Combinatorial Methods for Cybersecurity Testing

Rick Kuhn and Raghu Kacker

National Institute of
Standards and Technology
Gaithersburg, MD

IDGA Military Test and Evaluation Summit June 24, 2009

Tutorial Overview

National Institute of Standards and Technology

1. What is combinatorial testing and why is it useful?
2. Costs and volume of tests required
3. Advantages and disadvantages
4. Security testing
5. Tools

Automated Combinatorial Testing

NIST National Institute of Standards and Technology

- Goals – reduce testing cost, improve cost-benefit ratio for software assurance
- Merge automated test generation with combinatorial methods
- New algorithms and faster processors make large-scale combinatorial testing practical
- Accomplishments – huge increase in performance, scalability + proof-of-concept demonstration
- Also non-testing applications – modelling and simulation, genome

The University of Texas ARLINGTON

NIST

NC STATE

GEORGE MASON UNIVERSITY

UtahState University

UMBC

Software Engineering Institute | Carnegie Mellon

Tutorial Overview

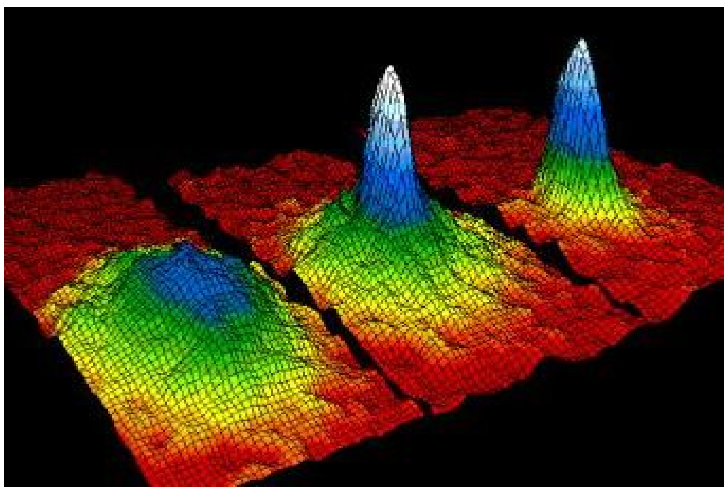
National Institute of Standards and Technology

1. **What is combinatorial testing and why is it useful?**
2. Costs and volume of tests required
3. Advantages and disadvantages
4. Security testing
5. Tools

What is NIST?

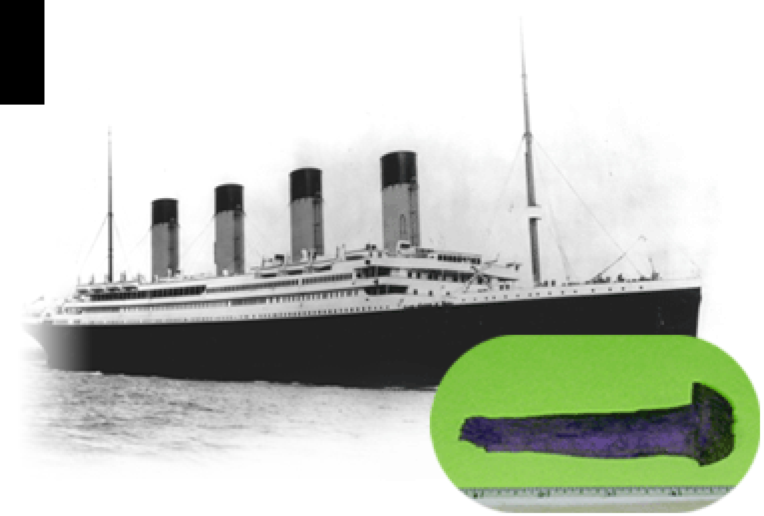
NIST
National Institute of Standards and Technology

- 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, and ...

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?
    - logic errors?
    - calculation errors?
    - inadequate input checking?   Etc.
- 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?
    - e.g., failure occurs if
      pressure < 10 (1-way interaction – found by all-values testing)
      pressure < 10 & volume > 300 (2-way interaction – found by all-pairs testing)

Pairwise testing is popular, but is it enough?

National Institute of Standards and Technology

- Pairwise testing commonly applied to software
- Intuition: some problems only occur as the result of an interaction between parameters/components
- Pairwise testing finds about 50% to 90% of flaws
  - Cohen, Dalal, Parelius, Patton, 1995 – 90% coverage with pairwise, all errors in small modules found
  - Dalal, et al. 1999 – effectiveness of pairwise testing, no higher degree interactions
  - Smith, Feather, Muscetolla, 2000 – 88% and 50% of flaws for 2 subsystems

Finding 90% of flaws is pretty good, right?

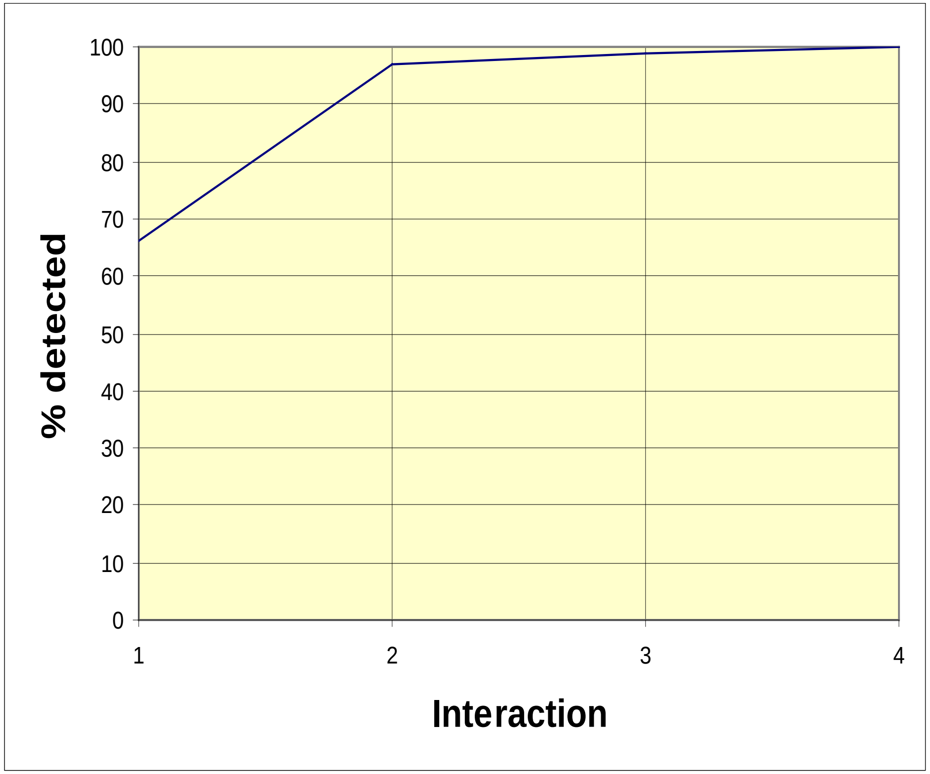
"Relax, our engineers found 90 percent of the flaws."

I don't know if I want to get on that plane.

NIST
National Institute of Standards and Technology

How about hard-to-find flaws?

- Interactions e.g., failure occurs if
- pressure < 10 (1-way interaction)
- pressure < 10 & volume > 300 (2-way interaction)
- pressure < 10 & volume > 300 & velocity = 5 (3-way interaction)
- The most complex failure reported required 4-way interaction to trigger

