DetReduce: Minimizing Android Test Suites for Regression Testing

Wontae Choi*, **Koushik Sen** @ UC Berkeley, George Necula*

Wenyu Wang** @ UIUC

* Currently at Google

** Work done while the author was at UC Berkeley
Motivation

• Surge in apps for smartphones and tablets
  • More mobile phone apps than desktops
• Mobile apps have complex Graphical User Interfaces (GUI)
• Testing of mobile apps focus on GUI
Observation

• Many automated GUI testing tools
  • Learning-based
  • Model-based
  • Fuzzing
  • Static analysis based
Observation

• Our experience with automated GUI testing tools
  • SwiftHand [OOPSLA’13] and Monkey
  • The good:
    • achieve good coverage and find bugs
  • The bad:
    • Runs for several hours
    • Generates a large test suite
      • Unreadable, not easy to reuse

Programmers don't like this
Can we generate a small regression test suite by minimizing a machine generated large test suite?
Why minimize?

• **Without Minimization**

  - App ver1
    - Without Minimization
      - Automated testing (few hours)
  - App ver2
    - Small change
  - App ver3
    - Automated testing (few hours)

  Repetitively pay a high cost

• **With Minimization**

  - App ver1
    - Automated testing (few hours)
  - App ver2
    - Small change
  - App ver3
    - Small change

  Minimized test suite

  One-time high cost + cheaper repetition cost
What is test case? Example

A partial model of a file browser app
Test suite and test cases

- Test suite is a set of test cases
Our goal

Original large test suite

Minimized small test suite
Our goal

Original large test suite

Minimized small test suite

DetReduce
How to Minimize? Existing work

- Optimal reduction => NP hard
- Delta debugging based minimization [Clapp et al.]
  - Creates a lot of intermediate test cases
  - Expensive to test feasibility of each each intermediate test case
  - Few hours to minimize a test case with 500 transitions
  - 10,000 transitions will take months
- **Problem**: Creates intermediate test cases by removing transitions
  - Expensive to check feasibility of each test case
- **Our goal**: develop a technique that can run within a day
DetReduce: Idea

• Key observation: 3-types of common redundancies

| Redundant test cases | Redundant loops | Redundant sub-traces |
DetReduce: Idea 1

- Key observation: 3-types of common redundancies

Redundant test cases

- Redundant loops
- Redundant sub-traces

Original set of test cases
DetReduce: Idea 1

- Key observation: 3-types of common redundancies

- Redundant test cases
- Redundant loops
- Redundant sub-traces

Original set of test cases

Redundant traces
Same color = Same test coverage
Only need to keep one of them
DetReduce: Idea 1

- Key observation: 3-types of common redundancies

1. Redundant loops
2. Redundant sub-traces
3. Redundant traces

- Same color = Same test coverage
- Only need to keep one of them

Original set of test cases

Eliminate redundant test cases

Reduced set of test cases
DetReduce: Idea 1

• Key observation: 3-types of common redundancies

- Redundant loops
- Redundant sub-traces
- Redundant traces

Same color = Same test coverage
Only need to keep one of them

Removing redundant traces = Minimal vertex cover problem (NP-hard)
Solved using a greedy selection algorithm (no feasibility check is necessary)
DetReduce: Idea 2

• Key observation: 3-types of common redundancies

Redundant test cases

Redundant loops

Redundant sub-traces

Original test-case

Menu C1

Back C2

Menu C1

Menu C2

Menu C3

foo

/foo (s1)

/foo/foo (s0)

/foo/foo (s0)

/foo (s0)

Loop = sub-trace starts and ends with the same screen

A loop is redundant if it can be removed without affecting the coverage of the trace
DetReduce: Idea 2

• Key observation: 3-types of common redundancies

Remove redundant loops using an exhaustive search (# of loops per test case is small)
Learns infeasible prefixes to reduce search space.
DetReduce: Idea 3

• Key observation: 3-types of common redundancies

Redundant test cases

Redundant loops

Redundant sub-traces

Test-case 1

Test-case 2
DetReduce: Idea 3

- Key observation: 3-types of common redundancies

Redundant test cases

Redundant loops

Redundant sub-traces

Test-case 1

S0 ▶ a C1 ▶ S1 ▶ b C2 ▶ S2 ▶ c C3 ▶ S3 ▶ d C4 ▶ S4

Test-case 2

S0 ▶ a C1 ▶ S1 ▶ b C2 ▶ S2 ▶ e C4 ▶ S4 ▶ f C5 ▶ S5

Redundant sub-test case
DetReduce: Idea 3

- Key observation: 3-types of common redundancies

**Redundant test cases**
- Redundant test cases
- Redundant loops
- Redundant sub-traces

**Test-case 1**
- S0 → S1 (a, C1)
- S1 → S2 (b, C2)
- S2 → S3 (c, C3)
- S3 → S4 (d, C4)

**Test-case 2**
- S0 → S1 (a, C1)
- S1 → S2 (b, C2)
- S2 → S3 (c, C3)
- S3 → S4 (d, C4)

Splicing useful sub-test cases

Optimal splicing = TSP problem (NP-hard)
Solved using a greedy search with a bound on # of sub-test cases per resulting trace (N=3)
**DetReduce: Summary**

