Isolating Failure-Inducing Combinations in Combinatorial Testing using Test Augmentation and Classification

Kiran Shakya    Tao Xie
North Carolina State University

Nuo Li    Yu Lei    Raghu Kacker    Richard Kuhn
ABB Robotics    University of Texas at Arlington    Information Technology Lab NIST

CT 2012 workshop
• Software normally has faults.
• Given a System Under Test (SUT) with N input parameters, a failure is usually caused by interaction among k parameters where k << N.

Problem:
- Generating CT for even a small k (such as 5 or 6) is computationally expensive for SUT with large N.
- CT results may be insufficient for diagnosis due to failures caused by interactions among 5 or more parameters (aka faulty combinations)
Background

Previous Approach

CT Suite Results → Classification → Failure Inducing Combination

Our Approach

CT Suite Results → Test Suite Augmentation → Feature Selection → Classification → Failure Inducing Combination
Problem

1. Often it's hard to judge the size of faulty interactions.
2. Generating CT of higher strength is expensive.
3. Fault diagnosis on lower strength CT results may not provide good results.
1. Problem
2. Example
3. Approach
4. Proof of Concept
5. Conclusion
Example

- Consider TCAS v16
  - # of Parameters: 12
  - Total Input Space: $3 \times 2^3 \times 3 \times 2 \times 4 \times 10^2 \times 3 \times 2 \times 3 = 1036800$
- Assume we don’t know in advance the nature of failures.
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cur_Vertical_Sep</td>
<td>299, 300, 601</td>
</tr>
<tr>
<td>High_Confidence</td>
<td>0, 1</td>
</tr>
<tr>
<td>Two_of_Three_Reports_Vaid</td>
<td>0, 1</td>
</tr>
<tr>
<td>Own_Tracked_Alt</td>
<td>1, 2</td>
</tr>
<tr>
<td>Own_Tracked_Alt_Rate</td>
<td></td>
</tr>
<tr>
<td>Other_Tracked_Alt</td>
<td>1, 2</td>
</tr>
<tr>
<td>Alt_Layer_Value</td>
<td>0, 1, 2, 3</td>
</tr>
<tr>
<td>Up_Separation</td>
<td>0, 399, 400, 499, 500, ...</td>
</tr>
<tr>
<td>Down_Separation</td>
<td>0, 399, 400, 499, 500, ...</td>
</tr>
<tr>
<td>Other_RAC</td>
<td>0, 1, 2</td>
</tr>
<tr>
<td>Other_Capability</td>
<td>1, 2</td>
</tr>
<tr>
<td>Climb_Inherit</td>
<td>0, 1</td>
</tr>
</tbody>
</table>
### Characteristic of Failure (TCAS v16)

<table>
<thead>
<tr>
<th>CT Strength</th>
<th>Failing/Total Number of Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-way</td>
<td>0/156</td>
</tr>
<tr>
<td>3-way</td>
<td>1/461</td>
</tr>
<tr>
<td>4-way</td>
<td>6/1450</td>
</tr>
<tr>
<td>5-way</td>
<td>14/4309</td>
</tr>
</tbody>
</table>
Result of Classification Tree:

• ( EMPTY )

Reason:

• Data Set is Highly Unbalanced.
• Not enough Failing Tests.
Approach

Faulty Combinations

Labeled Test cases

Test Augmentation

Feature Selection

Classification Model

Ranking

Test Execution

Combinatorial Tests
Test Augmentation

- Use OFOT\(^1\) (one factor one time) method to generate additional tests from failing tests.

Ex: Given a Failing Test:

\[601,1,1,1,600,2,3,740,400,0,2,1\]

OFOT generates

\[300,1,1,1,600,2,3,740,400,0,2,1\]

\[299,1,1,1,600,2,3,740,400,0,2,1\]

\[601,0,1,1,600,2,3,740,400,0,2,1\]

…..

Maximum number of tests generated by OFOT is

\[ m \times \left( \sum_{i=1}^{k} a_i - k \right) \]

where \( m \) is total no of failing tests, \( k \) is the number of parameters, and \( a_i \) is distinct input values for each parameter.

This is far less than the number of tests required to build higher strength array.