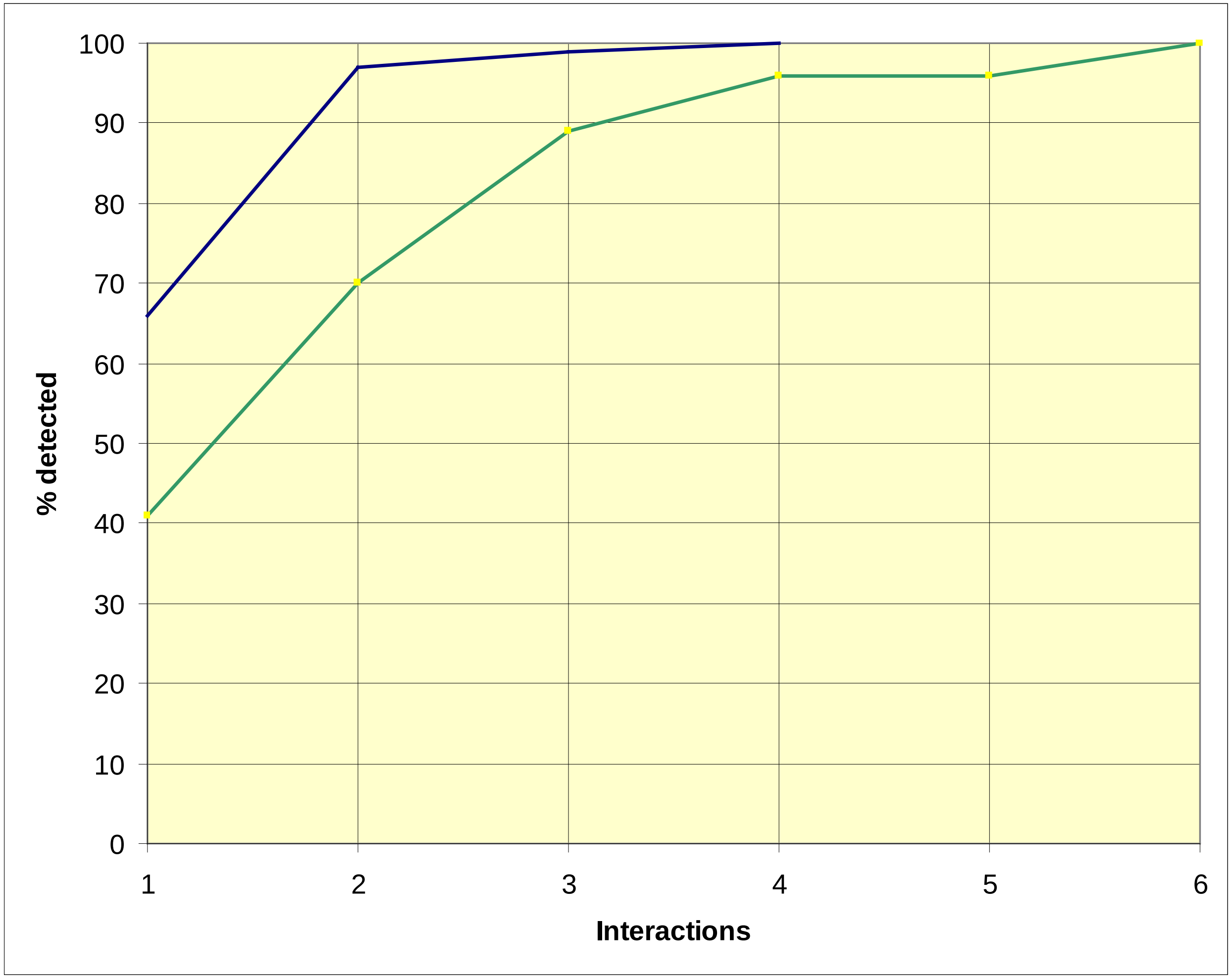


Interesting, but that's only one kind of application!

How about other applications?

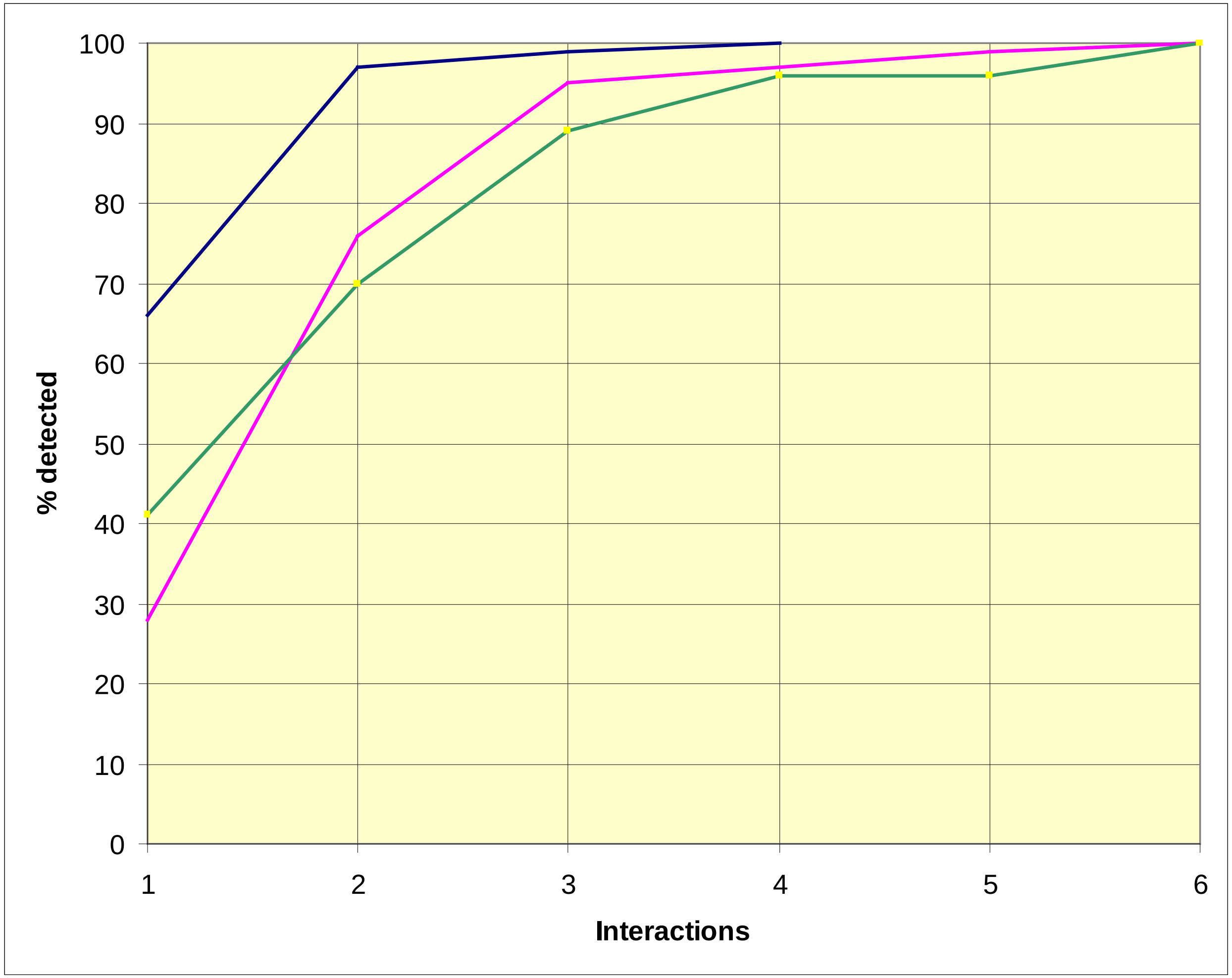
NIST
National Institute of Standards and Technology

Browser



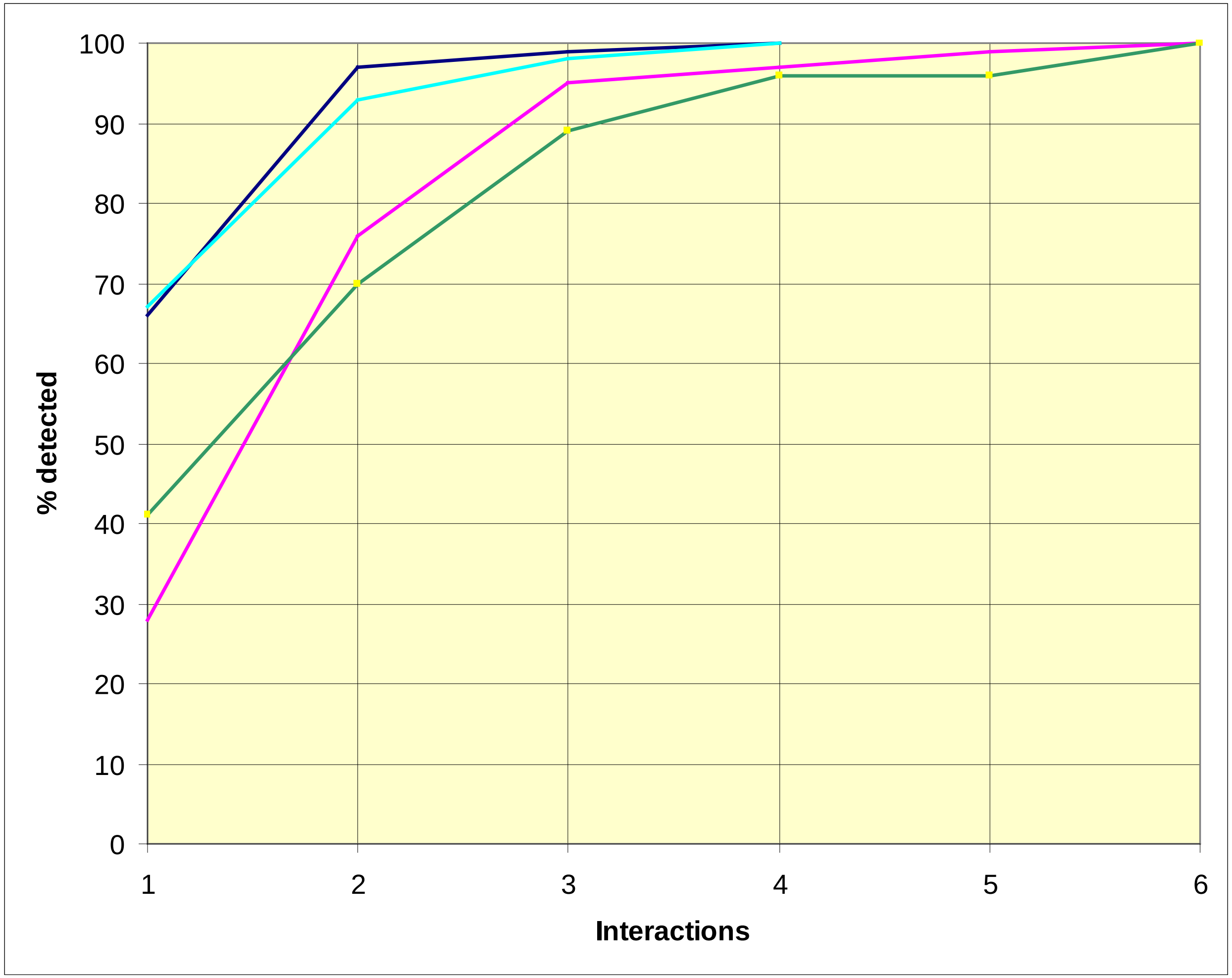
And other applications?

