IPOG: A General Strategy for T-Way Software Testing

Y. Lei (UT Arlington), R. Kacker (NIST), D. R. Kuhn (NIST), V. Okun (NIST), J. Lawrence (GMU)

ECBS, Tucson, Arizona

Software Engineering

- Software has become pervasive in modern society
  - Directly contributes to quality of life
  - Malfunctions cost billions of dollars every year, and could have severe consequences in a safety-critical environment

- Build better software in better ways, especially for large-scale development
  - Requirements, design, coding, testing, maintenance, configuration, documentation, deployment, and etc.
Software Testing

- A dynamic approach to detecting software faults
  - Alternatively, static analysis can be performed, which is however often intractable

- Involves sampling the input space, running the test object, and observing the runtime behavior
  - Intuitive, easy-to-use, scalable, and can be very effective for fault detection

- Perhaps the most widely used approach to ensuring software quality in practice
The Challenge

- Testing is labor intensive and can be very costly
  - often consumes more than 50% of the development cost
- Exhaustive testing is often impractical, and is not always necessary
- How to make a good trade-off between test effort and test coverage?
Outline

- **Introduction**
  - T-way testing
  - State-of-the-art

- **The IPOG Strategy**
  - Algorithm IPOG-Test
  - Experimental results

- Related Work on T-Way Testing

- Conclusion and Future Work
T-Way Testing

- Given any \( t \) input parameters of a test object, every combination of values of these parameters be covered by at least one test

- Motivation: Many faults can be exposed by interactions involving a few parameters
  - Each combination of parameter values represents one possible interaction between these parameters

- Advantages
  - Light specification, requires no access to source code, automated test input generation, excellent trade-off between test effort and test coverage
Example

Three parameters, each with values 0 and 1

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

pairwise

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

exhaustive
State-of-the-Art

- **Greedy construction**
  - Involves explicit enumeration of all possible combinations
  - tries to cover as many combinations as possible at each step

- **Algebraic Construction**
  - Test sets are constructed using pre-defined rules

- **Most approaches focus on 2-way (or pairwise) testing**
Beyond pairwise

- Many software faults are caused by interactions involving more than two parameters
  - A recent NIST study by R. Kuhn indicates that failures can be triggered by interactions up to 6 parameters

- Increased coverage leads to a higher level of confidence
  - Safety-critical applications have very strict requirements on test coverage
Outline

- Introduction
  - T-way testing
  - State-of-the-art

- The IPOG Strategy
  - Algorithm IPOG-Test
  - Experimental results

- Related Work on T-Way Testing

- Conclusion and Future Work
The Framework

- Construct a *t*-way test set for the first *t* parameter
- Extend the test set to cover each of the remaining parameters one by one
  - **Horizontal growth** - extends each existing test by adding one value for the new parameter
  - **Vertical growth** - adds new tests, if needed, to make the test set complete
Algorithm IPOG-Test

Algorithm IPOG-Test (int t, ParameterSet ps)
{
1. initialize test set ts to be an empty set
2. denote the parameters in ps, in an arbitrary order, as P₁, P₂, ..., and Pₙ
3. add into test set ts a test for each combination of values of the first t parameters
4. for (int i = t + 1; i ≤ n; i ++){
5.   let π be the set of t-way combinations of values involving parameter Pᵢ
       and t - 1 parameters among the first i - 1 parameters
6.   // horizontal extension for parameter Pᵢ
7.   for (each test τ = (v₁, v₂, ..., vᵢ₋₁) in test set ts) {
8.     choose a value vᵢ of Pᵢ and replace τ with τ’ = (v₁, v₂, ..., vᵢ₋₁, vᵢ) so that τ’ covers the
        most number of combinations of values in π
9.     remove from π the combinations of values covered by τ’
10. }
11. } // vertical extension for parameter Pᵢ
12. for (each combination σ in set π){
13.   if (there exists a test that already covers σ) {
14.     remove σ from π
15. } else {
16.     change an existing test, if possible, or otherwise add a new test
         to cover σ and remove it from π
17. }
18. }
19.}
20. return ts;
}
Example

- Four parameters: P1, P2, P3, and P4
- P1, P2, and P3 have 2 values
- P4 has 3 values

(a) Horizontal growth

(b) Vertical growth

(c) 3-way test set
Experimental Results (1)

Question 1: How does the size of a test set generated by IPOG-Test, as well as the time taken, grow in terms of \( t \), \# of parameters, and \# of values?