For Example: 6-way Tests: 6,785 vs OFOT: 612
Run the classification tree algorithm

High_Confidence = 0: 0 (2248.0/12.0)
High_Confidence = 1
|   Alt_Layer_Value = 0
|   |   Own_Tracked_Alt_Rate = 600
|   |   |   Cur_Vertical_Sep = 299: 0 (149.0/12.0)
|   |   |   Cur_Vertical_Sep = 300
|   |   |   |   Two_of_Three_Reports_Valid = 0: 0 (28.0/2.0)
|   |   |   |   Two_of_Three_Reports_Valid = 1
|   |   |   |   Other_RAC = 0
|   |   |   |   |   Other_Tracked_Alt = 1
|   |   |   |   |   |   Other_Capability = 1: 1 (4.0)
|   |   |   |   |   |   |   Other_Capability = 2: 0 (3.0)
|   |   |   |   |   |   |   Other_Tracked_Alt = 2: 1 (6.0)
...(and many more nodes)
Test Augmentation Result

<table>
<thead>
<tr>
<th>Version</th>
<th>Test Aug</th>
<th>Effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>302/357</td>
<td>73%</td>
</tr>
<tr>
<td>26</td>
<td>407/407</td>
<td>80%</td>
</tr>
</tbody>
</table>
Can we do more?

- Developers typically use classification tree to manually analyze the nature of faults
- Clearly smaller the size of tree, easier will be the debugging process

For Example:

- Classification tree generated for TCAS has 56 nodes
- Can we reduce the size of classification tree?
Objective of Feature Selection

- Identifying and removing irrelevant and redundant information as much as possible.

What kind of feature Selection:

- Correlation based feature selection (H.A. Mark, Ph.D. dissertation, Univ of Waikato, 1999.)
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cur_Vertical_Sep</td>
<td>299, 300, 601</td>
</tr>
<tr>
<td>High_Confidence</td>
<td>0, 1</td>
</tr>
<tr>
<td>Two_of_Three_Reports_Valid</td>
<td>0, 1</td>
</tr>
<tr>
<td>Own_Tracked_Alt</td>
<td>1, 2</td>
</tr>
<tr>
<td>Own_Tracked_Alt_Rate</td>
<td></td>
</tr>
<tr>
<td>Other_Tracked_Alt</td>
<td>1, 2</td>
</tr>
<tr>
<td>Alt_Layer_Value</td>
<td>0, 1, 2, 3</td>
</tr>
<tr>
<td>Up_Separation</td>
<td>0, 399, 400, 499, 500, ...</td>
</tr>
<tr>
<td>Down_Separation</td>
<td>0, 399, 400, 499, 500, ...</td>
</tr>
<tr>
<td>Other_RAC</td>
<td>0, 1, 2</td>
</tr>
<tr>
<td>Other_Capability</td>
<td>1, 2</td>
</tr>
<tr>
<td>Climb_Inherit</td>
<td>0, 1</td>
</tr>
<tr>
<td>Version</td>
<td>Test Aug</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>16</td>
<td>302/357</td>
</tr>
<tr>
<td>26</td>
<td>407/407</td>
</tr>
</tbody>
</table>
For each leaf node that indicates a failure, a corresponding likely faulty combination is computed by:

- Taking the conjunction of the parameter values found in the path from the root node to the leaf node
- Calculate its score

Output: Fail

Output: Pass

Combination: $A = 1$ and $B = 1$

$10/12 = 0.83$
Proof of Concept

- Hypothesis: The faulty should show up higher in the rank.

- Final Outcome:
  - TCAS v26, our approach did find the faulty combination.
  - TCAS v16, out of two combinations, our approach found one of them.
int alt_sep_test() {
    ....
    enabled=High_Confidence &&
    /*(Own_Tracked_Alt_Rate<=OLEV) && BUG */
    (Cur_Vertical_Sep>MAXALTDIFF);
    ....
}

HighConfidence=1 && OwnTrackedAltRate>OLEV(=600) &&
CurVerticalSep>MAXALTDIFF(=600)
Diagnosis of failure when the number of failures are low.

Our approach:

- Tries to balance the test generation and classification for fault diagnosis

Proof of concept on two versions of TCAS
Thank you

Questions?