- Combination of 3 heuristics
  - Successive reduction steps
  - Applies cheaper reductions first

---

<table>
<thead>
<tr>
<th>Granularity</th>
<th>Redundant test-case elimination</th>
<th>Redundant loop elimination</th>
<th>Splicing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimizes a test-suite by removing test-cases</td>
<td>Optimizes each test-case by removing loops</td>
<td>Optimizes a test-suite by splicing sub-test cases</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feasibility check</th>
<th>Not required</th>
<th>Required</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimizes a test-suite by removing test-cases</td>
<td>Optimizes each test-case by removing loops</td>
<td>Optimizes a test-suite by splicing sub-test cases</td>
<td></td>
</tr>
<tr>
<td>Not required</td>
<td>Required</td>
<td>Required</td>
<td>O(#sub-test cases ^ bound)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cost</th>
<th>Almost free (greedy + no feasibility check)</th>
<th>Linear in # of test cases</th>
<th>O(#sub-test cases ^ bound)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimizes a test-suite by removing test-cases</td>
<td>Optimizes each test-case by removing loops</td>
<td>Optimizes a test-suite by splicing sub-test cases</td>
<td>Optimizes a test-suite by splicing sub-test cases</td>
</tr>
</tbody>
</table>
Evaluation

• Implementation
  • **Front-end**: instruments an app to get runtime info
  • **Back-end**: guides testing using runtime info

  [https://github.com/wtchoi/swifthand2](https://github.com/wtchoi/swifthand2)

• Setup
  • 18 Android Apps
  • on real devices
  • Run **SwiftHand/Random** for 8 hours
    => Remove non-deterministic test cases
    => Run **DetReduce**
### Evaluation: Retained Coverage

- Retains 99.8% of branches / 98.3% of screens.
- Inherent non-determinism of apps => prevents 100% coverage

<table>
<thead>
<tr>
<th>app</th>
<th>unoptimized test suites</th>
<th>phase 1 results</th>
<th>phase 2 results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#br</td>
<td>#s.</td>
<td>#act.</td>
</tr>
<tr>
<td>acar</td>
<td>4360</td>
<td>226</td>
<td>11154</td>
</tr>
<tr>
<td>amemo</td>
<td>2846</td>
<td>139</td>
<td>1073</td>
</tr>
<tr>
<td>amoney</td>
<td>4977</td>
<td>171</td>
<td>12220</td>
</tr>
<tr>
<td>astrid</td>
<td>6075</td>
<td>254</td>
<td>10537</td>
</tr>
<tr>
<td>cnote</td>
<td>5385</td>
<td>165</td>
<td>13878</td>
</tr>
<tr>
<td>dmoney</td>
<td>2301</td>
<td>101</td>
<td>13614</td>
</tr>
<tr>
<td>emobile</td>
<td>1561</td>
<td>263</td>
<td>12201</td>
</tr>
<tr>
<td>explore</td>
<td>6753</td>
<td>108</td>
<td>7554</td>
</tr>
<tr>
<td>mileage</td>
<td>1850</td>
<td>131</td>
<td>9697</td>
</tr>
<tr>
<td>mnote</td>
<td>1015</td>
<td>153</td>
<td>14421</td>
</tr>
<tr>
<td>monefy</td>
<td>4143</td>
<td>77</td>
<td>16174</td>
</tr>
<tr>
<td>sanity</td>
<td>1091</td>
<td>195</td>
<td>14373</td>
</tr>
<tr>
<td>tipppy</td>
<td>1024</td>
<td>21</td>
<td>15729</td>
</tr>
<tr>
<td>todo</td>
<td>1828</td>
<td>78</td>
<td>10436</td>
</tr>
<tr>
<td>table</td>
<td>3445</td>
<td>167</td>
<td>14893</td>
</tr>
<tr>
<td>vlc</td>
<td>2322</td>
<td>64</td>
<td>13647</td>
</tr>
<tr>
<td>whocas</td>
<td>242</td>
<td>25</td>
<td>13175</td>
</tr>
<tr>
<td>xmp</td>
<td>2134</td>
<td>56</td>
<td>15105</td>
</tr>
<tr>
<td>median</td>
<td>2312</td>
<td>135</td>
<td>13631</td>
</tr>
</tbody>
</table>

Table 2. Test reduction result using DETREDUCE
• Running time is reduced by factor of 13.2x (on average)