<table>
<thead>
<tr>
<th>t-way</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>48</td>
<td>308</td>
<td>1843</td>
<td>10119</td>
<td>50920</td>
</tr>
<tr>
<td>time</td>
<td>0.11</td>
<td>0.56</td>
<td>6.38</td>
<td>63.8</td>
<td>791.35</td>
</tr>
</tbody>
</table>

Results for 10 5-value parameters for 2- and 6-way testing
# Experimental Results (2)

## # of params

<table>
<thead>
<tr>
<th># of params</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>784</td>
<td>1064</td>
<td>1290</td>
<td>1491</td>
<td>1677</td>
<td>1843</td>
<td>1990</td>
<td>2132</td>
<td>2254</td>
<td>2378</td>
<td>2497</td>
</tr>
<tr>
<td>Time</td>
<td>0.19</td>
<td>0.45</td>
<td>0.92</td>
<td>1.88</td>
<td>3.58</td>
<td>6.38</td>
<td>10.83</td>
<td>17.52</td>
<td>27.3</td>
<td>41.71</td>
<td>61.26</td>
</tr>
</tbody>
</table>

Results for 5 to 15 5-value parameters for 4-way testing

## # of values

<table>
<thead>
<tr>
<th># of values</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>46</td>
<td>229</td>
<td>649</td>
<td>1843</td>
<td>3808</td>
<td>7061</td>
<td>11993</td>
<td>19098</td>
<td>28985</td>
</tr>
<tr>
<td>Time</td>
<td>0.16</td>
<td>0.547</td>
<td>1.8</td>
<td>6.33</td>
<td>16.44</td>
<td>38.61</td>
<td>83.96</td>
<td>168.37</td>
<td>329.36</td>
</tr>
</tbody>
</table>

Results for 10 parameters with 2 to 10 values for 4-way testing
Experimental Results (3)

Question 2: How does FireEye compare to other tools, both in terms of # of tests and time to produce them?

<table>
<thead>
<tr>
<th>t-way</th>
<th>FireEye</th>
<th>ITCH</th>
<th>Jenny</th>
<th>TConfig</th>
<th>TVG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size</td>
<td>Time</td>
<td>Size</td>
<td>Time</td>
<td>Size</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>0.8</td>
<td>120</td>
<td>0.73</td>
<td>108</td>
</tr>
<tr>
<td>3</td>
<td>400</td>
<td>0.36</td>
<td>2388</td>
<td>1020</td>
<td>413</td>
</tr>
<tr>
<td>4</td>
<td>1361</td>
<td>3.05</td>
<td>1484</td>
<td>5400</td>
<td>1536</td>
</tr>
<tr>
<td>5</td>
<td>4219</td>
<td>18.41</td>
<td>NA</td>
<td>&gt;1 day</td>
<td>4580</td>
</tr>
<tr>
<td>6</td>
<td>10919</td>
<td>65.03</td>
<td>NA</td>
<td>&gt;1 day</td>
<td>11625</td>
</tr>
</tbody>
</table>

Results of different tools for the TCAS application

**TCAS:** Seven 2-value parameters, two 3-value parameters, one 4-value parameter, two 10-value parameters
Outline

- Introduction
  - T-way testing
  - State-of-the-art

- The IPOG Strategy
  - Algorithm IPOG-Test
  - Experimental Results

- Related Work on T-Way Testing

- Conclusion and Future Work
AETG (1)

- Starts with an empty set and adds one (complete) test at a time
- Each test is locally optimized to cover the most number of missing pairs:
- Has a higher order of complexity, both in terms of time and space, than IPOG
AETG (2)

- Adds the 1st test
- Adds the 2nd test
- Adds the last test
Orthogonal Arrays (1)

- Given any $t$ columns, every combination of the possible values is covered in the same number of times
  - Originally used for statistical design, which often requires a balanced coverage
  - Often computed using some pre-defined mathematical functions

- Each row can be considered as a test, and each column as a parameter

- Can be constructed extremely fast, and are optimal by definition, but do not always exist
Orthogonal Arrays (2)

<table>
<thead>
<tr>
<th>(b0, b1)</th>
<th>A = b1</th>
<th>B = b0 + b1</th>
<th>C = b0 + 2 * b1</th>
<th>D = b0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 0)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(0, 1)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>(0, 2)</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(1, 0)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(1, 1)</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(1, 2)</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>(2, 0)</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>(2, 1)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>(2, 2)</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
Outline

- Introduction
  - T-way testing
  - State-of-the-art

- The IPOG Strategy
  - Algorithm IPOG-Test
  - Experimental Results

- Related Work on T-Way Testing

- Conclusion and Future Work
Conclusion

- T-way testing can substantially reduce the number of tests, while remaining effective for fault detection
- IPOG produces a t-way test set incrementally, covering one parameter at a step
- Comparing to existing tools, IPOG can produce smaller tests faster.
Future Work

- Explicit enumeration can be very costly
  - How to reduce the number of combinations that have to be enumerated?

- Support for parameter relations and constraints
  - No need to cover combinations of independent parameters
    - Invalid combinations must be excluded

- Integration of $t$-way testing with other tools to increase the degree of automation