SEENG3320/6320:
Software Verification and Validation

School of Electrical Engineering and Computing

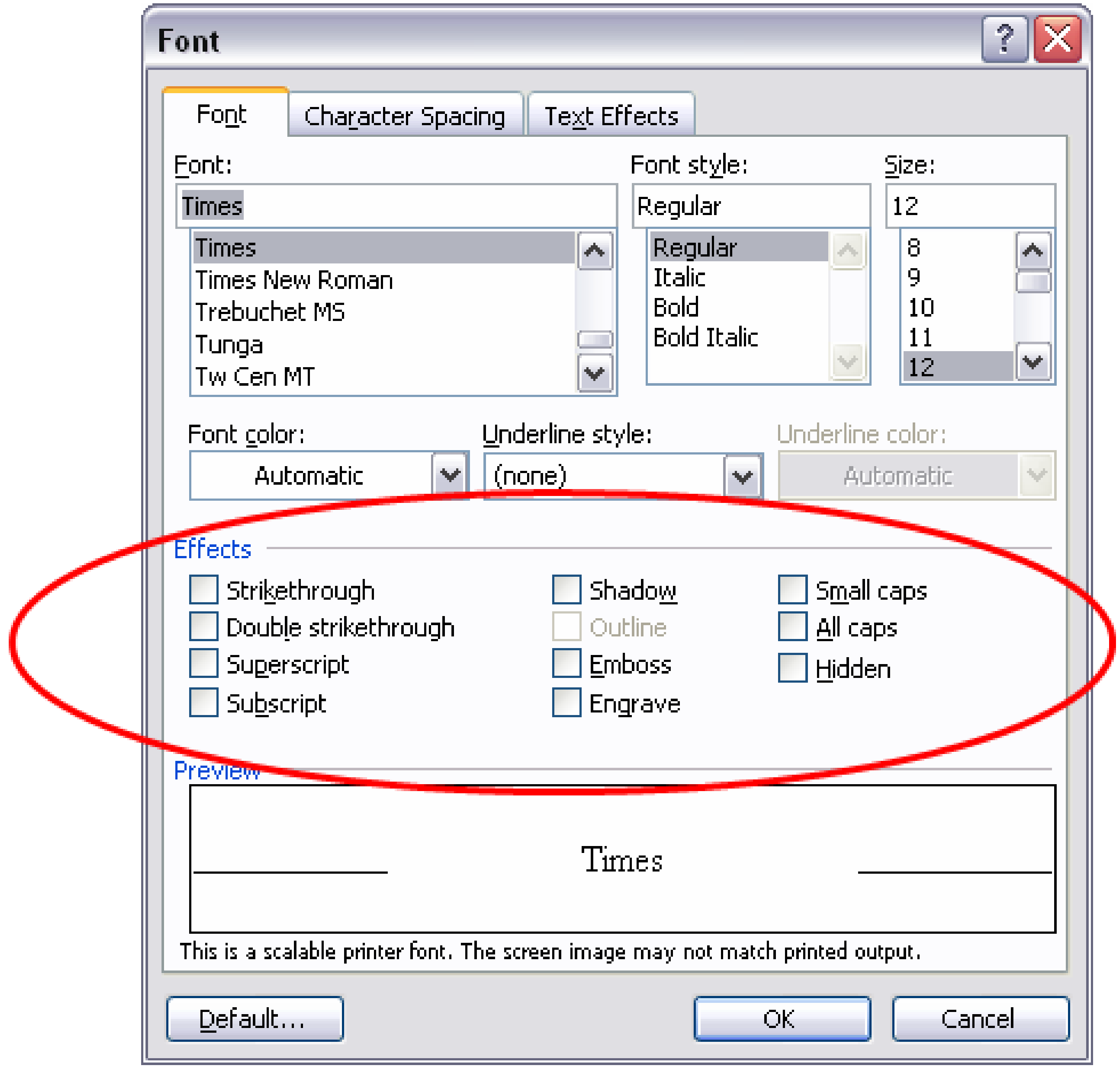
Semester 1, 2020

Combinatorial Testing

What is combinatorial testing? A simple example

- 10 effects, each can be on or off
- All combinations is $2^{10} = 1{,}024$ tests

