ASK THE MUTANTS

MUTATING FAULTY PROGRAMS FOR FAULT LOCALISATION

Shin Yoo, University College London
Seokhyun Mun, KAIST
Yunho Kim, KAIST
Moonzoo Kim, KAIST
• MUSE: Mutation-based Fault Localisation Engine
• Locality Information Loss: a new evaluation metric
• Ongoing work (post ICST 2014)
OUR CLAIM

On the 10KLOC SIR benchmark programs

Ranking of faulty stmt among all executed stmts (%)

- **Tarantula** [ICSE 2002]: 25%
- **Ochiai** [PRDC 06]: 9%
- **Wong** [JSS 10]: 1%
- **Op2** [TOSEM 11]: 1%

**MUSE** (MUtation baSEd FL): 2002 - 2014
MOTIVATION

- We have hit the ceiling of Spectrum-based Fault Localisation
- Not accurate enough, effectiveness varies significantly depending on test suites, inherently limited by block-level granularity
- Can we use mutation testing in a pre-emptive manner?
WHAT HAPPENS WHEN YOU MUTATE ALREADY FAULTY PROGRAMS?
CASE 1: MUTATING CORRECT STATEMENTS

Equivalent

P  F

New Fault

P-  F+

Mask

P+  F-
CASE 2: MUTATING FAULTY STATEMENT

(Partial) Fix

P+ F-

(New) Fault

P? F?

Equivalent

P F

Mask

P+ F-
HYPOTHESES

• An arbitrary mutation operator applied to a correct statement is likely to introduce a new fault

• An arbitrary mutation operator applied to a faulty statement is either likely to keep the program still faulty or, even better, (partially) fix the program

• The majority of statements in a faulty statement is correct; we detect the faulty one by observing the anomaly from our hypotheses
MUSE

\[ \mu(s) = \frac{1}{|\text{mut}(s)|} \sum_{m \in \text{mut}(s)} \left( \frac{|f_P(s) \cap p_m|}{|f_P|} \right) \left( \frac{|p_P(s) \cap f_m|}{|p_P|} \right) - \alpha \cdot \frac{|p_P(s) \cap f_m|}{|p_P|} \]

Proportion of test cases
that mutant \( m \) turns
from fail to pass

Proportion of test cases
that mutant \( m \) turns
from pass to fail

Average over all
mutation applied
to statement \( s \)

\[ \alpha = \frac{f2p}{|\text{mut}(P)| \cdot |f_P|} \cdot \frac{|\text{mut}(P)| \cdot |p_P|}{p2f} \]
## EMPIRICAL EVALUATION

<table>
<thead>
<tr>
<th>Subject Program</th>
<th>% of executed stmts examined</th>
<th>Rank of a faulty stmt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Jaccard</td>
<td>Ochiai</td>
</tr>
<tr>
<td>flex v1</td>
<td>49.48</td>
<td>45.04</td>
</tr>
<tr>
<td>flex v7</td>
<td>3.60</td>
<td>3.60</td>
</tr>
<tr>
<td>flex v11</td>
<td>19.76</td>
<td>19.54</td>
</tr>
<tr>
<td>grep v3</td>
<td>1.06</td>
<td>1.01</td>
</tr>
<tr>
<td>grep v11</td>
<td>3.44</td>
<td>3.44</td>
</tr>
<tr>
<td>gzip v2</td>
<td>2.14</td>
<td>2.14</td>
</tr>
<tr>
<td>gzip v5</td>
<td>1.83</td>
<td>1.83</td>
</tr>
<tr>
<td>gzip v13</td>
<td>1.03</td>
<td>1.03</td>
</tr>
<tr>
<td>sed v1</td>
<td>0.54</td>
<td>0.54</td>
</tr>
<tr>
<td>sed v3</td>
<td>2.56</td>
<td>2.56</td>
</tr>
<tr>
<td>sed v5</td>
<td>37.84</td>
<td>37.84</td>
</tr>
<tr>
<td>space v19</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>space v21</td>
<td>0.45</td>
<td>0.45</td>
</tr>
<tr>
<td>space v28</td>
<td>11.57</td>
<td>10.66</td>
</tr>
<tr>
<td>Average</td>
<td>9.67</td>
<td>9.27</td>
</tr>
</tbody>
</table>

### Notes:
- The values in the table represent the performance metrics of the four methods: Jaccard, Ochiai, Op2, and MUSE.
- **% of executed stmts examined** indicates the proportion of executed statements required to be examined before localizing the fault.
- **Rank of a faulty stmt** indicates the rank of the faulty statement among the total number of statements.
- **Correct Faulty (B)/(A)**: Precision.
- **Correct Faulty (C)/(D)**: Recall.