<table>
<thead>
<tr>
<th>app</th>
<th>unoptimized test suites</th>
<th>phase 1 results</th>
<th>phase 2 results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#br</td>
<td>#s.</td>
<td>#act.</td>
</tr>
<tr>
<td>acar</td>
<td>4360</td>
<td>226</td>
<td>11154</td>
</tr>
<tr>
<td>amevo</td>
<td>2846</td>
<td>139</td>
<td>15150</td>
</tr>
<tr>
<td>amonkey</td>
<td>4977</td>
<td>171</td>
<td>12220</td>
</tr>
<tr>
<td>astrid</td>
<td>6075</td>
<td>254</td>
<td>10537</td>
</tr>
<tr>
<td>cnote</td>
<td>5385</td>
<td>165</td>
<td>13878</td>
</tr>
<tr>
<td>dmoney</td>
<td>2301</td>
<td>101</td>
<td>13614</td>
</tr>
<tr>
<td>emobile</td>
<td>1561</td>
<td>263</td>
<td>12201</td>
</tr>
<tr>
<td>explore</td>
<td>6753</td>
<td>108</td>
<td>7554</td>
</tr>
<tr>
<td>mileage</td>
<td>1850</td>
<td>131</td>
<td>9697</td>
</tr>
<tr>
<td>mnote</td>
<td>1015</td>
<td>153</td>
<td>14421</td>
</tr>
<tr>
<td>money</td>
<td>4143</td>
<td>77</td>
<td>16174</td>
</tr>
<tr>
<td>sanity</td>
<td>1091</td>
<td>195</td>
<td>14373</td>
</tr>
<tr>
<td>tippy</td>
<td>1024</td>
<td>21</td>
<td>15729</td>
</tr>
<tr>
<td>todo</td>
<td>1828</td>
<td>78</td>
<td>10436</td>
</tr>
<tr>
<td>ttable</td>
<td>3445</td>
<td>167</td>
<td>14893</td>
</tr>
<tr>
<td>vlc</td>
<td>2322</td>
<td>64</td>
<td>13647</td>
</tr>
<tr>
<td>whosas</td>
<td>242</td>
<td>25</td>
<td>13175</td>
</tr>
<tr>
<td>xmp</td>
<td>2134</td>
<td>56</td>
<td>15105</td>
</tr>
<tr>
<td>median</td>
<td>2312</td>
<td>135</td>
<td>13631</td>
</tr>
</tbody>
</table>

Table 2. Test reduction result using DetReduce
Evaluation: Minimization Cost

<table>
<thead>
<tr>
<th>app</th>
<th>unoptimized test suites</th>
<th>phase 1 results</th>
<th>phase 2 results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#br</td>
<td>#s.</td>
<td>#act.</td>
</tr>
<tr>
<td>acar</td>
<td>4360</td>
<td>226</td>
<td>11154</td>
</tr>
<tr>
<td>amemo</td>
<td>2846</td>
<td>139</td>
<td>9150</td>
</tr>
<tr>
<td>amoney</td>
<td>4977</td>
<td>171</td>
<td>12220</td>
</tr>
<tr>
<td>astrid</td>
<td>6075</td>
<td>254</td>
<td>10537</td>
</tr>
<tr>
<td>cnote</td>
<td>5385</td>
<td>165</td>
<td>13878</td>
</tr>
<tr>
<td>dmoney</td>
<td>2301</td>
<td>101</td>
<td>13614</td>
</tr>
<tr>
<td>emobile</td>
<td>1561</td>
<td>263</td>
<td>12201</td>
</tr>
<tr>
<td>explore</td>
<td>6753</td>
<td>108</td>
<td>7554</td>
</tr>
<tr>
<td>mileage</td>
<td>1850</td>
<td>131</td>
<td>9697</td>
</tr>
<tr>
<td>mnote</td>
<td>1015</td>
<td>153</td>
<td>14421</td>
</tr>
<tr>
<td>moneyf</td>
<td>4143</td>
<td>77</td>
<td>16174</td>
</tr>
<tr>
<td>sanity</td>
<td>1091</td>
<td>195</td>
<td>14373</td>
</tr>
<tr>
<td>tippity</td>
<td>1024</td>
<td>21</td>
<td>15729</td>
</tr>
<tr>
<td>todo</td>
<td>1828</td>
<td>78</td>
<td>10436</td>
</tr>
<tr>
<td>table</td>
<td>3445</td>
<td>167</td>
<td>14893</td>
</tr>
<tr>
<td>vlc</td>
<td>2322</td>
<td>64</td>
<td>13647</td>
</tr>
<tr>
<td>whohas</td>
<td>242</td>
<td>25</td>
<td>13175</td>
</tr>
<tr>
<td>xmp</td>
<td>2134</td>
<td>56</td>
<td>15105</td>
</tr>
<tr>
<td>median</td>
<td>2312</td>
<td>135</td>
<td>13631</td>
</tr>
</tbody>
</table>

Table 2. Test reduction result using DetReduce

• Minimization time < 6x of the input test suite’s running time
Minimizing an automatically generated test suite is challenging
- NP hard problem => need heuristic
- Feasibility check => each reduction attempt is expensive

Automatically generated GUI test suites can be minimized
- Problem specific heuristic is key to scalability
- DetReduce focuses on 3-common types of redundancies in GUI test suites

https://github.com/wtchoi/swifthand2