Can Search-Based Prioritizers Improve the Coverage Effectiveness of Regression Test Suites?

Alexander P. Conrad and Gregory M. Kapfhammer

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

Centre for Research on Evolution, Search, and Testing
King’s College London, September 2008
Can Search-Based Prioritizers Improve the Coverage Effectiveness of Regression Test Suites?

**Important Contributions**

**Search-Based Prioritizers**

**Empirical Results**

**Design, implement** and empirically **evaluate** test suite prioritizers that use **order-based** genetic algorithms to find test orderings that maximize **coverage effectiveness**
Can Search-Based Prioritizers Improve the Coverage Effectiveness of Regression Test Suites?

- Structural **adequacy criteria** focus on the coverage of nodes, edges, call tree paths, and/or 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
- Test suite reduction discards the redundant test cases while prioritization re-orders all of the tests
Prioritization re-orders the tests so that they cover the requirements more effectively

GRT uses the same prioritization across multiple runs of the test suite whereas VSRT creates a new prioritization for each test run
Can Search-Based Prioritizers Improve the Coverage Effectiveness of Regression Test Suites?

- Prioritize to increase 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>$R_1$, $R_2$</td>
</tr>
<tr>
<td>$T_2$</td>
<td>10</td>
<td>$R_1$, $R_2$, $R_3$</td>
</tr>
<tr>
<td>$T_3$</td>
<td>4</td>
<td>$R_1$, $R_3$, $R_4$, $R_5$</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**
### Varying 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 $CE < CE_u$
Standard Greedy Approaches

Consider (i) **greedy choices** (cost, coverage, and ratio) and (ii) **algorithms**

- Harrold, Gupta, Soffa (HGS)
- Delayed Greedy (DGR)
- Traditional Greedy (GRD)
- 2-Optimal Greedy (2OPT)

Repeatedly invoking a **reduction** algorithm enables the **prioritization** of a test suite
**Greedy Choices Impact Effectiveness**

### Table: Prioritization Techniques

<table>
<thead>
<tr>
<th></th>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
<th>$R_4$</th>
<th>$R_5$</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>$T_2$</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>$T_3$</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>$T_4$</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

### Greedy-by

<table>
<thead>
<tr>
<th>Greedy-by</th>
<th>$T_r$</th>
<th>$time(T_r)$</th>
<th>$T_p$</th>
<th>CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>coverage</td>
<td>$\langle T_1, T_4 \rangle$</td>
<td>5</td>
<td>$\langle T_1, T_4, T_2, T_3 \rangle$</td>
<td>0.400</td>
</tr>
<tr>
<td>time</td>
<td>$\langle T_2, T_3, T_4 \rangle$</td>
<td>3</td>
<td>$\langle T_2, T_3, T_4, T_1 \rangle$</td>
<td>0.714</td>
</tr>
<tr>
<td>ratio</td>
<td>$\langle T_2, T_4, T_3 \rangle$</td>
<td>3</td>
<td>$\langle T_2, T_4, T_3, T_1 \rangle$</td>
<td>0.743</td>
</tr>
</tbody>
</table>
Questions: Do the greedy prioritizers find test orderings that maximize coverage effectiveness? Is there a need for search-based approaches?
Motivating Empirical Results

Using **ratio** and **time** improves the CE of the prioritized test suite

Can Search-Based Prioritizers Improve the Coverage Effectiveness of Regression Test Suites?
### Comparison to Reverse and Initial

<table>
<thead>
<tr>
<th>Technique</th>
<th>GCM</th>
<th>REV</th>
<th>INIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2OPT</td>
<td>cost</td>
<td>JD</td>
<td>-</td>
</tr>
<tr>
<td>2OPT</td>
<td>coverage</td>
<td>RP, RM, JD</td>
<td>SK</td>
</tr>
<tr>
<td>2OPT</td>
<td>ratio</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DGR</td>
<td>all</td>
<td>RM, RP, JD</td>
<td>SK</td>
</tr>
<tr>
<td>GRD</td>
<td>cost</td>
<td>JD</td>
<td>-</td>
</tr>
<tr>
<td>GRD</td>
<td>coverage</td>
<td>RM, RP, JD</td>
<td>SK</td>
</tr>
<tr>
<td>GRD</td>
<td>ratio</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>HGS</td>
<td>all</td>
<td>RM, RP, JD</td>
<td>RP, SK, TM</td>
</tr>
</tbody>
</table>

**Important Observation:** Overall, the greedy prioritizers do worse than REV or INIT 44.7% of the time. Yet, GRD and 2OPT do well – are there test suites that will cause these prioritizers to do poorly?
"Greedy Fooling" Coverage Generation

**Generation Procedure**

- **The greedy** test prioritizer iteratively selects test cases according to the \((\text{coverage} / \text{cost})\) 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
Approach: use one dimensional optimization (e.g., golden section search and successive parabolic interpolation or genetic algorithms) to pick a value for $cost(T_n)$

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

Example: $cost(T_4) \in [2.138803, 2.472136]$ so that $CE_{min}(T') = .5838004 \quad CE_{min}(T) = .6108033$ $CE_{max}(T') = .5482172 \quad CE_{max}(T) = .6345125$
Search-based approaches may be able to do better than the greedy prioritizers.

