CS1632, Lecture 16: Pairwise and Combinatorial Testing

Wonsun Ahn

Let’s Test A Word Processor

- Specifically, its ten possible font effects
  - Italic
  - Bold
  - Underline
  - Strikethrough
  - Superscript
  - Shadow
  - Embossed
  - 3-D
  - Outline
  - Inverse

These can be combined

- Plain text
- Superscript
- **Bold**
- *~~Italic and strikethrough~~*
- **Bold and underlined**
- ***~~Bold italic strikethrough shadowed superscript~~***

How many tests would you need to test all the possible font combinations?

2^{10}

1,024 tests!

That’s quite a few tests…

b	i	u	s	h	e	d	o	n
T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	F
T	T	T	T	T	T	T	F	T
T	T	T	T	T	T	T	F	F
T	T	T	T	T	T	F	T	T
T	T	T	T	T	T	F	T	F
T	T	T	T	T	T	F	F	T
T	T	T	T	T	T	F	F	F
T	T	T	T	T	F	T	T	T
T	T	T	T	T	F	T	T	F
T	T	T	T	T	F	T	F	T
T	T	T	T	T	F	T	F	F
T	T	T	T	T	F	F	T	T
T	T	T	T	T	F	F	T	F
T	T	T	T	T	F	F	F	T
T	T	T	T	T	F	F	F	F
T	T	T	T	F	T	T	T	T
T	T	T	T	F	T	T	T	F
T	T	T	T	F	T	T	F	T
T	T	T	T	F	T	T	F	F
T	T	T	T	F	T	F	T	T
T	T	T	T	F	T	F	T	F
T	T	T	T	F	T	F	F	T
T	T	T	T	F	T	F	F	F
T	T	T	T	F	F	T	T	T
T	T	T	T	F	F	T	T	F
T	T	T	T	F	F	T	F	T
T	T	T	T	F	F	T	F	F
T	T	T	T	F	F	F	T	T
T	T	T	T	F	F	F	T	F
T	T	T	T	F	F	F	F	T
T	T	T	T	F	F	F	F	F
T	T	T	F	T	T	T	T	T
T	T	T	F	T	T	T	T	F
T	T	T	F	T	T	T	F	T
T	T	T	F	T	T	T	F	F
T	T	T	F	T	T	F	T	T
T	T	T	F	T	T	F	T	F
T	T	T	F	T	T	F	F	T
T	T	T	F	T	T	F	F	F
T	T	T	F	T	F	T	T	T
T	T	T	F	T	F	T	T	F
T	T	T	F	T	F	T	F	T
T	T	T	F	T	F	T	F	F
T	T	T	F	T	F	F	T	T
T	T	T	F	T	F	F	T	F
T	T	T	F	T	F	F	F	T

But it's necessary! What if…

… a problem only occurs with 3-D shadowed bold italic superscript text?

That's going to be hard to find.