Combinatorial Explosion: assuming that each effect has 5 values…



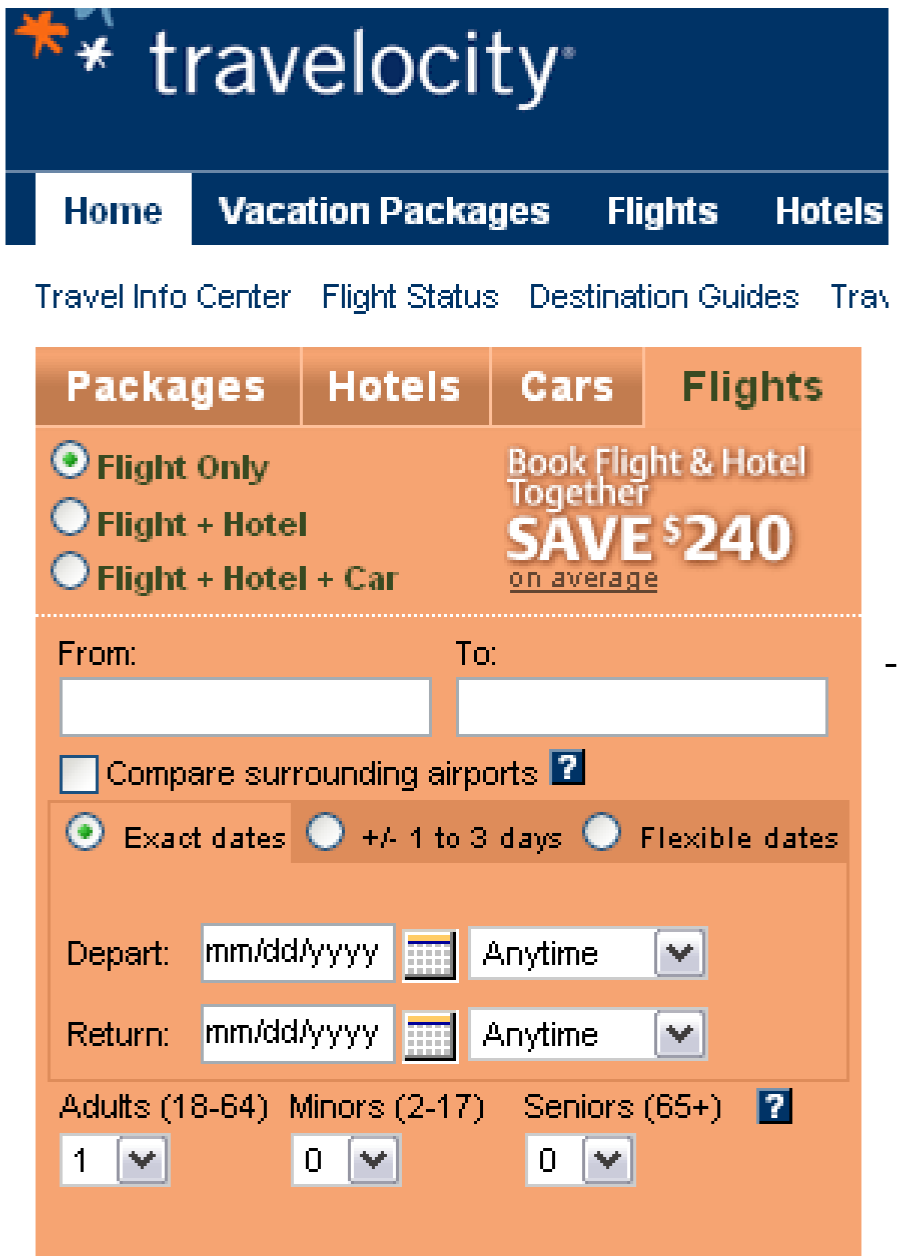
A larger example

Suppose we have a system with on-off switches. Software must produce the right response for any combination of switch settings:

34 switches = 2^{34} = 1.7×10^{10} possible inputs = **1.7×10^{10}** tests

NIST
National Institute of Standards and Technology

A real-world example



Plan: flt, flt+hotel, flt+hotel+car
From: CONUS, HI, Europe, Asia …
To: CONUS, HI, Europe, Asia …
Compare: yes, no
Date-type: exact, 1to3, flex
Depart: today, tomorrow, 1yr, Sun, Mon …
Return: today, tomorrow, 1yr, Sun, Mon …
Adults: 1, 2, 3, 4, 5, 6
Minors: 0, 1, 2, 3, 4, 5
Seniors: 0, 1, 2, 3, 4, 5

https://www.travelocity.com/

Combinatorial Testing

- Instead of testing all possible combinations, a subset of combinations is generated to satisfy some well-defined combination strategies.
- A key observation is that not every variable contributes to every fault, and it is often the case that a fault is caused by interactions among a few variables.
- Combinatorial testing can dramatically reduce the number of combinations to be covered but remains very effective in terms of fault detection.

t-way Interaction

- Many faults are caused by the interactions between variables.
- A t-way interaction fault is a fault that is triggered by a certain combination of t input values.
- A simple fault is a t-way fault where t = 1; a pairwise fault is a t-way fault where t = 2.
- In practice, the majority of faults in a software applications consist of simple and pairwise faults.

Each Choice Coverage

- Target at 1-way interaction: each variable value must be covered in at least one test case
- Also called: **All-values Testing.**
- Consider the previous example, a test set that satisfies each choice coverage is the following:

{(A, 1, x), (B, 2, y), (A, 3, x)}

Three variables:

P1 = (A, B), P2 = (1, 2, 3), and P3 = (x, y),

Pairwise Coverage

- Target at 2-way Interaction: Given any two variables, every combination of values of these two variables are covered in at least one test case.
- Also called 2-way coverage, pairwise testing.
- A pairwise test set of the previous example is the following:

P1	P2	P3
A	1	x
A	2	x
A	3	x
A	-	y
B	1	y
B	2	y
B	3	y
B	-	x

P1 = (A, B), P2 = (1, 2, 3), and P3 = (x, y),

3-way Coverage

- Target at 3-way interaction: Given any three variables, every combination of values of these three variables are covered in at least one test case.

