On the Equivalence of Certain Fault Localization Techniques

Vidroha Debroy and W. Eric Wong
Department of Computer Science
The University of Texas at Dallas
vxd024000@utdallas.edu
http://www.utdallas.edu/~vxd024000

Fault Localization, i.e., finding the locations of faults in a program is one of the most expensive activities in program debugging.

Thus, a lot of different fault localization techniques have been proposed in the recent past.

– Each technique tries to locate faults in a different way.

However, the norm for comparing techniques to one another has been empirical in nature, i.e., using time consuming and tedious case studies.

We must ask ourselves…is a data-driven comparison always required in order to compare fault localization techniques?

– Particularly if we are trying to show they are equal?
Outline

• Fault Localization
  – Preliminaries
  – An Illustrative Example
• Fault Localization Equivalence
• An Additional Benefit of Equivalence – Increased Efficiency
• Conclusion/Future Work
• Questions/Comments

Fault Localization
Fault Localization

- Fault localization techniques assist programmers in finding the locations of faults.
- Many of them produce a ranking of program components such as statements such that they are ranked in decreasing order of suspiciousness (likelihood of being faulty).
- Programmers can examine the ranking starting from the top (i.e., most suspicious) and continue down the ranking until they encounter a fault.
- For example, the Tarantula technique assigns suspiciousness as:
  - suspiciousness \( s \) = \( \frac{X}{X+Y} \)
  - where \( X \) = fraction of failed test cases that execute statement \( s \)
  - and \( Y \) = fraction of successful test cases that execute statement \( s \)

Let us look at an example using Tarantula:

<table>
<thead>
<tr>
<th>Stmt. #</th>
<th>Desired Program (P)</th>
<th>Faulty Program (P')</th>
<th>Coverage</th>
<th>Suspiciousness based on Tarantula</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>read(a);</td>
<td>read(a);</td>
<td>• • •</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>read(b);</td>
<td>read(b);</td>
<td>• • •</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>sum= a +b;</td>
<td>sum= a +b;</td>
<td>• • •</td>
<td>0.5</td>
</tr>
<tr>
<td>4</td>
<td>if(sum &lt; 100)</td>
<td>if(sum &lt; 100)</td>
<td>• • •</td>
<td>0.5</td>
</tr>
<tr>
<td>5</td>
<td>sum = sum / 2;</td>
<td>sum = sum / 2;</td>
<td>• • 0.66</td>
<td>0.5</td>
</tr>
<tr>
<td>6</td>
<td>print(sum);</td>
<td>print(sum);</td>
<td>• • •</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Test \( t_1 \):
- Input: a = 0
- b = 0
- output: 0, 0 (Pass)

Test \( t_2 \):
- Input: a = 50
- b = 50
- output: 100, 100 (Pass)

Test \( t_3 \):
- Input: a = 10
- b = 10
- output: 40, 10 (Fail)

- Identified statement 5 as the most suspicious
- Programmer will examine it first among all the statements
A host of different techniques have been proposed recently
- Statistical techniques, heuristic-based techniques, similarity-coefficient-based techniques, slicing-based techniques, etc.

Which technique should be used? …the most effective
- If technique $\alpha$ is better than technique $\beta$, then it should lead programmers to the location of fault(s) faster than $\beta$.
- Percentage of code that needs be examined to find a fault (the more effective the technique, the smaller this value should be)

The norm has always been to perform case studies and use data to compare the effectiveness of fault localization techniques.
- This can be very tedious and time consuming

Can such empirical comparisons be avoided in certain cases?
Fault Localization Equivalence (1)

- Note that fault localization techniques produce rankings of statements in descending order of their suspiciousness.

- This means that the suspiciousness of a statement is irrelevant from an absolute sense.
  - It only matters how the suspiciousness of two (or more) statements compare with respect to each other (i.e., relative to one another).
  - It does not matter if the suspiciousness of a faulty statement is high; it only matters if it is higher than that of a non-faulty statement.