Server



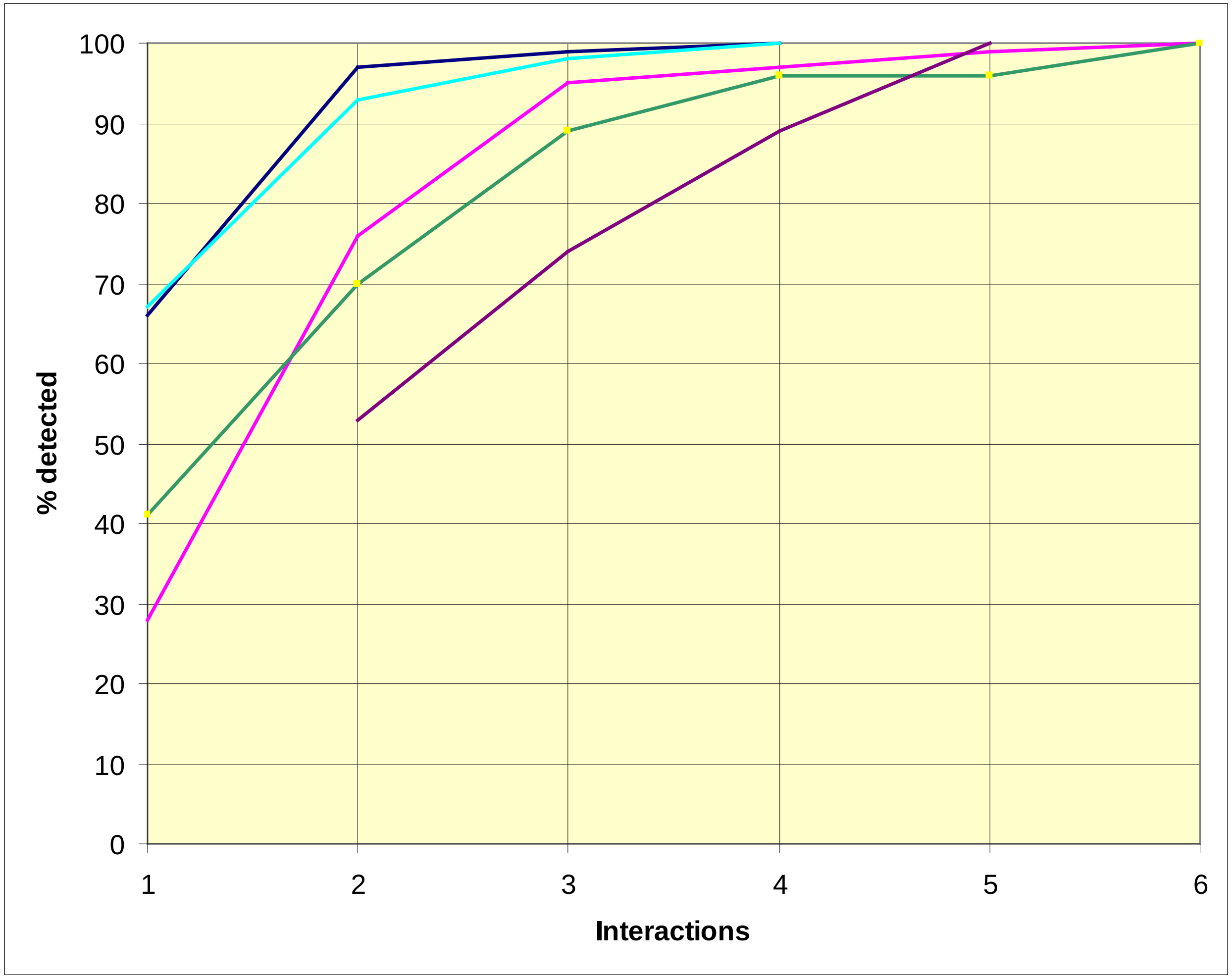
Still more?

NASA distributed database



Even more?

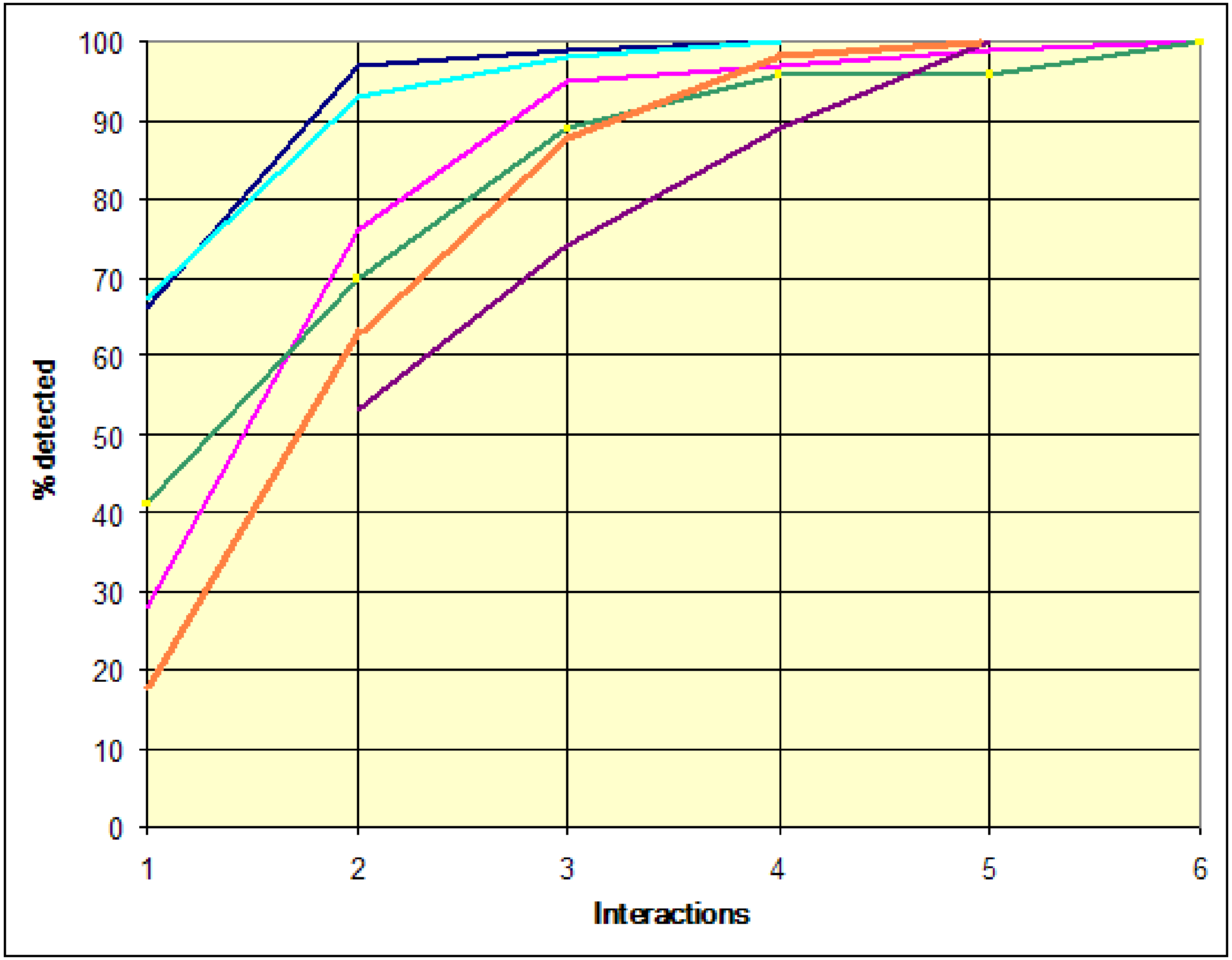
TCAS module (seeded errors)



Finally

NIST National Institute of Standards and Technology

Network security (Bell, 2006)



So, how many parameters are involved in really tricky faults?

NIST
National Institute of Standards and Technology

- Maximum interactions for fault triggering for these applications was 6
- Much more empirical work needed
- Reasonable evidence that maximum interaction strength for fault triggering is relatively small

How is this knowledge useful?

How is this knowledge useful?

- Suppose we have a system with on-off switches:

How do we test this?

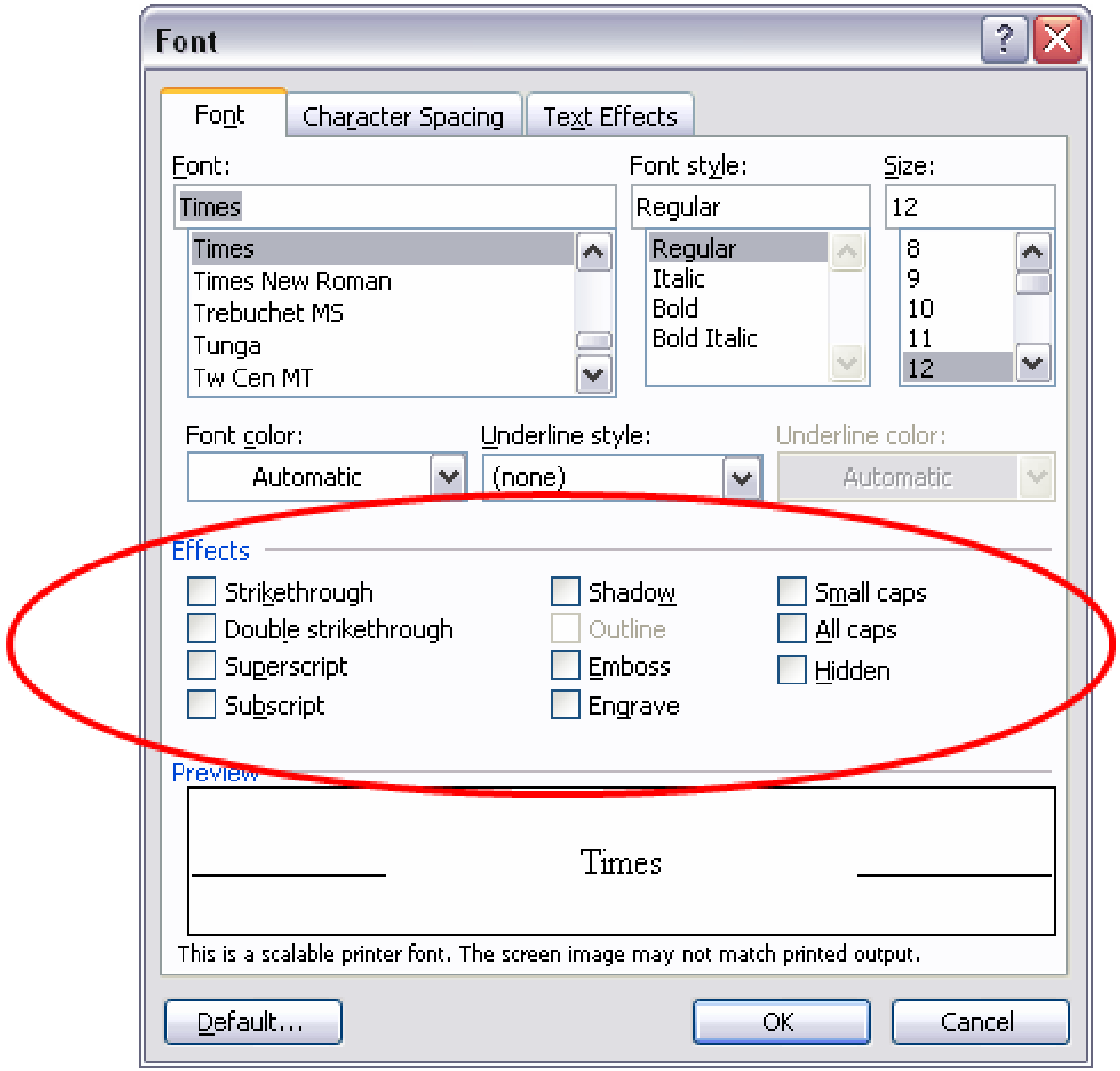
- 34 switches = 2^{34} = 1.7×10^{10} possible inputs = 1.7×10^{10} tests