A 3-way coverage test set

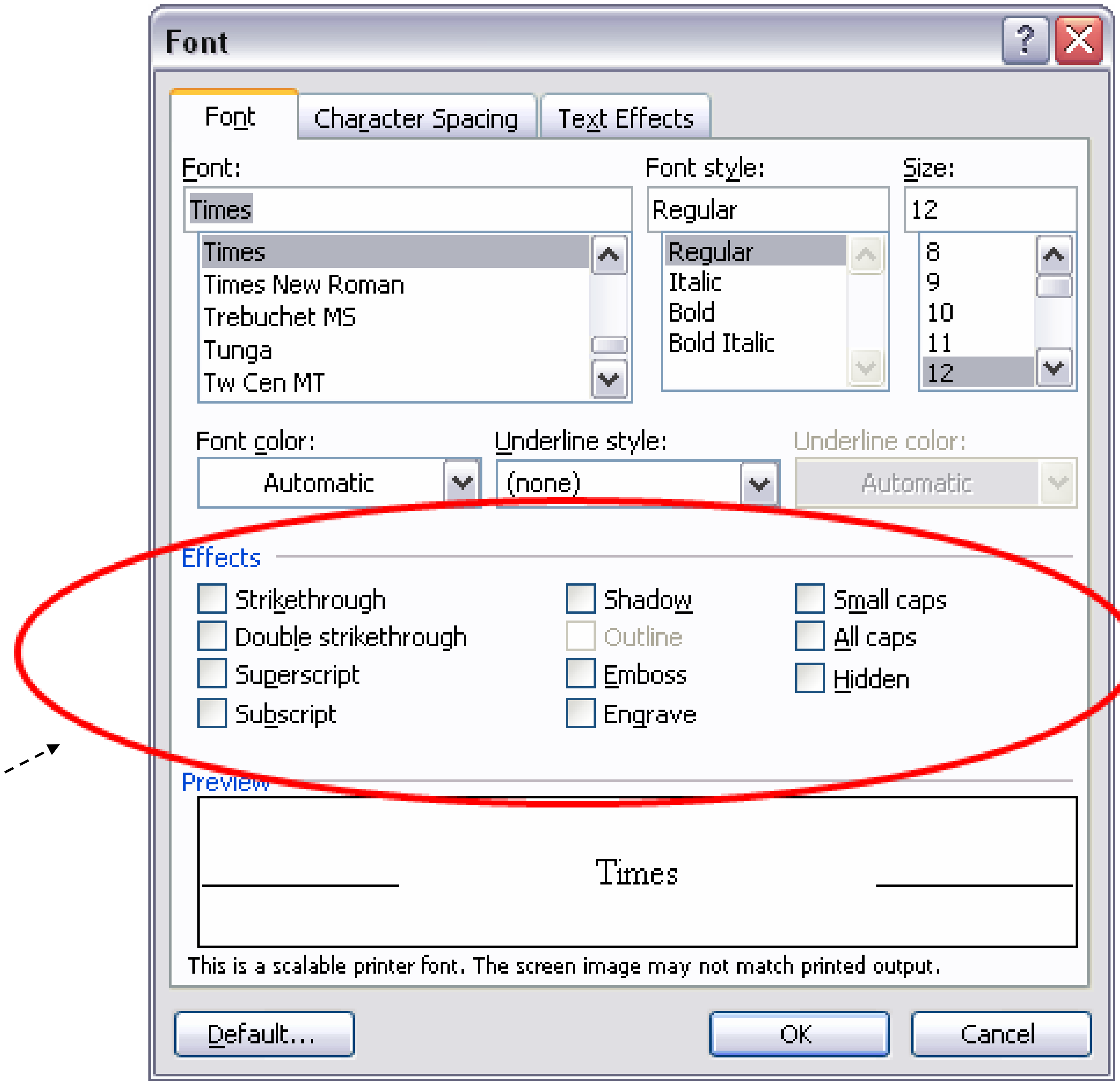
(A, 1, x),
(A, 1, y),
(A, 2, x),
(A, 2, y),
(A, 3, x),
(A, 3, y),
(B, 1, x),
(B, 1, y),
(B, 2, x),
(B, 2, y),
(B, 3, x),
(B, 3, y)

P1 = (A, B), P2 = (1, 2, 3), and P3 = (x, y),

t-Way Coverage

- Given any *t* variables, every combination of values of these *t* variables must be covered in at least one test case.
  - For example, a 3-way coverage requires every triple be covered in at least one test case.
  - Note that each choice, and pairwise coverage can be considered to be a special case of t-way coverage.
- Need to design a **minimum number of** test inputs that achieve t-way coverage.