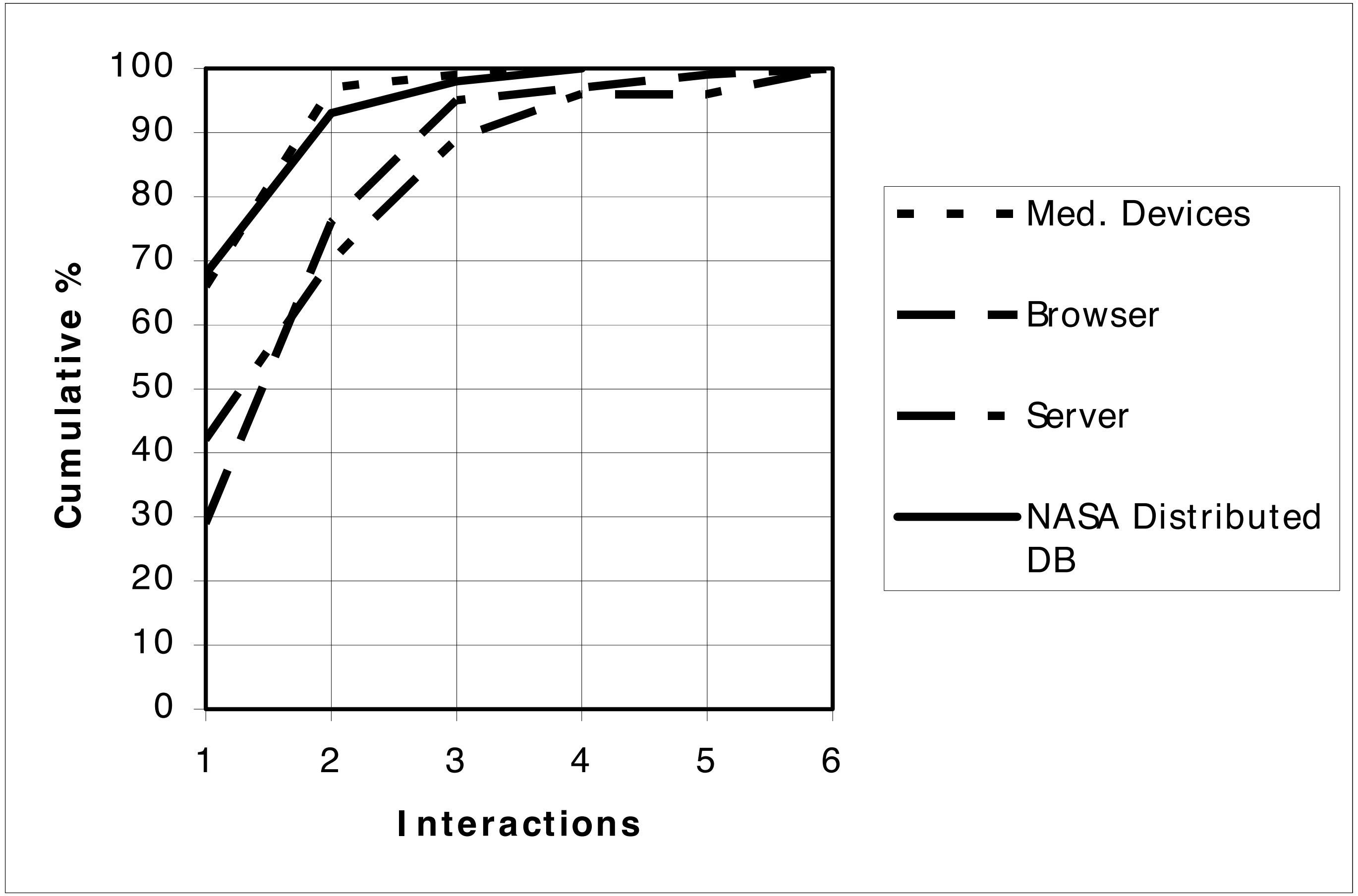
Turns Out Other People Have Thought About This!

- The National Institute of Standards and Technology did a survey
- See: "Practical Combinatorial Testing", http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-142.pdf
- Study of dozens of applications in 6 domains
  - Medical devices, Web Browser, Web Server, Database, Network Security

Turns Out That's Unlikely!

- On average, percentage of defects covered by …
  - A single variable: 17 − 68%
  - Two interacting variables: 53 − 97%
  - Three interacting variables: 74 − 99%
  - Four interacting variables: 89 − 100%
  - Five interacting variables: 96 − 100%
  - Six interacting variables: 100%
- Majority of defects are found just by testing all possible pairs
- At max, just SIX variables were involved in a defect

Error detection rates for interactions 1 to 6



Similar Distribution Found In Many Domains

- Web browser
- Avionics software
- Telecommunications software
- Flight Traffic Control
- Network security software

Pairwise Testing

- This is called “pairwise”, or “all-pairs” testing.
- We are testing all possible pairs of interactions, e.g.:
  - Not-Bold / Not-Italic
  - Bold / Not-Italic
  - Not-Bold / Italic
  - Bold / Italic

Remember our 10-font-effect testing plan?

- It was 1,024 (2 ^ 10) tests to test exhaustively.
- How many tests would it require to test only pairs of interactions?
  - That is, all possible combinations of:
    - bold/italic,
    - subscript/bold
    - underline/strikethrough
    - Every possible pairing of two variables
- Choose 2 from 10 = $\binom{10}{2} = \frac{10 * 9}{2} = 45$ and 4 combination per pair
  - 45 * 4 = 180 tests?
  - No! A single test can test multiple pairs of interactions at the same time

Answer: 8

No.	BOLD	ITALIC	STRIKETHROUGH	UNDERLINE	THREAD	SHADOW	SUPERSCRIPT	SUBSCRIPT	EMBOSSED	ENGRAVED
1	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE	FALSE
2	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE
3	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
4	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE
5	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
6	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	TRUE
7	-	-	-	-	-	-	-	FALSE	-	FALSE
8	-	-	-	-	-	-	-	TRUE	-	TRUE

- Reduced tests by more than an order of magnitude! (180 → 8)
- This is called a *covering array* (will tell you how to make this soon)
- Is this always good enough test coverage?