What if we knew no failure involves more than 3 switch settings interacting?

- 34 switches = 2^{34} = 1.7×10^{10} possible inputs = **1.7×10^{10}** tests
- If only 3-way interactions, need only **33** tests
- For 4-way interactions, need only **85** tests

National Institute of Standards and Technology

What is combinatorial testing? A simple example



National Institute of Standards and Technology

How Many Tests Would It Take?

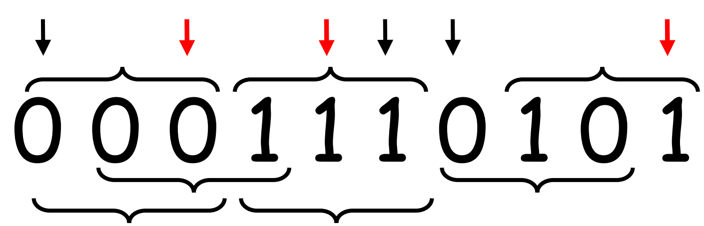
- There are 10 effects, each can be on or off
- All combinations is 2^{10} = 1,024 tests

  too many to visually check …

- Let's look at all 3-way interactions …

Now How Many Would It Take?

- There are $\binom{10}{3}$ = 120 3-way interactions.
- Naively 120 x 2^3 = 960 tests.
- Since we can pack 3 triples into each test, we need no more than 320 tests.
- Each test exercises many triples:



We oughtta be able to pack a lot in one test, so what's the smallest number we need?

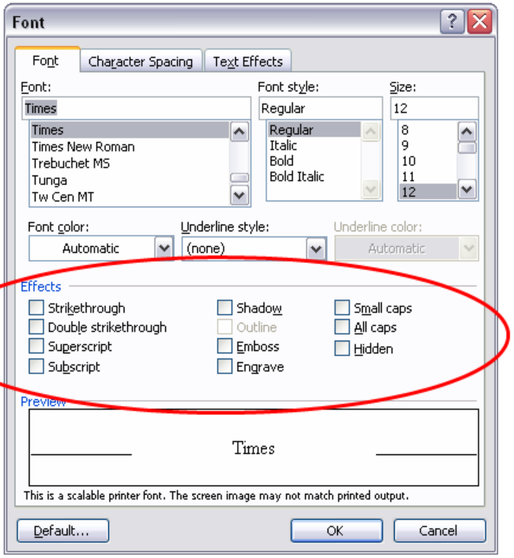
NIST National Institute of Standards and Technology

All Triples Take Only 13 Tests

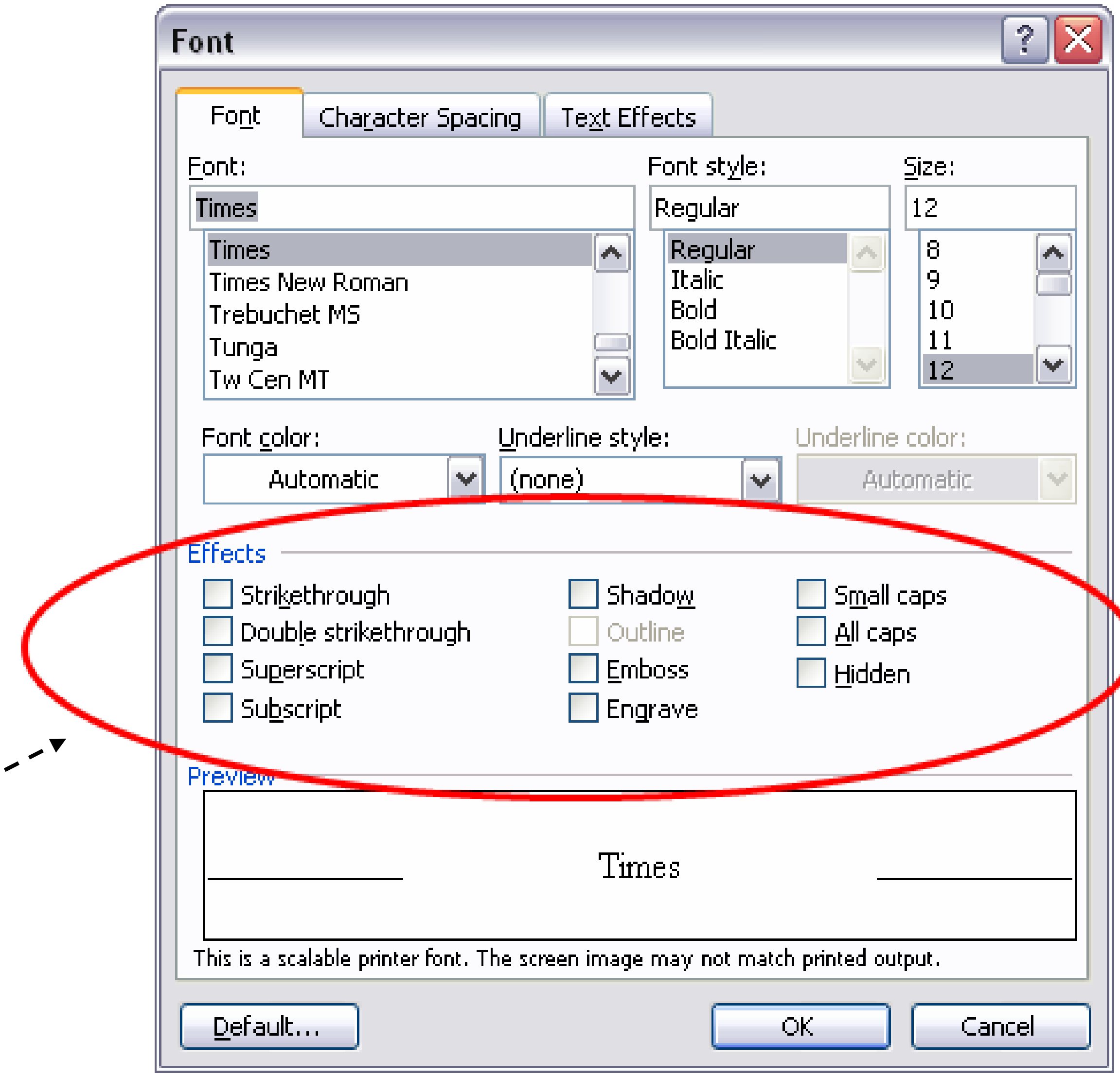
Each row is a test:

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:



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**

New algorithms to make it practical

- **Tradeoffs to minimize calendar/staff time:**
- FireEye (extended IPO) – Lei – roughly optimal, can be used for most cases under 40 or 50 parameters
  - Produces minimal number of tests at cost of run time
  - Currently integrating algebraic methods
- Adaptive distance-based strategies – Bryce – dispensing one test at a time w/ metrics to increase probability of finding flaws
  - Highly optimized covering array algorithm
  - Variety of distance metrics for selecting next test
- PRMI – Kuhn –for more variables or larger domains
  - Randomized algorithm, generates tests w/ a few tunable parameters; computation can be distributed
  - Better results than other algorithms for larger problems

NIST
National Institute of Standards and Technology

New algorithms

National Institute of Standards and Technology

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

IPOG (Lei, 06)

T-Way	IPOG		ITCH (IBM)		Jenny (Open Source)		TConfig (U. of Ottawa)		TVG (Open Source)	
	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time
2	100	0.8	120	0.73	108	0.001	108	>1 hour	101	2.75
3	400	0.36	2388	1020	413	0.71	472	>12 hour	9158	3.07
4	1363	3.05	1484	5400	1536	3.54	1476	>21 hour	64696	127
5	4226	18.41	NA	>1 day	4580	43.54	NA	>1 day	313056	1549
6	10941	65.03	NA	>1 day	11625	470	NA	>1 day	1070048	12600

Traffic Collision Avoidance System (TCAS): $2^7 3^2 4^1 10^2$

PRMI (Kuhn, 06)