- Subtracting the same constant from (or adding it to) the suspiciousness of every statement will have no effect on the final ranking. The same applies for multiplication/division operations.

Fault Localization Equivalence (2)

- For example, recall the suspiciousness computation of Tarantula: suspiciousness \( s \) = \( X/(X+Y) \)

- An identical ranking will be produced by using the formulae:
  - suspiciousness \( s \) = \( X/(X+Y) \) ± 1, or
  - suspiciousness \( s \) = \( X/(X+Y) \) ± 2, or
  - suspiciousness \( s \) = \( (X/(X+Y)) \times 2 \), or
  - suspiciousness \( s \) = \( (X/(X+Y)) / 10 \), etc.

- Any operation that is order-preserving can be safely performed on the suspiciousness function, without changing the ranking.

- If the ranking does not change…then the effectiveness will not change either. *We can exploit this!*
Fault Localization Equivalence

- Consider a program $P$ with $M$ statements. Let $\text{rank}(r,s)$ be a function that returns the position of statement $s$ in ranking $r$.

- Two rankings $r_\alpha$ and $r_\beta$ (produced by using two techniques $L_\alpha$ and $L_\beta$ on the same input data) are equal if:
  - $\forall s \in M \text{ rank}(r_\alpha,s) = \text{rank}(r_\beta,s)$.
  - Simply put, two rankings are equal if for every statement, the position is the same in both rankings.

- If two fault localization techniques $L_\alpha$ and $L_\beta$ always produce rankings that are equal, then the techniques are said to be equivalent, i.e., $L_\alpha \equiv L_\beta$ and therefore, they shall always be equally as effective (at fault localization).

Usefulness of Equivalence

- This fault localization equivalence relation can help reduce the need for extensive case studies.

- For example, in “A practical evaluation of spectrum-based fault localization” - Journal of Systems and Software (Volume 82 Issue 11, November, 2009), the authors propose the use of the Ochiai coefficient to compute suspiciousness.
  - The coefficient is compared to several other coefficients empirically.
  - Among others, Ochiai is compared to the Jaccard, Anderberg and Sorensen-Dice coefficients.
  - The paper states that it is also compared to the Simple Matching and Rogers and Tanimoto coefficients.

- This could have been avoided, as per the equivalence relation.
Usefulness of Equivalence

(2)

- Jaccard:
  \[ \text{suspiciousness}(s) = \frac{N_{CF}}{N_{CF} + N_{UF} + N_{CP}} \]

- Sorensen-Dice
  \[ \text{suspiciousness}(s) = \frac{2N_{CF}}{2N_{CF} + N_{UF} + N_{CP}} \]

- Via a set of order-preserving operations, both of the above can be reduced to:
  \[ \text{suspiciousness}(s) = \frac{N_{CF}}{N_{CF} + N_{UF}} \]

\[ \text{Jaccard} \equiv \text{Sorensen-Dice} \]

Usefulness of Equivalence

(3)

- As it turns out the similarity coefficient Anderberg also evaluates to the same form (as Jaccard and Sorensen-Dice).

- The SimpleMatching and Rogers and Tanimoto coefficients, can also be shown to be equivalent to one another.

Such redundant comparisons could have been avoided by making use of the fault localization equivalence relation.

- Also, if a data-based comparison reveals two techniques to be equally effective, it does not mean they will always be equally effective (on other subject programs). However, if two techniques can be shown to be equivalent, then they will always be equally as effective regardless of subject program.
Improved Efficiency

(1)

• As shown, if Jaccard were the chosen fault localization technique, using the suspiciousness function:

\[
suspiciousness(s) = \frac{N_{CF}}{N_{CF} + N_{UF} + N_{CP}}
\]

would give the same results as using:

\[
suspiciousness(s) = \frac{N_{CF}}{N_{UF} + N_{CP}}
\]

• We should go with the simplest computation as it is expected to be the fastest.

• Using the equivalence relation can thus, help reduce techniques to simplified forms, thereby increasing efficiency.
Improved Efficiency

- We performed a case study on 7 small-sized programs (Siemens suite) and one large-sized program (make).