Of course not

- But we can “dial up” the number of possible interactions
  - To check for any *t* number of interactions
- For example, check every three-way combination ($t = 3$):
  - Bold / Italic / Underline
- Or four-way ($t = 4$)
  - Bold / Italic / Underline / Superscript
- Up to whatever number of interactions possible
  - At this point, would be the same as exhaustive testing

Combinatorial Testing

- This generalized testing for any t is known as "combinatorial testing"
- Pairwise testing is an instance of combinatorial testing where $t = 2$

Combinatorial Testing Example

- According to NIST, max number of interactions in a defect is 6
  - So let's test all six-way combinations of our font effects

- Recall that:
  - # tests required for full pairwise testing was 8
  - # tests required for exhaustive testing was 1,024
  - How many to test all six-way interactions?

Difficult to answer off the top of your head

- Determining the exact number necessary is an NP-Hard problem.
- But there are some good algorithms out there that approximate it
- "IPOG: A General Strategy for T-Way Software Testing" (ECBS '07) https://www.nist.gov/publications/ipog-general-strategy-t-way-software-testing

… and the answer is…

- The best answer IPOG software could come up with is 165.
- Approximately an order of magnitude less than exhaustive testing!
- But finds the same number of defects, according to NIST.

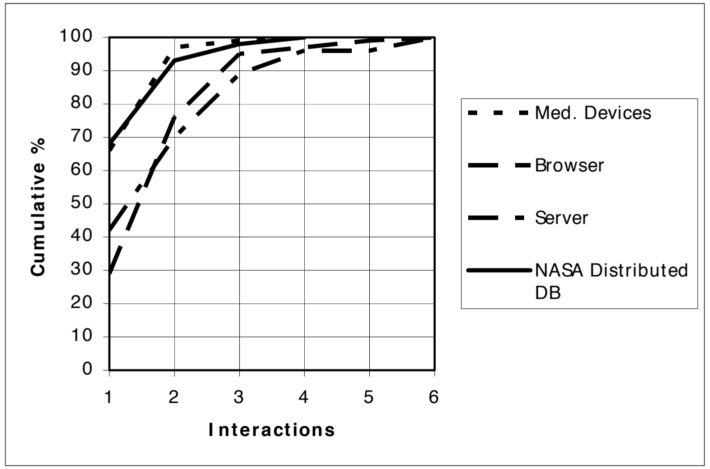
Interesting!

- Pairwise testing (8 tests): catches 90% of defects
- Six-way testing (165 tests): catches ~99.9999999% of defects
- Exhaustive testing (1024 tests): catch ~100% of defects

- IF THEY ARE DONE RIGHT!
- The "right" combination of tests for each situation is given in: https://math.nist.gov/coveringarrays/ipof/ipof-results.html
  - Even these are not optimal (remember the problem is NP-Hard)
  - But they are pretty close

Law of Diminishing Returns (on *t*)

- We already saw increasing *t* doesn't get us much beyond *t = 2*



- How about testing cost (10 variables, 5 values per variable)?

t-way	2	3	4	5	6
No. of Tests	48	308	1843	10119	50920

☛Cost increases exponentially as we increase *t*!

Law of Diminishing Returns (on *t*)

- Testing Cost = $O(d^{t} * \log n)$
  - ***t***: number of interactions
  - *d*: domain size (number of values a parameter can take)
  - *n*: number of parameters
- Takeaways
  - Limit *t*: no benefit beyond *t=6*, and cost is prohibitive
  - *log n*: can cover a large number of parameters with ease

Covering Arrays

- Covering array: set of test cases covering all *t*-way combinations
- At below is a covering array where t = 3

Tests

A	B	C	D	E	F	G	H	I	J
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	1
1	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0
1	1	0	1	0	0	1	0	1	0
0	0	0	1	1	1	0	0	1	1
0	0	1	1	0	0	1	0	0	1
0	1	0	1	1	0	0	1	0	0
1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	1	1	1	0	1

Steps To Make Your Own Covering Array

- Make a truth table with all variables
  - Each line in truth table indicates a test
  - Running all these tests would be an exhaustive test
- Make a list of all *t*-way interactions for desired *t*
  - *Example: Bold, Italic, Underline. t = 2*
    - Bold / Italic
    - Bold /Underline
    - Italic/Underline

Generating Covering Arrays

- Goal: Complete "mini truth table" for each t-way interaction
  - E.g. For 2-way: F F | F T | T F | T T
- Starting from the first interaction do the following:
  - Add test case that fulfills each entry in the mini truth table
    E.g. Bold / Underline = T F can be fulfilled by:

No.	BOLD	ITALIC	STRIKETHROUGH	UNDERLINE	THREAD	SHADOW	SUPERSCRIPT	SUBSCRIPT	EMBOSSED	ENGRAVED
392	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE	FALSE

  - If an already added test case fulfills the entry, nothing to do!
- Continue until mini truth tables for all interactions are completed

Sounds easy enough. Why is it NP-Hard?