	10		15		20	
	tests	sec	tests	sec	tests	sec
1 proc.	46086	390	84325	16216	114050	155964
10 proc.	46109	57	84333	11224	114102	85423
20 proc.	46248	54	84350	2986	114616	20317
FireEye	51490	168	86010	9419	**	**
Jenny	48077	18953	**	**	**	**

Table 6. 6 way, 5^k configuration results comparison
** insufficient memory

Two ways of using combinatorial testing

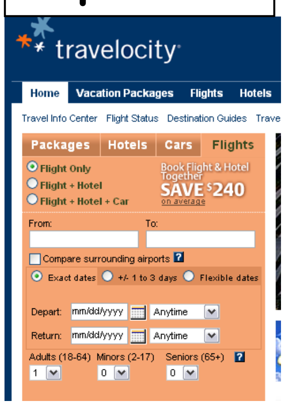
Use combinations here

or here

Test case	OS	CPU	Protocol
1	Windows	Intel	IPv4
2	Windows	AMD	IPv6
3	Linux	Intel	IPv6
4	Linux	AMD	IPv4

Configuration

Test data inputs



System under test

National Institute of Standards and Technology

Combinatorial testing with existing test set

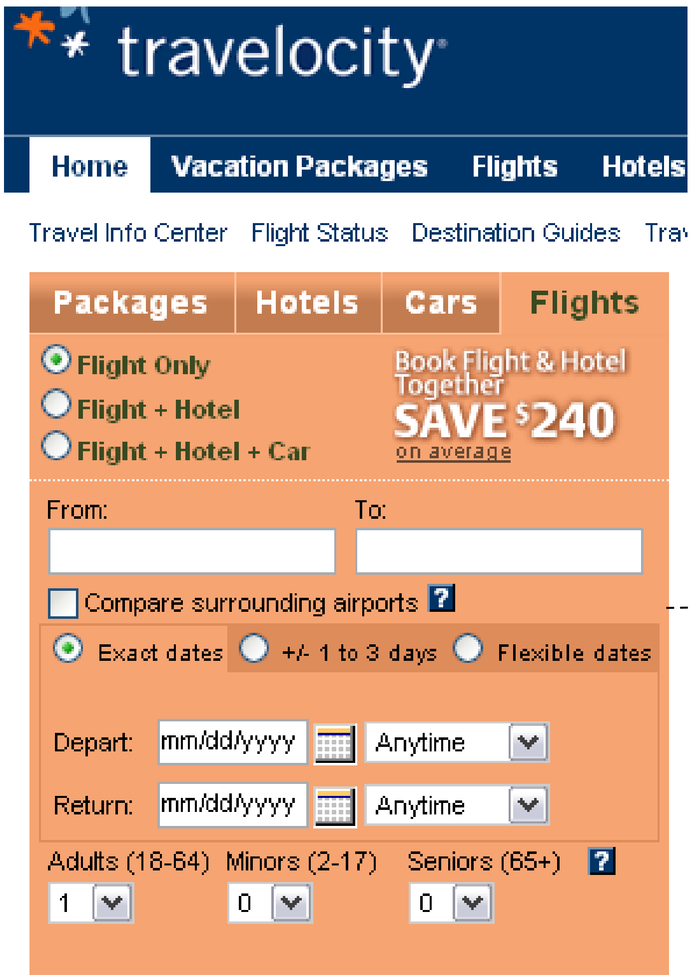
1. Use t-way coverage for system configuration values
2. Apply existing tests

Test case	OS	CPU	Protocol
1	Windows	Intel	IPv4
2	Windows	AMD	IPv6
3	Linux	Intel	IPv6
4	Linux	AMD	IPv4

- Common practice in telecom industry
- May be expensive to apply but long-run cost savings

NIST National Institute of Standards and Technology

A Real-World Example



No silver bullet because:

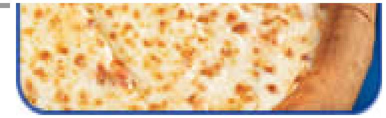
- Many values per variable
- Need to abstract values

But we can still increase information per test

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

Ordering Pizza

National Institute of Standards and Technology



Step 1 *Select your favorite size and pizza crust.*

Large Original Crust

Step 2

Select your favorite pizza toppings from the pull down. Whole toppings cover the entire pizza. First ½ and second ½ toppings cover half the pizza. For a regular cheese pizza, do not add toppings.

☑ I want to add or remove toppings on this pizza -- add on whole or half pizza.

Add toppings whole pizza — Extra Cheese (Remove), Bacon (Remove), Black Olives (Remove)

Add toppings 1st half

Add toppings 2nd half

$6 \times 2^{17} \times 2^{17} \times 2^{17} \times 4 \times 3 \times 2 \times 2 \times 5 \times 2$

= WAY TOO MUCH TO TEST

Simplified pizza ordering:

$6 \times 4 \times 4 \times 4 \times 4 \times 3 \times 2 \times 2 \times 5 \times 2$

= 184,320 possibilities

Step 3 *Select your pizza instructions.*

☑ I want to add special instructions for this pizza -- light, extra or no sauce; light or no cheese; well done bake

Regular Sauce | Normal Cheese | Normal Bake | Normal Cut

Step 4 *Add to order.*

Quantity 1

Add To Order | Add To Order & Checkout

Ordering Pizza Combinatorially

Simplified pizza ordering:

6x4x4x4x4x3x2x2x5x2
= 184,320 possibilities

2-way tests: 32

3-way tests: 150

4-way tests: 570

5-way tests: 2,413

6-way tests: 8,330

If all failures involve 5 or fewer parameters, then we can have confidence after running all 5-way tests.

So what? Who has time to check 2,413 test results?

How to automate checking correctness of output

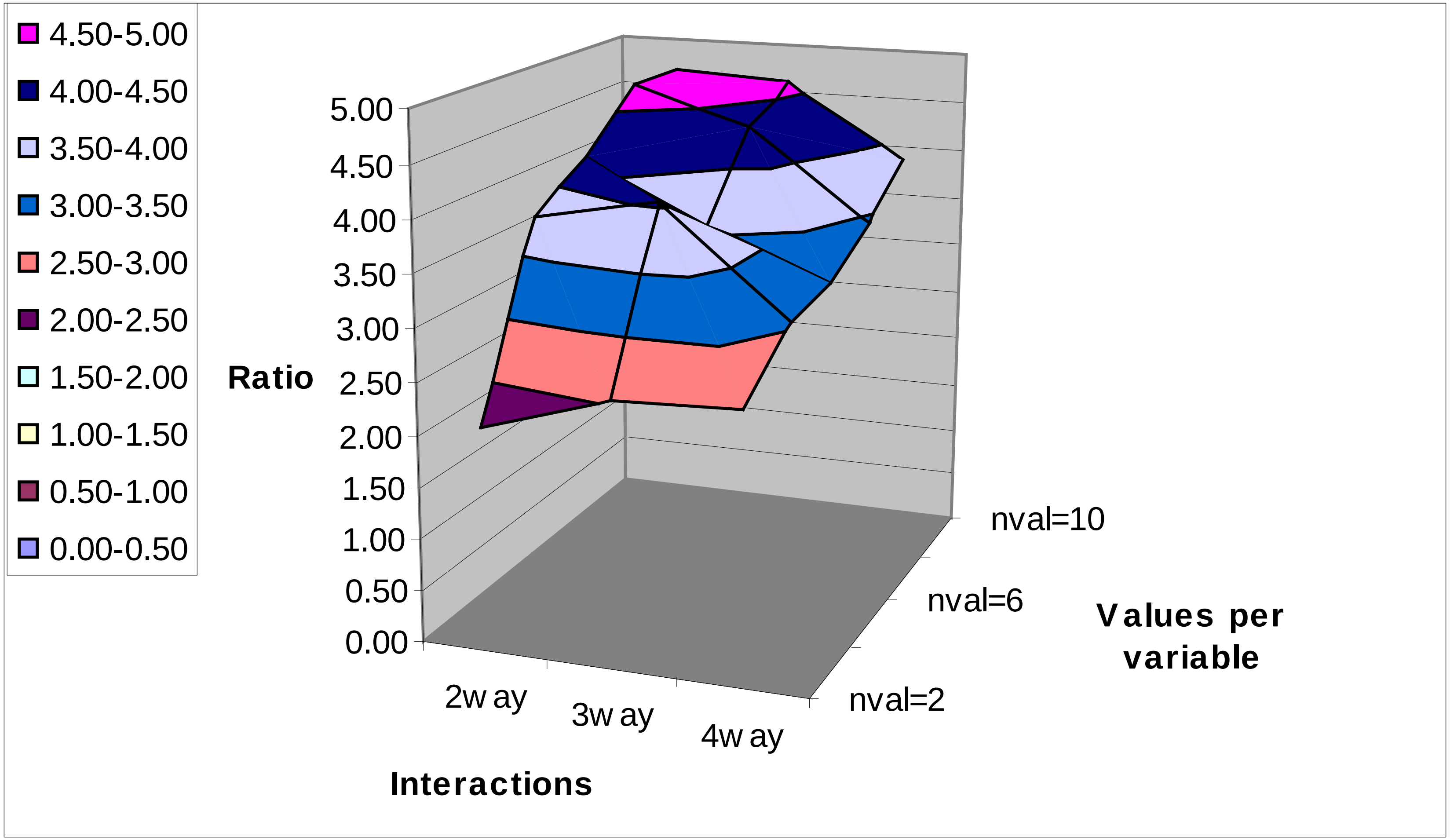
- Creating test data is the easy part!
- How do we check that the code worked correctly on the test input?
  - **Crash testing** server or other code to ensure it does not crash for any test input (like 'fuzz testing')
    - Easy but limited value
  - **Embedded assertions** – incorporate assertions 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

NIST
National Institute of Standards and Technology

Crash Testing

- Like “fuzz testing” - send packets or other input to application, watch for crashes
- Unlike fuzz testing, input is non-random; cover all t-way combinations
- May be more efficient - random input generation requires several times as many tests to cover the t-way combinations in a covering array