**Data Structures**
- Chromosome
- Individual
- Population

**Functions**
- Selection Operator
- Crossover Operator
- Mutation Operator
- Termination Condition
- Fitness Function

Can Search-Based Prioritizers Improve the Coverage Effectiveness of Regression Test Suites?
**Partially-Mapped Crossover (PMX)**

**Pick the Cutpoints**
- \( T = \langle T_1, T_2, T_3, |T_4, T_5, T_6, |T_7, T_8 \rangle \)
- \( T' = \langle T_3, T_7, T_5, |T_1, T_6, T_8, |T_2, T_4 \rangle \)

**Construct the Mappings**
- \( T_4 \leftrightarrow T_1, T_5 \leftrightarrow T_6 \), and \( T_6 \leftrightarrow T_8 \)

**Create the Children**
- \( T_c = \langle T_4, T_2, T_3, |T_1, T_6, T_8, |T_7, T_5 \rangle \)
- \( T'_c = \langle T_3, T_7, T_8, |T_4, T_5, T_6, |T_2, T_1 \rangle \)
Pick the Cutpoints

- \( T = \langle T_1, T_2, T_3, |T_4, T_5, T_6, |T_7, T_8 \rangle \)
- \( T' = \langle T_3, T_7, T_5, |T_1, T_6, T_8, |T_2, T_4 \rangle \)

Construct the Mappings

- \( T_4 \leftrightarrow T_1, T_5 \leftrightarrow T_6 \), and \( T_6 \leftrightarrow T_8 \)

Create the Children

- \( T_c = \langle T_4, T_2, T_3, |T_1, T_6, T_8, |T_7, T_5 \rangle \)
- \( T'_c = \langle T_3, T_7, T_8, |T_4, T_5, T_6, |T_2, T_1 \rangle \)

Can Search-Based Prioritizers Improve the Coverage Effectiveness of Regression Test Suites?
Partially-Mapped Crossover (PMX)

**Pick the Cutpoints**
- \( T = \langle T_1, T_2, T_3, | T_4, T_5, T_6, | T_7, T_8 \rangle \)
- \( T' = \langle T_3, T_7, T_5, | T_1, T_6, T_8, | T_2, T_4 \rangle \)

**Construct the Mappings**
- \( T_4 \leftrightarrow T_1, T_5 \leftrightarrow T_6 \), and \( T_6 \leftrightarrow T_8 \)

**Create the Children**
- \( T_c = \langle T_4, T_2, T_3, | T_1, T_6, T_8, | T_7, T_5 \rangle \)
- \( T'_c = \langle T_3, T_7, T_8, | T_4, T_5, T_6, | T_2, T_1 \rangle \)
Order Crossover (OX1)

**Pick the Cutpoints**

- \( T = \langle T_1, T_2, | T_3, T_4, T_5, | T_6, T_7, T_8 \rangle \)
- \( T' = \langle T_2, T_4, | T_6, T_8, T_7, | T_5, T_3, T_1 \rangle \)

**Copy the Cores**

- \( T_c = \langle \Box, \Box, | T_6, T_8, T_7, | \Box, \Box, \Box \rangle \)
- \( T'_c = \langle \Box, \Box, | T_3, T_4, T_5, | \Box, \Box, \Box \rangle \)

**Create the Children**

- \( T_c = \langle T_4, T_5, | T_6, T_8, T_7, | T_1, T_2, T_3 \rangle \)
- \( T'_c = \langle T_8, T_7, | T_3, T_4, T_5, | T_1, T_2, T_6 \rangle \)
Order Crossover (OX1)

Pick the Cutpoints
- \( T = \langle T_1, T_2, | T_3, T_4, T_5, | T_6, T_7, T_8 \rangle \)
- \( T' = \langle T_2, T_4, | T_6, T_8, T_7, | T_5, T_3, T_1 \rangle \)

Copy the Cores
- \( T_c = \langle \Box, \Box, | T_6, T_8, T_7, | \Box, \Box, \Box \rangle \)
- \( T'_c = \langle \Box, \Box, | T_3, T_4, T_5, | \Box, \Box, \Box \rangle \)

Create the Children
- \( T_c = \langle T_4, T_5, | T_6, T_8, T_7, | T_1, T_2, T_3 \rangle \)
- \( T'_c = \langle T_8, T_7, | T_3, T_4, T_5, | T_1, T_2, T_6 \rangle \)
Order Crossover (OX1)