- Note there are *many* candidates to choose from.

  E.g. Bold / Underline = T F can be fulfilled by:

No.	BOLD	ITALIC	STRIKETHROUGH	UNDERLINE	THREAD	SHADOW	SUPERSCRIPT	SUBSCRIPT	EMBOSSED	ENGRAVED
392	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE	FALSE

  But also:

No.	BOLD	ITALIC	STRIKETHROUGH	UNDERLINE	THREAD	SHADOW	SUPERSCRIPT	SUBSCRIPT	EMBOSSED	ENGRAVED
123	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE

- But there are only a handful of optimal choices.
  - Here is where the NP-Hardness creeps in.
  - We’ll just choose randomly. It affects quality but not correctness of solution.

Covering Array Example

Bold	Italic	Underline		Mini-Truth	
F	F	F		F	F
F	F	T		F	T
F	T	F		T	F
F	T	T		T	T
T	F	F			
T	F	T			
T	T	F			
T	T	T			

Covering Array Example

Test	Bold	Italic	Underline		
1	F	F	F		Bold / Italic
2	F	F	T		Bold / Underline
3	F	T	F		Italic / Underline
4	F	T	T		
5	T	F	F		
6	T	F	T		
7	T	T	F		
8	T	T	T		

Covering Array Example – Bold / Italic

Test	Bold	Italic	Underline		
1	F	F	F		Bold / Italic
2	F	F	T		Bold / Underline
3	F	T	F		Italic / Underline
4	F	T	T		
5	T	F	F		
6	T	F	T		
7	T	T	F		
8	T	T	T		

Covering Array Example – Bold / Underline

Test	Bold	Italic	Underline		
1	F	F	F		Bold / Italic
2	F	F	T		Bold / Underline
3	F	T	F		Italic / Underline
4	F	T	T		
5	T	F	F		
6	T	F	T		
7	T	T	F		
8	T	T	T		

Covering Array Example – Italic / Underline

Test	Bold	Italic	Underline			
1	F	F	F		Bold / Italic	
2	F	F	T		Bold / Underline	
3	F	T	F		Italic / Underline	
4	F	T	T			
5	T	F	F			
6	T	F	T			
7	T	T	F			
8	T	T	T			

Run a Subset of Tests

Test	Bold	Italic	Underline		
1	F	F	F		Bold / Italic
2	F	F	T		Bold / Underline
3	F	T	F		Italic / Underline
4	F	T	T		
5	T	F	F		Necessary Tests
6	T	F	T		Unnecessary Tests
7	T	T	F		
8	T	T	T		

Can Minimize Further Using Better Algorithms

Test	Bold	Italic	Underline		
1	F	F	F		Bold / Italic
2	F	F	T		Bold / Underline
3	F	T	F		Italic / Underline
4	F	T	T		
5	T	F	F		Necessary Tests
6	T	F	T		Unnecessary Tests
7	T	T	F		
8	T	T	T		

OK, this works for small numbers of variables, but what about big ones?

- Imagine a 34-variable system
  - Exhaustive testing: 17 billion tests
  - All 3-way interactions: 33 tests
  - All 4-way interactions: 85 tests
- Actually gets BETTER the higher the number of variables (n)
  - Cost of exhaustive testing: $O(2^n)$ ➔ Exponential!
  - Cost of covering array: $O(2^t * \log n)$ ➔ Logarithmic!
- Not just a little better – many orders of magnitude better

Remember at the beginning of the term when I talked about the impossibility of testing every combination of inputs?

This is a possible amelioration.

Won’t It Take Long To Manually Make Covering Arrays For Large Number of Variables?

YES

- Are you kidding? I already told you it is an NP-Hard problem.
- You can use a program to do it for you.
- Example: NIST ACTS

https://csrc.nist.gov/Projects/automated-combinatorial-testing-for-software/downloadable-tools

- Or you can use a pre-generated covering array for your situation

https://math.nist.gov/coveringarrays/ipof/ipof-results.html

Now Please Read Textbook Chapter 17