Ratio of Random/Combinatorial Test Set Required to Provide t-way Coverage



Crash Testing Bottom Line

- Limited utility, but can detect high-risk problems such as:
  - buffer overflow
  - server crashes

Embedded Assertions

Simple example:

```
assert( x != 0);    // ensure divisor is not zero
```

Or pre and post-conditions:

```
requires amount >= 0;

ensures balance  == \old(balance) - amount &&
\result == balance;
```

NIST
National Institute of Standards and Technology

Embedded Assertions

Assertions check properties of expected result:

```
ensures balance  == \old(balance) - amount
&&  \result == balance;
```

- Reasonable assurance that code works correctly across the range of expected inputs
- May identify problems with handling unanticipated inputs
- Example: Smart card testing
  - Used Java Modeling Language (JML) assertions
  - Detected 80% to 90% of flaws

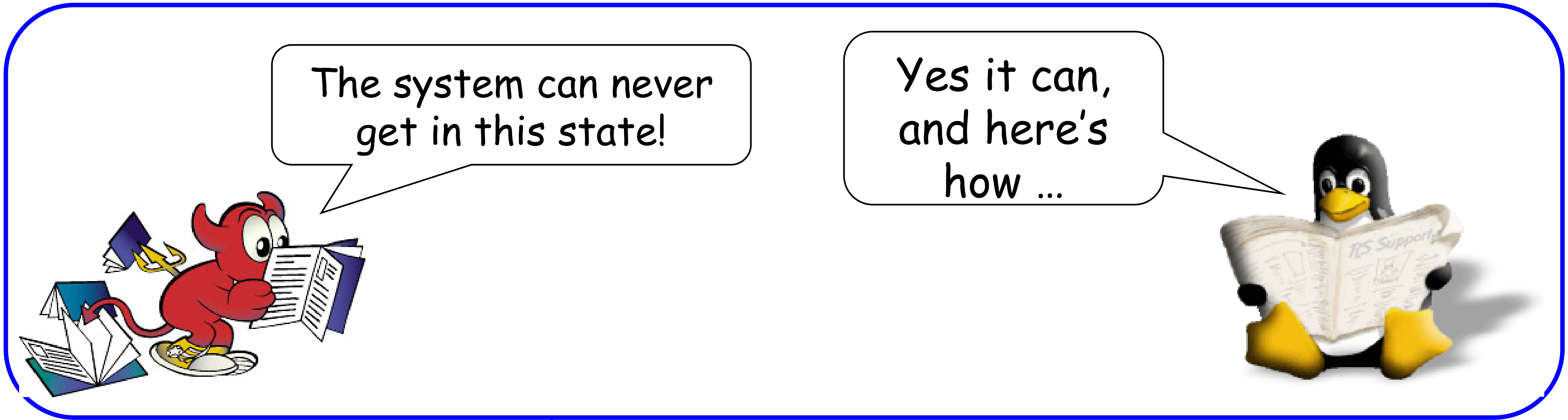
National Institute of Standards and Technology

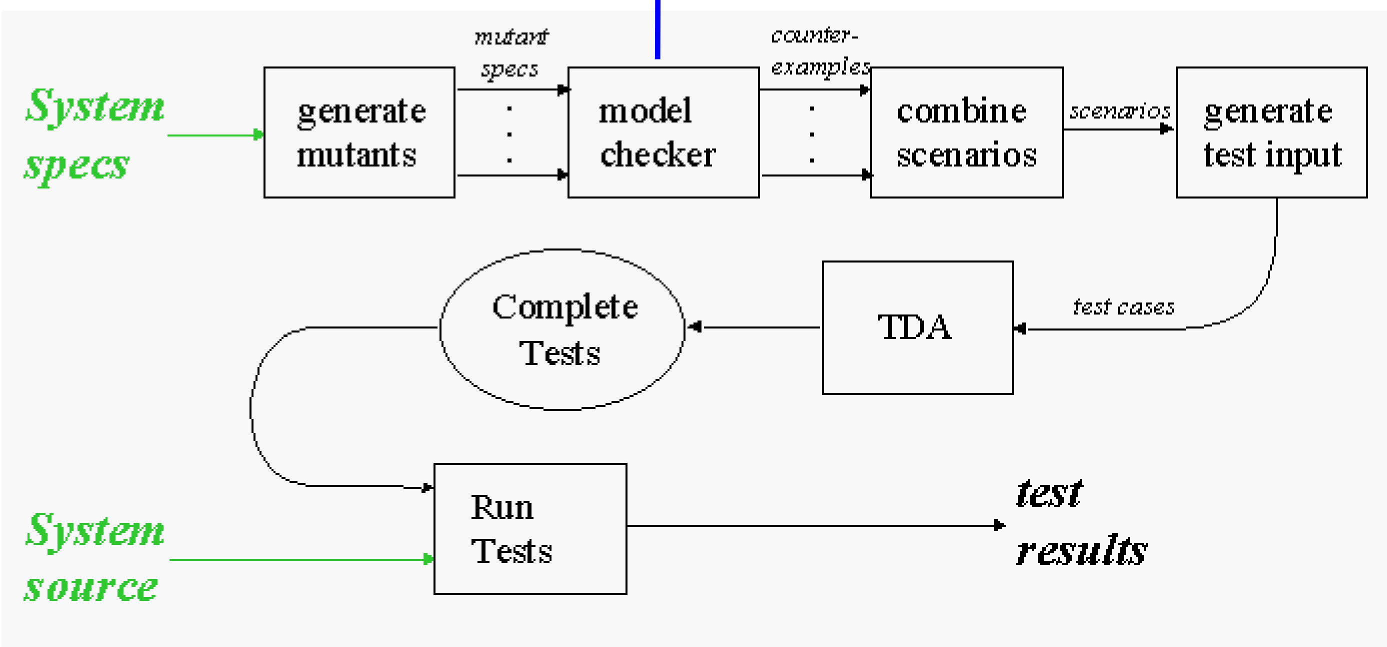
Model checking example

```
-- specification for a portion of tcas - altitude separation.
-- The corresponding C code is originally from Siemens Corp. Research
-- Vadim Okun 02/2002
MODULE main
VAR
  Cur_Vertical_Sep : { 299, 300, 601 };
  High_Confidence : boolean;
...
init(alt_sep) := START_;
  next(alt_sep) := case
    enabled & (intent_not_known | !tcas_equipped) : case
      need_upward_RA & need_downward_RA : UNRESOLVED;
      need_upward_RA : UPWARD_RA;
      need_downward_RA : DOWNWARD_RA;
      1 : UNRESOLVED;
    esac;
    1 : UNRESOLVED;
  esac;
...
SPEC AG ((enabled & (intent_not_known | !tcas_equipped) &
 !need_downward_RA & need_upward_RA) -> AX (alt_sep = UPWARD_RA))
-- “FOR ALL executions,
 -- IF enabled & (intent_not_known ....
 -- THEN in the next state alt_sep = UPWARD_RA”
```

NIST National Institute of Standards and Technology

Using model checking to produce tests





Black & Ammann, 1999

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

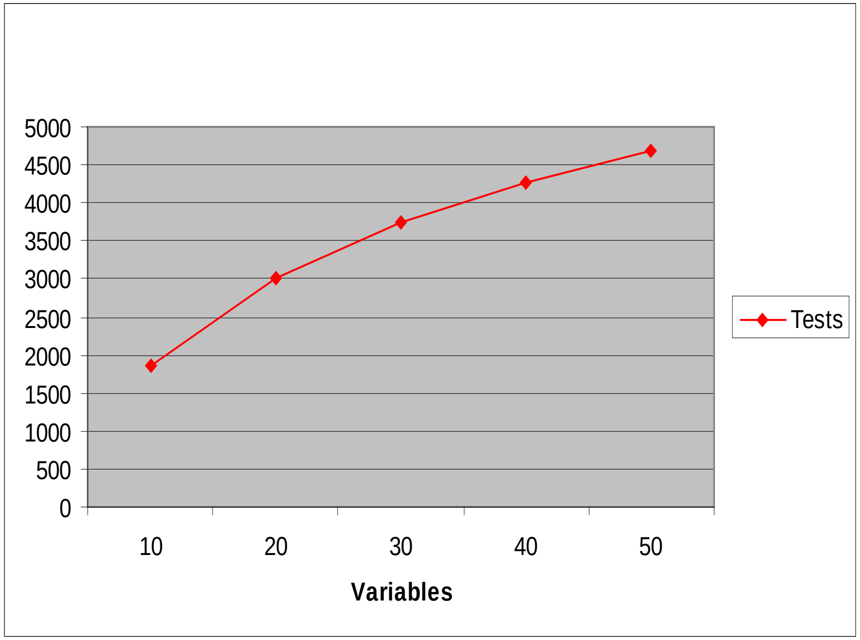
Tutorial Overview

National Institute of Standards and Technology

1. What is combinatorial testing and why is it useful?
2. **Costs and volume of tests required**
3. Advantages and disadvantages
4. Security testing
5. Tools

Cost and Volume of Tests

- Number of tests: proportional to $v^t \log n$
- *Thus:*
  - Tests increase exponentially with interaction strength t - BAD, but unavoidable
  - But only logarithmically with the number of parameters - GOOD!
- Example: suppose we want all 4-way combinations of *n* parameters, 5 values each:

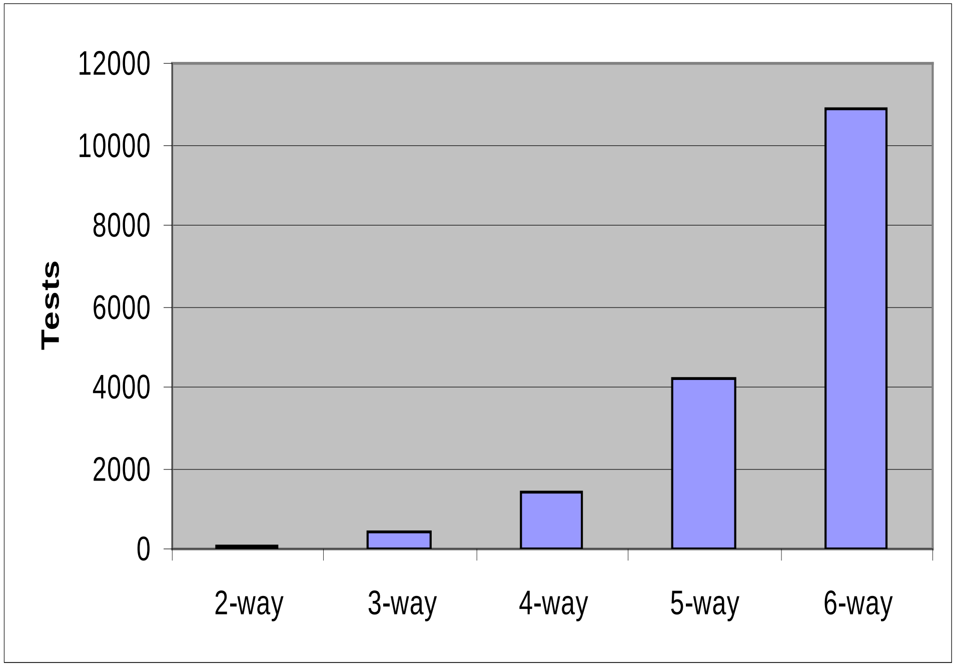


Example

- Traffic Collision Avoidance System (TCAS) module
  - Used in previous testing research
  - 41 versions seeded with errors
  - 12 variables: 7 boolean, two 3-value, one 4-value, two 10-value
  - All flaws found with 5-way coverage
  - Thousands of tests - generated by model checker in a few minutes

Tests generated

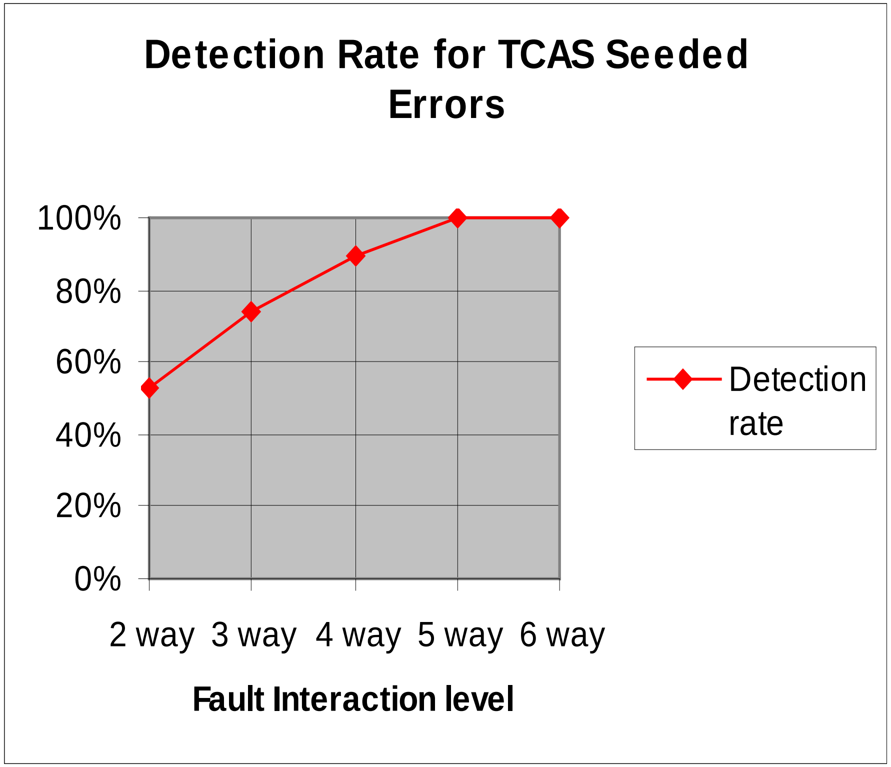
t	Test cases
2-way:	156
3-way:	461
4-way:	1,450
5-way:	4,309
6-way:	11,094

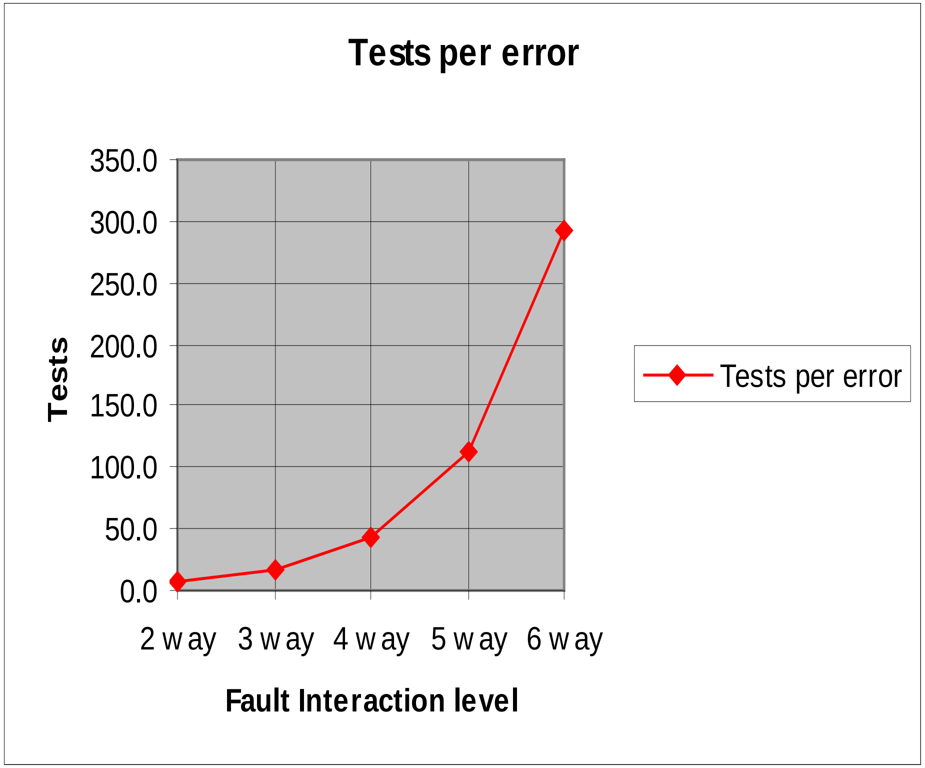


Results

NIST
National Institute of Standards and Technology

- Roughly consistent with data on large systems
- But errors harder to detect than real-world examples





Bottom line for model checking based combinatorial testing:
Expensive but can be highly effective

Tutorial Overview

National Institute of Standards and Technology

1. What is combinatorial testing and why is it useful?
2. Costs and volume of tests required
3. **Advantages and disadvantages**
4. Security testing
5. Tools

Where does this stuff make sense?

- More than (roughly) 7 or 8 parameters and less than 300, depending on interaction strength desired
- 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, so interaction testing is pointless (but we don't usually know this up front)

Examples

- sqrt(x) NO
- amortization_schedule(amt, rate, months) NO
- web e-commerce YES
- communication protocols YES

Useful when you have a lot of fields with multiple values

Tradeoffs

National Institute of Standards and Technology

- Advantages
  - Tests rare conditions
  - Produces high code coverage
  - Finds faults faster
  - May be lower overall testing cost
- Disadvantages
  - Very expensive at higher strength interactions (>4-way)
  - May require high skill level in some cases (if formal models are being used)

Tutorial Overview

1. What is combinatorial testing and why is it useful?
2. Costs and volume of tests required
3. Advantages and disadvantages
4. **Security testing**
5. Tools

Buffer Overflows

- Empirical data from the National Vulnerability Database
  - Investigated > 3,000 denial-of-service vulnerabilities reported in the NIST NVD for period of 10/06 – 3/07
  - Vulnerabilities triggered by:
    - Single variable – 94.7%
      example: *Heap-based buffer overflow in the SFTP protocol handler for Panic Transmit … allows remote attackers to execute arbitrary code via a long ftps:// URL.*
    - 2-way interaction – 4.9%
      example: *single character search string in conjunction with a single character replacement string, which causes an "off by one overflow"*
    - 3-way interaction – 0.4%
      example: *Directory traversal vulnerability when register_globals is enabled and magic_quotes is disabled and .. (dot dot) in the page parameter*

Finding Buffer Overflows

```
  if (strcmp(conn[sid].dat->in_RequestMethod, "POST")==0) {
         if (conn[sid].dat->in_ContentLength<MAX_POSTSIZE) {
……
  conn[sid].PostData=calloc(conn[sid].dat->in_ContentLength+1024,
sizeof(char));
                ……
          pPostData=conn[sid].PostData;
          do {
                rc=recv(conn[sid].socket, pPostData, 1024, 0);
          ……
                pPostData+=rc;
                x+=rc;
          } while ((rc==1024)||(x<conn[sid].dat->in_ContentLength));
  conn[sid].PostData[conn[sid].dat->in_ContentLength]='\0';
   }
```

Interaction: request-method="POST", content-length = -1000, data= a string > 24 bytes

```
if (strcmp(conn[sid].dat->in_RequestMethod, "POST")==0) {
        if (conn[sid].dat->in_ContentLength<MAX_POSTSIZE) {
  ……
  conn[sid].PostData=calloc(conn[sid].dat->in_ContentLength+1024,
  sizeof(char));
                ……
          pPostData=conn[sid].PostData;
          do {
                  rc=recv(conn[sid].socket, pPostData, 1024, 0);
              ……
                  pPostData+=rc;
                  x+=rc;
          } while ((rc==1024)||(x<conn[sid].dat->in_ContentLength));
  conn[sid].PostData[conn[sid].dat->in_ContentLength]='\0';
  }
```