**Pick the Cutpoints**

- \( T = \langle T_1, T_2, | T_3, T_4, T_5, | T_6, T_7, T_8 \rangle \)
- \( T' = \langle T_2, T_4, | T_6, T_8, T_7, | T_5, T_3, T_1 \rangle \)

**Copy the Cores**

- \( T_c = \langle \Box, \Box, | T_6, T_8, T_7, | \Box, \Box, \Box \rangle \)
- \( T'_c = \langle \Box, \Box, | T_3, T_4, T_5, | \Box, \Box, \Box \rangle \)

**Create the Children**

- \( T_c = \langle T_4, T_5, | T_6, T_8, T_7, | T_1, T_2, T_3 \rangle \)
- \( T'_c = \langle T_8, T_7, | T_3, T_4, T_5, | T_1, T_2, T_6 \rangle \)
Additional Crossover Operators

**CX** - Cycle, **OX2** - Order-based, **MPX** - Maximal Preservation, **POS** - Position-based, **VR** - Voting Recombination

Systematically study the efficiency and effectiveness trade-offs with the different crossover operators.

**Question**: which ones are the best?
Mutation Operators (DM and ISM)

Displacement Mutation (DM)

- **Pick:** $T = \langle T_1, T_2, | T_3, T_4, T_5, | T_6, T_7, T_8 \rangle$
- **Chop:** $T' = \langle T_1, T_2, T_6, T_7, T_8 \rangle$
- **Fill:** $T' = \langle T_1, T_2, T_6, _T_7, T_3, T_4, T_5, T_8 \rangle$

Insertion Mutation (ISM)

- **Pick:** $T = \langle T_1, T_2, T_3, T_4, T_5, T_6, _T_7, T_8 \rangle$
- **Chop:** $T = \langle T_1, T_2, T_3, T_4, T_5, T_6, T_8 \rangle$
- **Fill:** $T' = \langle T_1, _T_2, T_7, T_6, T_3, T_4, T_5, T_8 \rangle$

Can Search-Based Prioritizers Improve the Coverage Effectiveness of Regression Test Suites?
Mutation Operators (DM and ISM)

**Displacement Mutation (DM)**

- **Pick:** $T = \langle T_1, T_2, | T_3, T_4, T_5, | T_6, T_7, T_8 \rangle$
- **Chop:** $T' = \langle T_1, T_2, T_6, T_7, T_8 \rangle$
- **Fill:** $T' = \langle T_1, T_2, T_6, \underline{T_7}, T_3, T_4, T_5, T_8 \rangle$

**Insertion Mutation (ISM)**

- **Pick:** $T = \langle T_1, T_2, T_3, T_4, T_5, T_6, \underline{T_7}, T_8 \rangle$
- **Chop:** $T = \langle T_1, T_2, T_3, T_4, T_5, T_6, T_8 \rangle$
- **Fill:** $T' = \langle T_1, \underline{T_2}, T_7, T_6, T_3, T_4, T_5, T_8 \rangle$

Can Search-Based Prioritizers Improve the Coverage Effectiveness of Regression Test Suites?
Experiment Goals and Design

**Metrics:**
- Runtime of prioritization technique
- Coverage effectiveness of final test ordering

**Applications:**
- 9 real-world
- 54 synthetic

**Configurations:**
- 10,206 configurations
- 91,854 real-world experiments
- 551,124 synthetic experiments

---

Can Search-Based Prioritizers Improve the Coverage Effectiveness of Regression Test Suites?
Across applications, we find variability in efficiency and effectiveness
Can Search-Based Prioritizers Improve the Coverage Effectiveness of Regression Test Suites?

Random search outperforms the order-based genetic algorithm.
Empirical Results: Small Application (SK)

While many operators do not outperform random search, PMX does.
Except for OX1/OX2, many operators do not outperform random search.
Can Search-Based Prioritizers Improve the Coverage Effectiveness of Regression Test Suites?
Can Search-Based Prioritizers Improve the Coverage Effectiveness of Regression Test Suites?

Empirical Results: Large Application (RP)

Execution Time (in ms)

Coverage Effectiveness

Operators do better than the random search, yet some are costly
Separation begins to appear between POS/CX and random search.
Future Empirical Analysis

Detailed Statistical Analysis

Additional Applications

After including **greedy** prioritizers, use the **Kruskal-Wallis** rank sum test or **ANOVA** and then **Bonferroni** or **Tukey HSD** adjustments in order to determine statistical **significance**.
Use **mutation** faults (FOM and HOM) when you calculate **APFD** during search-based prioritization.

**Question**: does a high APFD test suite find real world faults?
Concluding Remarks

Search-Based Prioritizers

- Preliminary results indicate that search-based prioritizers can outperform random and greedy prioritizers.

- Consider additional fitness functions such as average percentage of requirements covered (APRC), test suite execution time (minimize database restarts or memory activity).


Initial Empirical Evaluation