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



0 = effect off
1 = effect on

13 tests for **all 3-way combinations**

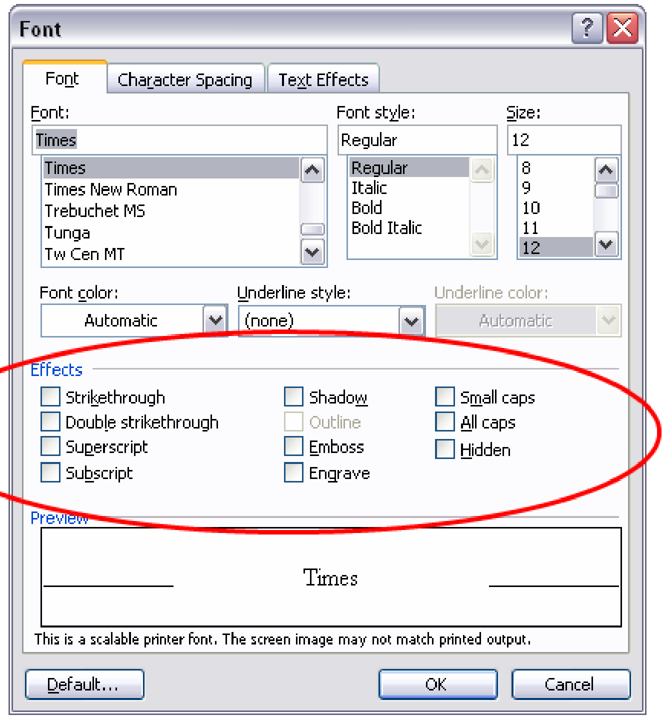
2^{10} = **1,024** tests for **all combinations**

All triples in only 13 tests, covering $\binom{10}{3} 2^3 = 960$ combinations

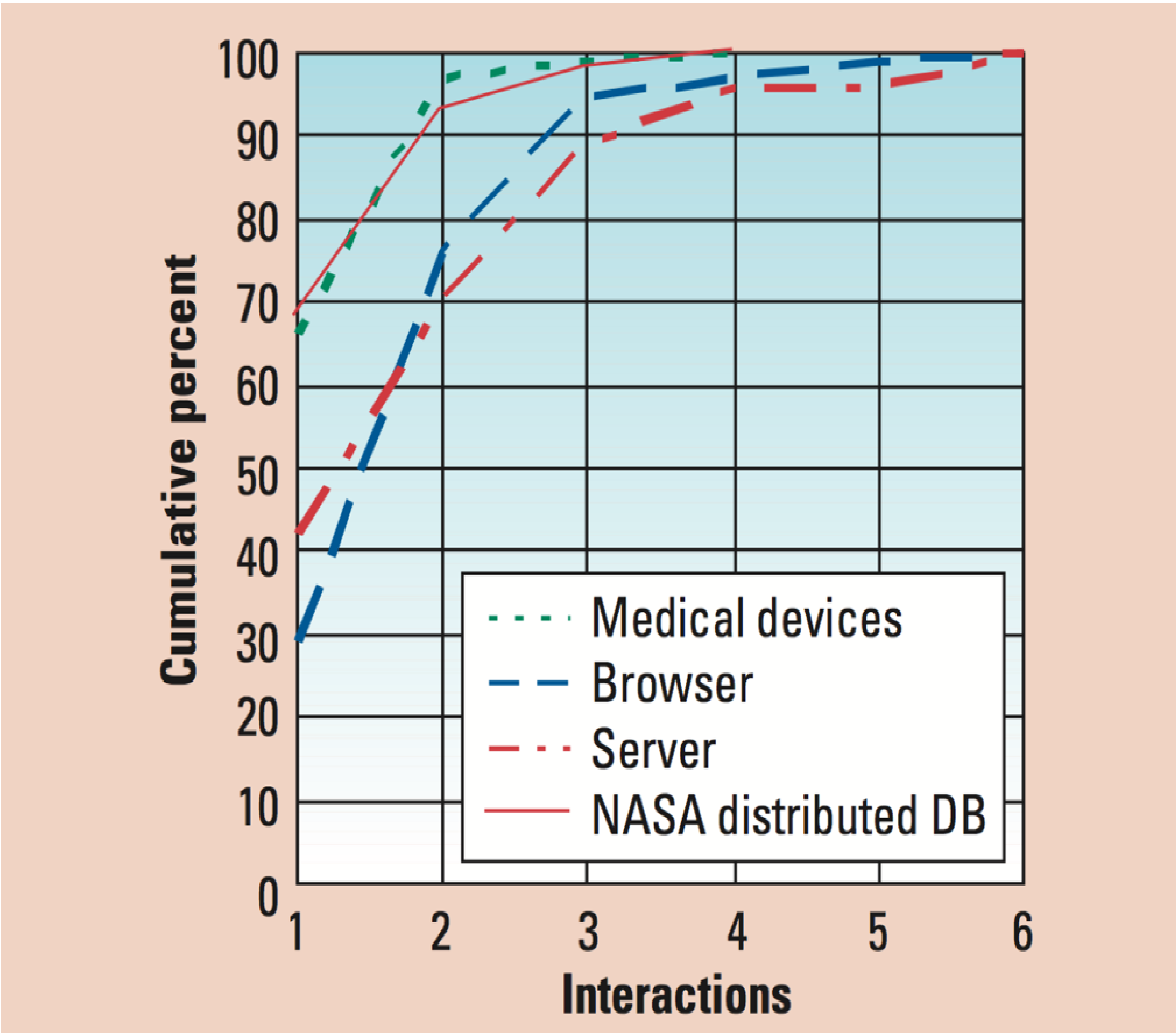
Each row is a test:

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

Each column is a parameter:



What should be the size of *t* ? /1



- An empirical study from National Institute of Standards and Technology (NIST)
- About 70% failures can be detected by a 2-way coverage (known as "**pairwise coverage**")

[http://csrc.nist.gov/groups/SNS/acts/index.html]

What should be the size of t ?   /2

- 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
- Reasonable evidence that maximum interaction strength for fault triggering is relatively small.