We believe MUSE is precise enough that its results can be interpreted as a good candidate for a new SBFL technique. The LIL metric values of MUSE are the smallest among the four methods, indicating that MUSE can localize a fault with more precision than the other methods. In addition, MUSE produces the most precise results for 11 out of the 14 studied faulty programs, which is a significant improvement over the state-of-the-art SBFL techniques. This provides quantitative evidence that MUSE is a promising technique for fault localization.
MOTIVATION

- Traditional evaluation metric for fault localisation is ranking based

- Measures something else than accuracy (and, even then, humans do not operate in linear ranking)

- Irrelevant for Automated Patching: Qi et al. show that rank-wise suboptimal formula helps GenProg better (ISSTA 2013)
LIL (LOCALITY INFORMATION LOSS)

- Any suspiciousness score distribution can be interpreted as a probability distribution.
- Describe the actual location of the fault as THE probability distribution.
- Calculate Kullbeck-Leibler divergence (distance between two probability distributions).
LIL

\[ \mathcal{L}(s_i) = \begin{cases} 
1 & (s_i = s_f) \\
\epsilon & (0 < \epsilon \ll 1, s_i \in S, s_i \neq s_f)
\end{cases} \]

\[ P_\tau(s_i) = \frac{\tau(s_i)}{\sum_{i=1}^{n} \tau(s_i)}, \ (1 \leq i \leq n) \]

\[ D_{KL}(P_\mathcal{L} \parallel P_\tau) = \sum_i \ln \left( \frac{P_\mathcal{L}(s_i)}{P_\tau(s_i)} \right) P_\mathcal{L}(s_i) \]
..WORTH A THOUSAND WORDS

Op2 (LIL=7.34)

Jaccard (LIL=4.92)

Ochiai (LIL=5.96)

MUSE (LIL=0.40)
ONGOING WORK

• With actual progress
  • Mutant Sampling
  • Hybridisation
• More adventurous
  • Multiple Faults
  • Pre-emptive localisation
• Learning mutation
• Genetic Programming for MUSE
Isn’t mutating every statement really expensive?

After random sampling, still more accurate than Op2
MUSE + SBFL(JACCARD)

- Evaluated 10 randomly chosen real PHP bugs from ICSE 2012 GenProg Paper
- In some cases, brings MUSE closer to Op2
- In other cases, pure version still wins

<table>
<thead>
<tr>
<th>Faulty version #</th>
<th>MUSE</th>
<th>MUSE + jacc (Current Hybrid Muse)</th>
<th>Op2</th>
<th>Target line #</th>
<th># failed</th>
<th># passed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>160</td>
<td>1957</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>13</td>
<td>53</td>
<td>112</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>42</td>
<td>14</td>
<td>14</td>
<td>143</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>177</td>
<td>8</td>
<td>8</td>
<td>362</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>371</td>
<td>140</td>
<td>140</td>
<td>553</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>4</td>
<td>170</td>
<td>191</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>18</td>
<td>64</td>
<td>119</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>9</td>
<td>164</td>
<td>222</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>9</td>
<td>49</td>
<td>45</td>
<td>45</td>
<td>49</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>24</td>
<td>362</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>Average</td>
<td>66</td>
<td>25.3</td>
<td>84.2</td>
<td>407.00</td>
<td>1.20</td>
<td>100.00</td>
</tr>
</tbody>
</table>
MULTIPLE FAULTS CONJECTURE

- For independent faults that result in disjoint failures, MUSE is not affected at all.
- For faults that interact with each other, test suite design/composition will play a key role.
PRE-EMPTIVE LOCALISATION

• Mutation analysis is still expensive, especially as a step for debugging which is often urgent

• Can we do the mutation analysis in advance, even with the previous version?
  • For each mutant, record the failure pattern across test cases
  • When a real failure is observed, track back to the point of mutation
GP FOR MUSE

• GP worked for SBFL; does it work for MUSE?
• More variables, which means a larger search space
What happens when you mutate already faulty programs?

Our claim:
- **Tarantula** ([ICSE 2002]
- **Ochiai** ([PRDC 06]
- **Wong** ([IS 10]
- **Op2** ([TOSEM 11]
- **MUSE** (MUtation baSED FL)

On the 10KLOC SIR benchmark programs

Isn’t mutating every statement really expensive?

Jaccard (LIL=4.92)

Executed Statements

Faulty Statement

Suspiciousness

After random sampling, still more accurate than Op2