- Correct versions of programs, faulty versions, test cases, all downloaded from the Software-artifact Infrastructure Repository:
  - http://sir.unl.edu/portal/index.html

- Observed the relative time saved in computing suspiciousness for all the statements in a faulty program, by using the simplified form of Jaccard ($J^*$) as opposed to the original ($J$).
  - The quantity ($J^* - J$) represents the computational time that is saved.
  - $((J^* - J)/J) \times 100\%$ represents the relative time saved, i.e., efficiency gained.

<table>
<thead>
<tr>
<th>Siemens suite</th>
<th>Programs</th>
<th>Lines of Code</th>
<th>Number of faulty versions</th>
<th>Number of test cases</th>
<th>Average Percentage Time Saved</th>
</tr>
</thead>
<tbody>
<tr>
<td>print_tokens</td>
<td>565</td>
<td>5</td>
<td>4130</td>
<td></td>
<td>35.37%</td>
</tr>
<tr>
<td>print_tokens2</td>
<td>510</td>
<td>10</td>
<td>4115</td>
<td></td>
<td>39.21%</td>
</tr>
<tr>
<td>schedule</td>
<td>412</td>
<td>9</td>
<td>2650</td>
<td></td>
<td>44.62%</td>
</tr>
<tr>
<td>schedule2</td>
<td>307</td>
<td>9</td>
<td>2710</td>
<td></td>
<td>49.74%</td>
</tr>
<tr>
<td>replace</td>
<td>563</td>
<td>32</td>
<td>5542</td>
<td></td>
<td>41.65%</td>
</tr>
<tr>
<td>tcas</td>
<td>173</td>
<td>41</td>
<td>1608</td>
<td></td>
<td>52.46%</td>
</tr>
<tr>
<td>tot_info</td>
<td>406</td>
<td>23</td>
<td>1052</td>
<td></td>
<td>47.68%</td>
</tr>
<tr>
<td>make</td>
<td>20014</td>
<td>31</td>
<td>793</td>
<td></td>
<td>43.22%</td>
</tr>
</tbody>
</table>
**Conclusion**

- We have proposed the notion of fault localization equivalence by virtue of which if two or more fault localization techniques can be theoretically compared.
  - Using a set of order-preserving operations, we prove the mutual equivalence of several fault localization techniques, and show that empirical comparisons in prior studies were unnecessary.

- In the future we plan to
  - investigate the mutual equivalency of a host of many different fault localization techniques.
  - find simpler forms of pre-existing techniques, thereby making them more efficient than before.

**Questions/Comments**

Thank You
Supporting Material

Jaccard and Sorensen-Dice (1)

- Let us start with Jaccard
  \[
  \text{suspiciousness}(s) = \frac{N_{CF}}{N_{CF} + N_{UF} + N_{CP}}
  \]

- Divide throughout by \( N_{CF} \)
  \[
  \text{suspiciousness}(s) = \frac{1}{1 + \frac{N_{UF} + N_{CP}}{N_{CF}}}
  \]

- Adding 1 to the denominator of each suspiciousness computation has no effect on the ranking
  \[
  \text{suspiciousness}(s) = \frac{1}{1 + \frac{N_{UF} + N_{CP}}{N_{CF}}} = \frac{N_{CF}}{N_{CF} + N_{UF} + N_{CP}}
  \]
Jaccard and Sorensen-Dice (2)

- Next we look at Sorensen-Dice
  \[ \text{suspiciousness}(s) = \frac{2N_{CF}}{2N_{CF} + N_{UF} + N_{CP}} \]

- Divide throughout by \(2N_{CF}\)
  \[ \text{suspiciousness}(s) = \frac{1}{1 + \frac{N_{UF} + N_{CP}}{2N_{CF}}} \]

- As with Jaccard this reduces to
  \[ \text{suspiciousness}(s) = \frac{2N_{CF}}{N_{UF} + N_{CP}} \]

- But scaling the suspiciousness by 2 does not change the ranking!!