Quiz

- "Of the six possible 2-way variable combinations, *ab*, *ac*, *ad*, *bc*, *bd*, *cd*, only__and__have all four binary values covered, so simple 2-way (pairwise) coverage for the four tests is ______%. "
- How to add more tests so that we can achieve a 100% 2-way coverage?

Each **column** is a parameter

a	b	c	d
0	0	0	0
0	1	1	0
1	0	0	1
0	1	1	1

Each **row** is a test

Example: Display Control

No constraints reduce the total number of combinations
432 (3x4x3x4x3) test cases
if we consider all combinations

Display Mode	Language	Fonts	Color	Screen size
full-graphics	English	Minimal	Monochrome	Hand-held
text-only	French	Standard	Color-map	Laptop
limited-bandwidth	Spanish	Document-loaded	16-bit	Full-size
	Portuguese		True-color	

Pairwise combinations: 17 test cases

Language	Color	Display Mode	Fonts	Screen Size
English	Monochrome	Full-graphics	Minimal	Hand-held
English	Color-map	Text-only	Standard	Full-size
English	16-bit	Limited-bandwidth	-	Full-size
English	True-color	Text-only	Document-loaded	Laptop
French	Monochrome	Limited-bandwidth	Standard	Laptop
French	Color-map	Full-graphics	Document-loaded	Full-size
French	16-bit	Text-only	Minimal	-
French	True-color	-	-	Hand-held
Spanish	Monochrome	-	Document-loaded	Full-size
Spanish	Color-map	Limited-bandwidth	Minimal	Hand-held
Spanish	16-bit	Full-graphics	Standard	Laptop
Spanish	True-color	Text-only	-	Hand-held
Portuguese	-	-	Monochrome	Text-only
Portuguese	Color-map	-	Minimal	Laptop
Portuguese	16-bit	Limited-bandwidth	Document-loaded	Hand-held
Portuguese	True-color	Full-graphics	Minimal	Full-size
Portuguese	True-color	Limited-bandwidth	Standard	Hand-held

How do we test this?

34 switches = 2^{34} = 1.7 x 10^{10} possible inputs = **1.7 x 10^{10}** tests

If only 3-way interactions, need only___33____tests

Algorithms for Combinatorial Testing /1

- Problem Formulation: For fixed *t*, *v* and *k* values, construct the smallest *t*-way covering array:
  - *t*-way covering array: for every *t* parameters, all value combinations must appear at least once in the covering array
  - *k* is the number of variables a configuration needs to specify
  - *v* is the number of possible values each of the *k* variables can take on.

Generating a **minimum covering array** is known as NP-complete problem

http://math.nist.gov/coveringarrays/

Algorithms for Combinatorial Testing /2

- Mathematical approach
  - Can yield the smallest possible covering arrays (orthogonal array) for a small number of parameters/values
- Computational approach
  - can be applied to any types of covering arrays, but consume more time
  - Commonly used methods
    - random approaches
    - greedy approaches
    - search based approaches

http://csrc.nist.gov/groups/SNS/acts/index.html

The IPO Algorithm (1)

- Builds a t-way test set in an incremental manner
  - A t-way test set is first constructed for the first t parameters,
  - Then, the test set is extended to generate a t-way test set for the first $t + 1$ parameters
  - The test set is repeatedly extended for each additional parameter.
- Two steps involved in each extension for a new parameter:
  - Horizontal growth: extends each existing test by adding one value of the new parameter
  - Vertical growth: adds new tests, if necessary

The IPO Algorithm (2)

```
Strategy In-Parameter-Order
begin
   /* for the first t parameters p1, p2 , …,  pt*/
   T := {(v1, v2, …, vt) | v1, v2, …, vt are values of p1, p2, …, Pk, respectively}
   if n = t then stop;
   /* for the remaining parameters */
   for parameter pi, i = t + 1, …, n do
   begin
      /* horizontal growth */
      for each test (v1, v2, …, vi-1) in T do
          replace it with (v1, v2, …, vi-1, vi), where vi is a value of pi
       /* vertical growth */
       while T does not cover all the interactions between pi and
            each of p1, p2, …, pi-1 do
          add a new test for p1, p2, …, pi to T;
    end
end
```

The IPO Algorithm: Example

- Consider a system with the following parameters and values:
- parameter A has values A1 and A2
- parameter B has values B1 and B2, and
- parameter C has values C1, C2, and C3

