldots, T_1 \rangle$ even though $CE(T') < CE(T)$ for $T = \langle T_1, \ldots, T_n \rangle$

**Inspiration**: Vazirani’s construction of a **tight example** for the greedy **minimal set cover** algorithm
Constructing “Greedy Fooling” Test Suites

**Approach:** use one dimensional **optimization** (e.g., golden section search and successive parabolic interpolation) to pick a value for $\text{cost}(T_n)$

**Construction:** set $\text{cost}(T_1) = \text{cost}(T_2) = \text{cost}(T_3) = 1$ and then determine the bounds for $\text{cost}(T_4) \in [C_{\text{min}}, C_{\text{max}}]$

**Example:** $\text{cost}(T_4) \in [2.138803, 2.472136]$ so that

$$CE_{\text{min}}(T') = 0.5838004 \quad CE_{\text{min}}(T) = 0.6108033$$

$$CE_{\text{max}}(T') = 0.5482172 \quad CE_{\text{max}}(T) = 0.6345125$$
Using Synthetic Coverage Information to Evaluate Test Suite Prioritizers
Increasing the value of $p$ changes the coverage overlap metrics.
Increasing test suite size **tightens** the coverage overlap metrics.
The generation of a small test suite takes less than 3 seconds.
Greedy Fooling Time - Large Test Suite

The generation of a large test suite takes up to 50 seconds.
Finding a bound for $\text{cost}(T_n)$ requires few iterations of the optimizer.
Increasing the value of $n$ does not markedly increase the iteration count.
The cost of generation is dominated by numerical integration’s cost.
### Fooling the Greedy Prioritizer

Using Synthetic Coverage Information to Evaluate Test Suite Prioritizers

<table>
<thead>
<tr>
<th>n</th>
<th>$C_{min}$</th>
<th>$C_{max}$</th>
<th>$CE_{min}(T')$</th>
<th>$CE_{max}(T')$</th>
<th>$CE_{min}(T)$</th>
<th>$CE_{max}(T)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5.2786</td>
<td>8.541</td>
<td>0.63031</td>
<td>0.51308</td>
<td>0.64983</td>
<td>0.71519</td>
</tr>
<tr>
<td>20</td>
<td>10.1320</td>
<td>18.885</td>
<td>0.65222</td>
<td>0.50150</td>
<td>0.65670</td>
<td>0.73680</td>
</tr>
<tr>
<td>30</td>
<td>15.1970</td>
<td>28.967</td>
<td>0.65616</td>
<td>0.50000</td>
<td>0.66076</td>
<td>0.74138</td>
</tr>
<tr>
<td>40</td>
<td>20.2630</td>
<td>38.622</td>
<td>0.65809</td>
<td>0.50243</td>
<td>0.66256</td>
<td>0.74239</td>
</tr>
<tr>
<td>50</td>
<td>25.3290</td>
<td>48.936</td>
<td>0.65922</td>
<td>0.50000</td>
<td>0.66354</td>
<td>0.74490</td>
</tr>
<tr>
<td>60</td>
<td>30.0610</td>
<td>58.723</td>
<td>0.66246</td>
<td>0.50117</td>
<td>0.66320</td>
<td>0.74514</td>
</tr>
<tr>
<td>70</td>
<td>35.1090</td>
<td>68.510</td>
<td>0.66276</td>
<td>0.50178</td>
<td>0.66377</td>
<td>0.74551</td>
</tr>
<tr>
<td>80</td>
<td>40.1240</td>
<td>78.948</td>
<td>0.66318</td>
<td>0.50000</td>
<td>0.66429</td>
<td>0.74684</td>
</tr>
<tr>
<td>90</td>
<td>45.1400</td>
<td>88.816</td>
<td>0.66347</td>
<td>0.50000</td>
<td>0.66448</td>
<td>0.74693</td>
</tr>
<tr>
<td>100</td>
<td>50.1550</td>
<td>98.684</td>
<td>0.66374</td>
<td>0.50000</td>
<td>0.66460</td>
<td>0.74707</td>
</tr>
</tbody>
</table>
Using Synthetic Coverage Information to Evaluate Test Suite Prioritizers
Use **heuristic search** (HC, SANN, GA) to prioritize the test suite
Systematically study the efficiency and effectiveness trade-offs with synthetic coverage and then conduct further experimental studies with real world applications.
Concluding Remarks

A comprehensive framework that furnishes a new perspective on the empirical evaluation of regression test suite prioritizers

http://www.cs.allegheny.edu/~gkapfham/