Automated Combinatorial Testing for Software

Rick Kuhn and Raghu Kacker

National Institute of Standards and Technology
Gaithersburg, MD
What is NIST?

• A US Government agency

• The nation’s measurement and testing laboratory – 3,000 scientists, engineers, and support staff including 3 Nobel laureates

Analysis of engineering failures, including buildings, materials ...

Research in physics, chemistry, materials, manufacturing, computer science
Software Failure Analysis

• NIST studied software failures in a variety of fields including 15 years of FDA medical device recall data

• What *causes* software failures?

• What testing and analysis would have prevented failures?

• Would all-values or all-pairs testing find all errors, and if not, then how many interactions would we need to test to find all errors?

• Surprisingly, no one had looked at this question before
Interaction testing

<table>
<thead>
<tr>
<th>Interest Rate</th>
<th>Amount</th>
<th>Months</th>
<th>Down Pmt</th>
<th>Pmt Frequency</th>
</tr>
</thead>
</table>

All values: every value of every parameter

All pairs: every value of each pair of parameters

t-way interactions: every value of every t-way combination of parameters

etc....
How to find all failures?

• Interactions:
  
  • E.g., failure occurs if
    pressure < 10 (1-way interaction)
    pressure < 10 & volume > 300
    (2-way interaction)
  
  • Most complex failure required
    4-way interaction
How about other applications?

- Browser
How about other applications?

- Server
How about other applications?

- NASA distributed database
How about other applications?

- TCAS module (seeded errors)
What interactions would we need to test to find **ALL** faults?

- **Max interactions for fault triggering** for these applications was **6**
  - Wallace, Kuhn 2001 - medical devices
    - 98% of flaws were pairwise interactions, no fault required > 4-way interactions to trigger
  - Kuhn, Reilly 2002 - web server, browser;
    no fault required > 6-way interactions to trigger
  - Kuhn, Wallace, Gallo 2004 - large NASA distributed database;
    no fault required > 4 interactions to trigger

- **Much more empirical work needed**
- Reasonable evidence that maximum interaction strength for fault triggering is **relatively small**

- **How can we apply what we have learned?**
Automated Combinatorial Testing

- Merge automated test generation with combinatorial methods

- Goals – reduce testing cost, improve cost-benefit ratio for software assurance

- New algorithms and faster processors make large-scale combinatorial testing practical

- Accomplishments – huge increase in performance, scalability + proof-of-concept demonstration

- Also non-testing application – modelling and simulation
Problem: the usual ...

- Too much to test
- Testing may exceed 50% of development cost
- Even with formal methods, we still need to test
- Need maximum amount of information per test

Example: 20 variables, 10 values each
- $10^{20}$ combinations
- Which ones to test?
Solution: Combinatorial Testing

- Pairwise testing commonly applied to software
- Suppose no failure requires more than a pair of settings to trigger in previous example
- Then test all pairs - 180 test cases sufficient to detect any failure
- Pairwise testing can find 50% to 90% of flaws

What if finding 50% to 90% of flaws is not good enough?
A simple example

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

0 = effect off
1 = effect on

13 tests for all 3-way combinations

$2^{10} = 1,024$ tests for all combinations
A covering array:
10 parameters, 2 values each, 3-way combinations

Any 3 columns contain all possible combinations

13 tests for all 3-way combinations

$2^{10} = 1,024$ tests for all combinations

So what happens for realistic examples?
A real-world example

Input data to web application:

Plan: flt, flt+hotel, flt+hotel+car

From: CONUS, HI, AK, Europe, Asia...

To: CONUS, HI, AK, Europe, Asia...

Compare: yes, no

Date-type: exact, 1to3, flex

Depart: today, tomorrow, 1month, 1yr...

Return: today, tomorrow, 1month, 1yr...

Adults: 1, 2, 3, 4, 5, 6

Minors: 0, 1, 2, 3, 4, 5

Seniors: 0, 1, 2, 3, 4, 5

No silver bullet because:

Many values per variable

Requires more tests and practical limits

Need to abstract values

But we can still increase information per test
Two ways of using combinatorial testing

Use combinations here

or here

Configuration

Test data inputs

System under test
Combinatorial testing requires a lot of tests, but now we can do this

• Generating covering arrays is a hard problem, one reason why anything beyond pairwise testing is rarely used

• Number of tests: suppose we want all 4-way combinations of 30 parameters, 5 values each: 3,800 tests

• May need $10^3$ to $10^7$ tests for realistic systems

• With new algorithms we can produce large covering arrays quickly
New algorithms

- Smaller test sets faster, with a more advanced user interface
- First parallelized covering array algorithm
- More information per test

### Table 6. 6 way, 5^k configuration results comparison

**insufficient memory**
Result Checking

• Creating test data is the easy part!

• How do we check that the code worked correctly on the test input?
  
  • **Configuration coverage**, using existing test set
    - Easy, if test set exists
  
  • **Crash testing** server or other code to ensure it does not crash for any test input
    - Easy but limited correctness check
  
  • Use basic **consistency checks on system output**
    - Better but more costly
  
  • **White box testing** - incorporate assertions in code to check critical states at different points in the code, or print out important values during execution
  
  • **Full scale model-checking** using mathematical model of system and model checker to generate expected results for each input
    - expensive but tractable
Using model checking to produce tests

The system can never get in this state!  Yes it can, and here's how ... 

- Model-checker test production: if assertion is not true, then a counterexample is generated.
- This can be converted to a test case.

Black & Ammann, 1999
Proof-of-concept experiments

• FAA Traffic Collision Avoidance System module
  • Mathematical model of system and model checker for results
  • 41 versions seeded w/ errors, used in previous testing research
  • 12 variables: 7 boolean, two 3-value, one 4-value, two 10-value
  • Tests generated w/ Lei algorithm extended for >2 parameters
  • >17,000 complete test cases, covering 2-way to 6-way combinations generated and executed in a few minutes
  • All flaws found with 5-way coverage

• Grid computer simulator
  • Preliminary results
  • Crashes in >6% of tests w/ valid values
  • “Interesting” combinations discovered
Where does this stuff make sense?

• More than (roughly) 8 parameters and less than 300-400
• Processing involves interaction between parameters (numeric or logical)

Where does it not make sense?

• Small number of parameters (where exhaustive testing is possible)
• No interaction between parameters
Summary

- Empirical research suggests that all software failures caused by interaction of few parameters.
- Combinatorial testing can exercise all t-way combinations of parameter values in a very tiny fraction of the time needed for exhaustive testing.
- New algorithms and faster processors make large-scale combinatorial testing possible.
- Project could produce better quality testing at lower cost for US industry and government.
- Beta release of tools in December, to be open source.
- New public catalog of covering arrays.
Future directions

• No silver bullet - but does it improve cost-benefit ratio?
  What kinds of software does it work best on?
  What kinds of errors does it miss?
• Large real-world examples will help answer these questions
• Other applications:
  • Modelling and simulation
  • Testing the simulation
  • Finding interesting combinations:
    performance problems, denial of service attacks
• Maybe biotech applications. Others?

Please contact us if you are interested!

Rick Kuhn                       Raghu Kacker
kuhn@nist.gov        raghu.kacker@nist.gov

http://csrc.nist.gov/acts