The IPO Algorithm: Example

A	B
A1	B1
A1	B2
A2	B1
A2	B2

A	B	C
A1	B1	C1
A1	B2	C2
A2	B1	C3
A2	B2	C1

Horizontal Growth

A	B	C
A1	B1	C1
A1	B2	C2
A2	B1	C3
A2	B2	C1
A2	B1	C2
A1	B2	C3

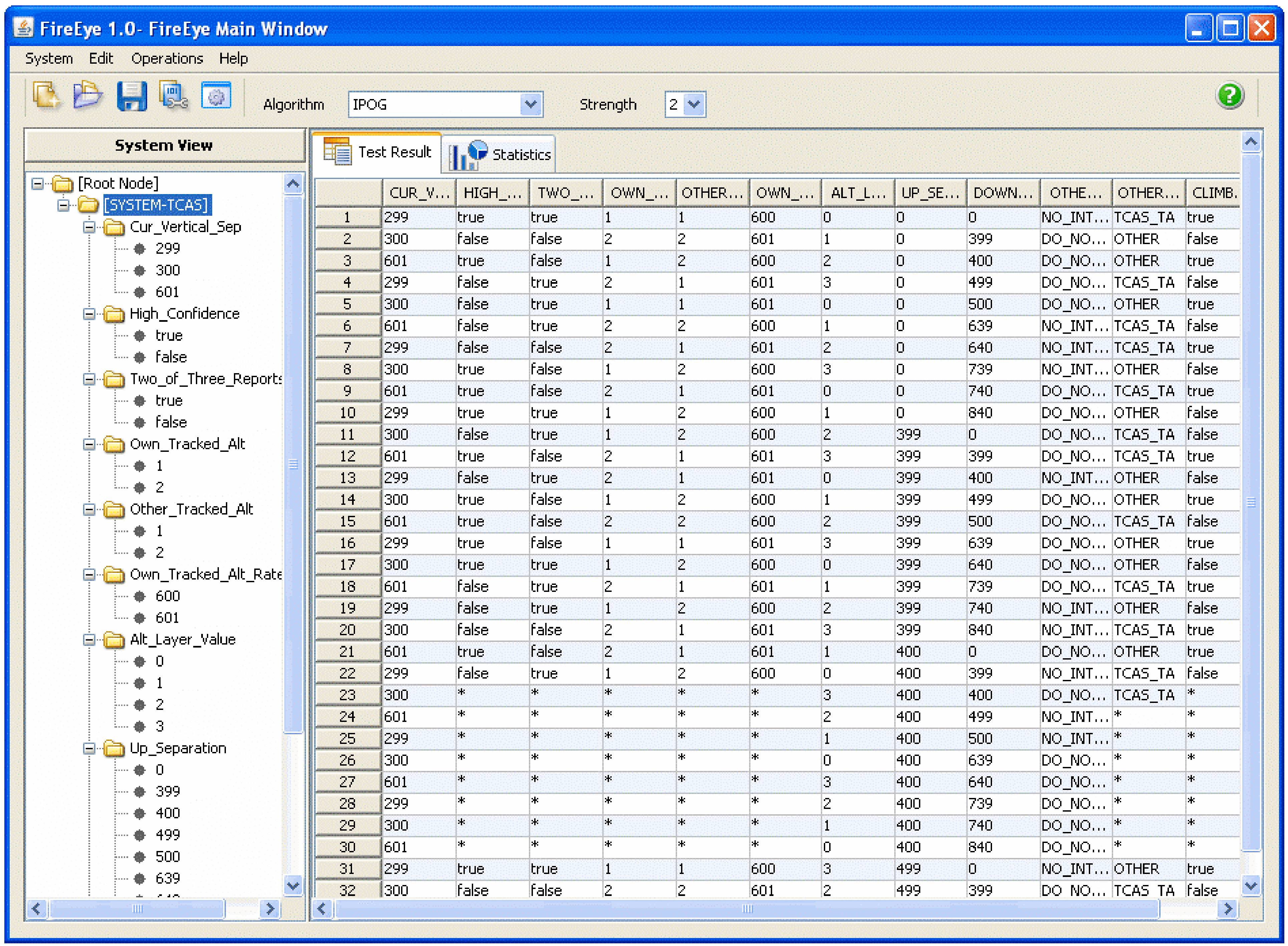
Vertical Growth

Tool Support

- National Institute of Standards and Technology
  - http://csrc.nist.gov/groups/SNS/acts/index.html
  - GUI tool - ACTS
  - Implementing the IPO algorithm
- Pairwise Testing: http://www.pairwise.org/tools.asp
- AETG (greedy algorithm, commercial tool)
- Web based tool - Hexawise (https://www.hexawise.com)

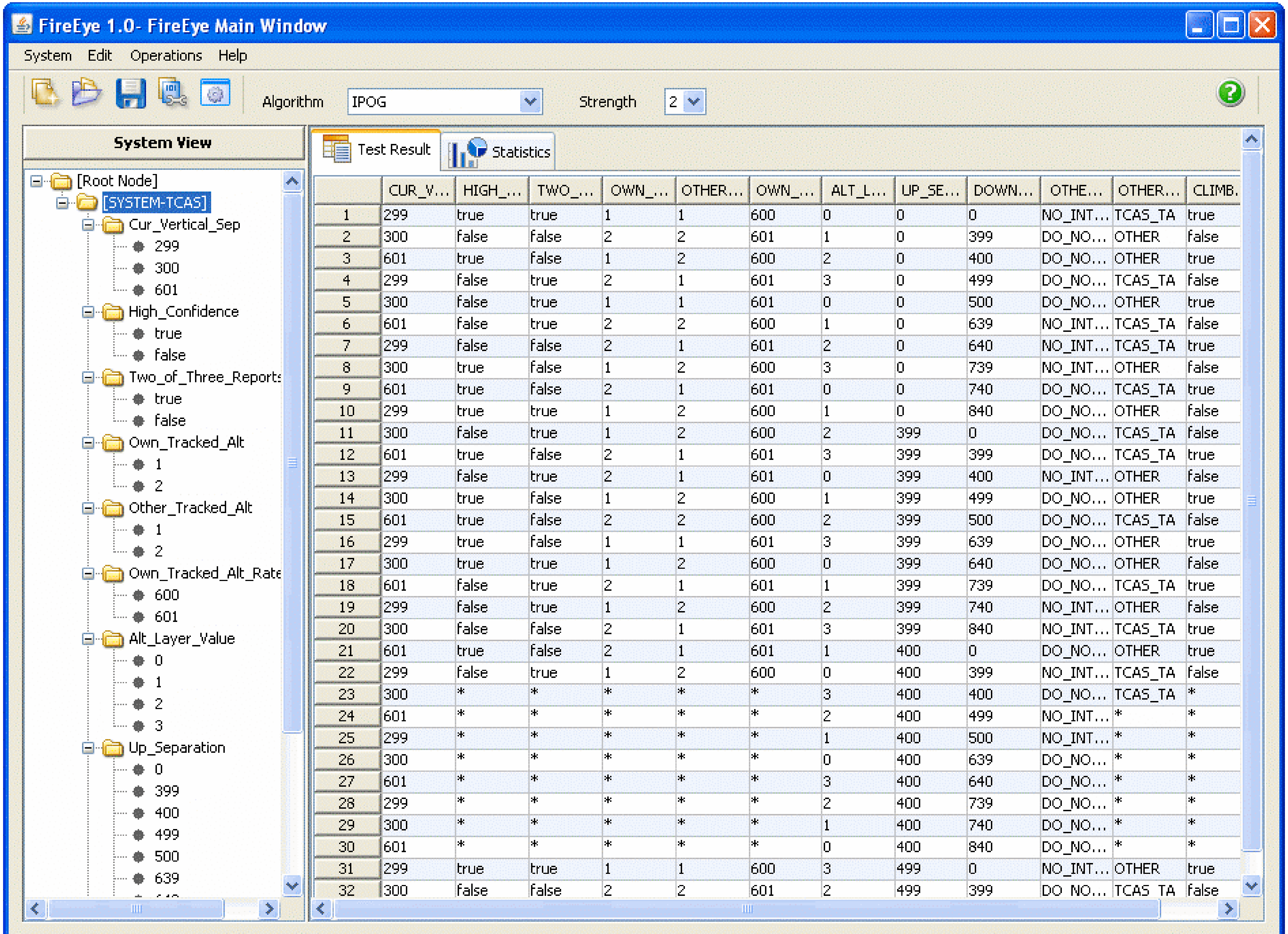
ACTS (FireEye) tool

- Publicly available at: http://csrc.nist.gov/groups/SNS/acts/download_tools.html#acts



	CUR_V...	HIGH_...	TWO_...	OWN_...	OTHER...	OWN_...	ALT_L...	UP_SE...	DOWN...	OTHE...	OTHER...	CLIMB.
1	299	true	true	1	1	600	0	0	0	NO_INT...	TCAS_TA	true
2	300	false	false	2	2	601	1	0	399	DO_NO...	OTHER	false
3	601	true	false	1	2	600	2	0	400	DO_NO...	OTHER	true
4	299	false	true	2	1	601	3	0	499	DO_NO...	TCAS_TA	false
5	300	false	true	1	1	601	0	0	500	DO_NO...	OTHER	true
6	601	false	true	2	2	600	1	0	639	NO_INT...	TCAS_TA	false
7	299	false	false	2	1	601	2	0	640	NO_INT...	TCAS_TA	true
8	300	true	false	1	2	600	3	0	739	NO_INT...	OTHER	false
9	601	true	false	2	1	601	0	0	740	DO_NO...	TCAS_TA	true
10	299	true	true	1	2	600	1	0	840	DO_NO...	OTHER	false
11	300	false	true	1	2	600	2	399	0	DO_NO...	TCAS_TA	false
12	601	true	false	2	1	601	3	399	399	DO_NO...	TCAS_TA	true
13	299	false	true	2	1	601	0	399	400	NO_INT...	OTHER	false
14	300	true	false	1	2	600	1	399	499	DO_NO...	OTHER	true
15	601	true	false	2	2	600	2	399	500	DO_NO...	TCAS_TA	false
16	299	true	false	1	1	601	3	399	639	DO_NO...	OTHER	true
17	300	true	true	1	2	600	0	399	640	DO_NO...	OTHER	false
18	601	false	true	2	1	601	1	399	739	DO_NO...	TCAS_TA	true
19	299	false	true	1	2	600	2	399	740	NO_INT...	OTHER	false
20	300	false	false	2	1	601	3	399	840	NO_INT...	TCAS_TA	true
21	601	true	false	2	1	601	1	400	0	DO_NO...	OTHER	true
22	299	false	true	1	2	600	0	400	399	NO_INT...	TCAS_TA	false
23	300	*	*	*	*	*	3	400	400	DO_NO...	TCAS_TA	*
24	601	*	*	*	*	*	2	400	499	NO_INT...	*	*
25	299	*	*	*	*	*	1	400	500	NO_INT...	*	*
26	300	*	*	*	*	*	0	400	639	DO_NO...	*	*
27	601	*	*	*	*	*	3	400	640	DO_NO...	*	*
28	299	*	*	*	*	*	2	400	739	DO_NO...	*	*
29	300	*	*	*	*	*	1	400	740	DO_NO...	*	*
30	601	*	*	*	*	*	0	400	840	DO_NO...	*	*
31	299	true	true	1	1	600	3	499	0	NO_INT...	OTHER	true
32	300	false	false	2	2	601	2	499	399	DO_NO...	TCAS_TA	false

Covering array output



	CUR_V…	HIGH_…	TWO_…	OWN_…	OTHER…	OWN_…	ALT_L…	UP_SE…	DOWN…	OTHE…	OTHER…	CLIMB.
1	299	true	true	1	1	600	0	0	0	NO_INT…	TCAS_TA	true
2	300	false	false	2	2	601	1	0	399	DO_NO…	OTHER	false
3	601	true	false	1	2	600	2	0	400	DO_NO…	OTHER	true
4	299	false	true	2	1	601	3	0	499	DO_NO…	TCAS_TA	false
5	300	false	true	1	1	601	0	0	500	DO_NO…	OTHER	true
6	601	false	true	2	2	600	1	0	639	NO_INT…	TCAS_TA	false
7	299	false	false	2	1	601	2	0	640	NO_INT…	TCAS_TA	true
8	300	true	false	1	2	600	3	0	739	NO_INT…	OTHER	false
9	601	true	false	2	1	601	0	0	740	DO_NO…	TCAS_TA	true
10	299	true	true	1	2	600	1	0	840	DO_NO…	OTHER	false
11	300	false	true	1	2	600	2	399	0	DO_NO…	TCAS_TA	false
12	601	true	false	2	1	601	3	399	399	DO_NO…	TCAS_TA	true
13	299	false	true	2	1	601	0	399	400	NO_INT…	OTHER	false
14	300	true	false	1	2	600	1	399	499	DO_NO…	OTHER	true
15	601	true	false	2	2	600	2	399	500	DO_NO…	TCAS_TA	false
16	299	true	false	1	1	601	3	399	639	DO_NO…	OTHER	true
17	300	true	true	1	2	600	0	399	640	DO_NO…	OTHER	false
18	601	false	true	2	1	601	1	399	739	DO_NO…	TCAS_TA	true
19	299	false	true	1	2	600	2	399	740	NO_INT…	OTHER	false
20	300	false	false	2	1	601	3	399	840	NO_INT…	TCAS_TA	true
21	601	true	false	2	1	601	1	400	0	DO_NO…	OTHER	true
22	299	false	true	1	2	600	0	400	399	NO_INT…	TCAS_TA	false
23	300	*	*	*	*	*	3	400	400	DO_NO…	TCAS_TA	*
24	601	*	*	*	*	*	2	400	499	NO_INT…	*	*
25	299	*	*	*	*	*	1	400	500	NO_INT…	*	*
26	300	*	*	*	*	*	0	400	639	DO_NO…	*	*
27	601	*	*	*	*	*	3	400	640	DO_NO…	*	*
28	299	*	*	*	*	*	2	400	739	DO_NO…	*	*
29	300	*	*	*	*	*	1	400	740	DO_NO…	*	*
30	601	*	*	*	*	*	0	400	840	DO_NO…	*	*
31	299	true	true	1	1	600	3	499	0	NO_INT…	OTHER	true
32	300	false	false	2	2	601	2	499	399	DO_NO…	TCAS_TA	false

Reference

- Perry, W.E. effective methods for software testing, c2006
- McCaffrey, J.D. Configuration Testing, 2009, https://jamesmccaffrey.wordpress.com/2009/01/31/configuration-testing/
- NIST, AUTOMATED COMBINATORIAL TESTING FOR SOFTWARE (ACTS), 2015 http://csrc.nist.gov/groups/SNS/acts/index.html
- H. Wu, C. Nie, F. C. Kuo, H. Leung and C. J. Colbourn, "A Discrete Particle Swarm Optimization for Covering Array Generation," in *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 4, pp. 575-591, Aug. 2015.

Thanks!

Testing
Software
development
articles
management
design
quality
test
code
People
Project
information
risk
good
usability
web
use
Agile
system
article
check
Ruby
work
time
bug
Tools
life
testers
exploratory
introduction