Interaction: request-method="POST", content-length = -1000, data= a string > 24 bytes

true branch

```
if (strcmp(conn[sid].dat->in_RequestMethod, "POST")==0) {
        if (conn[sid].dat->in_ContentLength<MAX_POSTSIZE) {
  ......
  conn[sid].PostData=calloc(conn[sid].dat->in_ContentLength+1024,
  sizeof(char));
              ......
         pPostData=conn[sid].PostData;
         do {
              rc=recv(conn[sid].socket, pPostData, 1024, 0);
         ......
              pPostData+=rc;
              x+=rc;
         } while ((rc==1024)||(x<conn[sid].dat->in_ContentLength));
  conn[sid].PostData[conn[sid].dat->in_ContentLength]='\0';
  }
```

Interaction: request-method="POST", content-length = -1000, data= a string > 24 bytes

```
if (strcmp(conn[sid].dat->in_RequestMethod, "POST")==0) {
        if (conn[sid].dat->in_ContentLength<MAX_POSTSIZE) {
  ......
        conn[sid].PostData=calloc(conn[sid].dat->in_ContentLength+1024, sizeof(char));
                ......
          pPostData=conn[sid].PostData;
          do {
                rc=recv(conn[sid].socket, pPostData, 1024, 0);
            ......
                pPostData+=rc;
                x+=rc;
          } while ((rc==1024)||(x<conn[sid].dat->in_ContentLength));
  conn[sid].PostData[conn[sid].dat->in_ContentLength]='\0';
 }
```

true branch

National Institute of Standards and Technology

Interaction: request-method="POST", content-length = -1000, data= a string > 24 bytes

```
if (strcmp(conn[sid].dat->in_RequestMethod, "POST")==0) {
        if (conn[sid].dat->in_ContentLength<MAX_POSTSIZE) {
  ......
        conn[sid].PostData=calloc(conn[sid].dat->in_ContentLength+1024,
  sizeof(char));
                ......
         pPostData=conn[sid].PostData;
         do {
              rc=recv(conn[sid].socket, pPostData, 1024, 0);
            ......
              pPostData+=rc;
              x+=rc;
         } while ((rc==1024)||(x<conn[sid].dat->in_ContentLength));
conn[sid].PostData[conn[sid].dat->in_ContentLength]='\0';
 }
```

true branch

Allocate -1000 + 1024 bytes = 24 bytes

Interaction: request-method="POST", content-length = -1000, data= a string > 24 bytes

NIST National Institute of Standards and Technology

```
if (strcmp(conn[sid].dat->in_RequestMethod, "POST")==0) {
        if (conn[sid].dat->in_ContentLength<MAX_POSTSIZE) {
  ......
        conn[sid].PostData=calloc(conn[sid].dat->in_ContentLength+1024,
  sizeof(char));
            ......
         pPostData=conn[sid].PostData;
         do {
              rc=recv(conn[sid].socket, pPostData, 1024, 0)
         ......
              pPostData+=rc;
              x+=rc;
         } while ((rc==1024)||(x<conn[sid].dat->in_ContentLength));
conn[sid].PostData[conn[sid].dat->in_ContentLength]='\0';
 }
```

true branch

Allocate -1000 + 1024 bytes = 24 bytes

Boom!

Network Deadlock Detection

- "Simured" network simulator
  - Kernel of ~ 5,000 lines of C++ (not including GUI)
- Objective: detect configurations that can produce deadlock:
  - Prevent connectivity loss when changing network
  - Attacks that could lock up network
- Compare effectiveness of random vs. combinatorial inputs
- Deadlock combinations discovered
- Crashes in >6% of tests w/ valid values (Win32 version only)

NIST
National Institute of Standards and Technology

Network Deadlock Detection

Parameter		Values
1	DIMENSIONS	1,2,4,6,8
2	NODOSDIM	2,4,6
3	NUMVIRT	1,2,3,8
4	NUMVIRTINJ	1,2,3,8
5	NUMVIRTEJE	1,2,3,8
6	LONBUFFER	1,2,4,6
7	NUMDIR	1,2
8	FORWARDING	0,1
9	PHYSICAL	true, false
10	ROUTING	0,1,2,3
11	DELFIFO	1,2,4,6
12	DELCROSS	1,2,4,6
13	DELCHANNEL	1,2,4,6
14	DELSWITCH	1,2,4,6

5x3x4x4x4x4x2x2x2x4x4x4x4x4
= 31,457,280
configurations

Are any of them dangerous?

If so, how many?

Which ones?

NIST
National Institute of Standards and Technology

Network Deadlock Detection

Deadlocks Detected - combinatorial

t	Tests	500 pkts	1000 pkts	2000 pkts	4000 pkts	8000 pkts
2	28	0	0	0	0	0
3	161	2	3	2	3	3
4	752	14	14	14	14	14

Average Deadlocks Detected – random

t	Tests	500 pkts	1000 pkts	2000 pkts	4000 pkts	8000 pkts
2	28	0.63	0.25	0.75	0. 50	0. 75
3	161	3	3	3	3	3
4	752	10.13	11.75	10.38	13	13.25

Network Deadlock Detection

Detected 14 configurations that can cause deadlock:

$14/ 31{,}457{,}280 = 4.4 \times 10^{-7}$

Combinatorial testing found more deadlocks than random, including some that might never have been found with random testing

Risks:

- accidental deadlock configuration: low
- deadlock config discovered by attacker: **much higher**

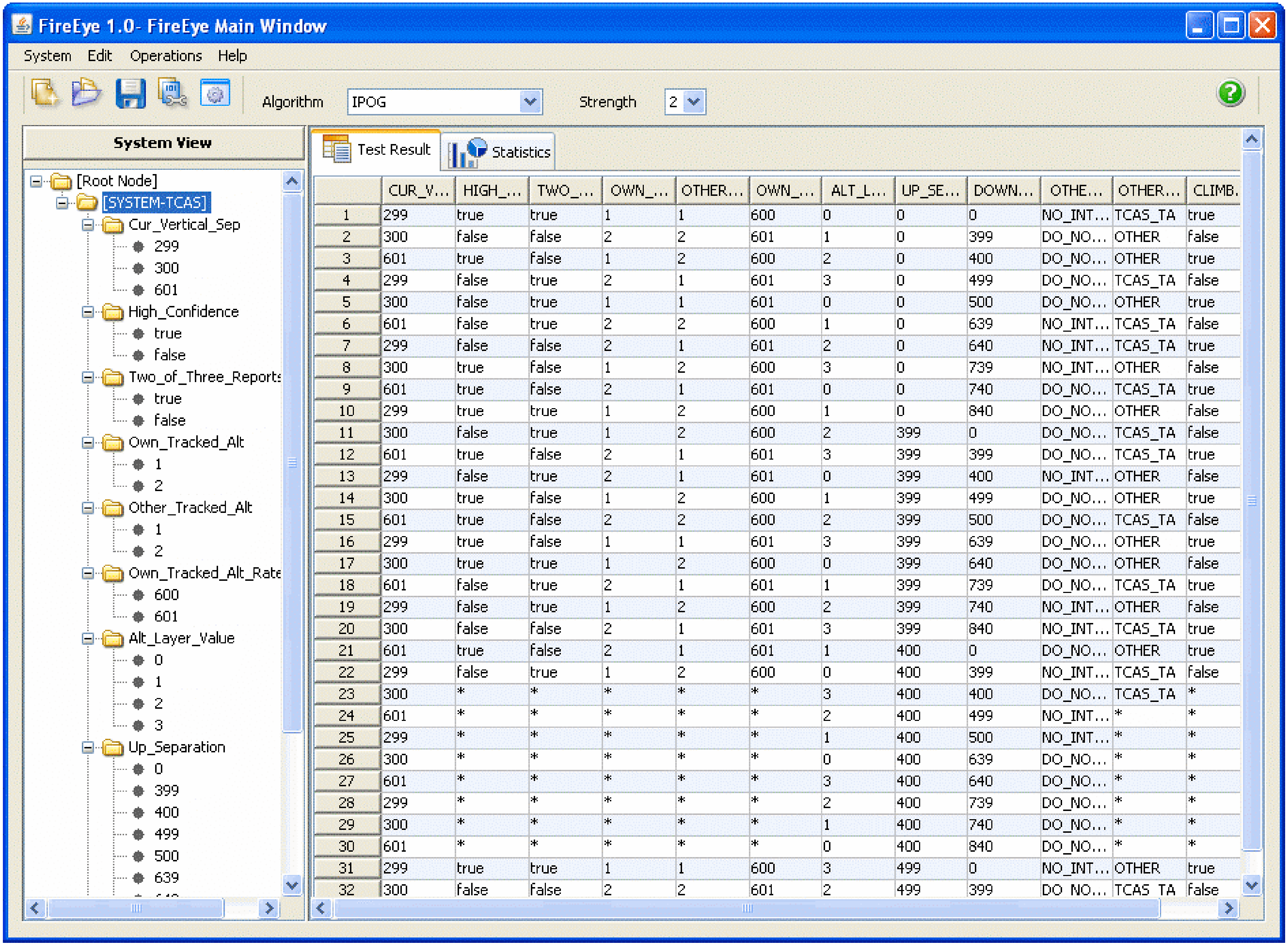
(because they are looking for it)

Tutorial Overview

1. What is combinatorial testing and why is it useful?
2. Costs and volume of tests required
3. Advantages and disadvantages
4. Security testing
5. **Tools**

ACTS Tool

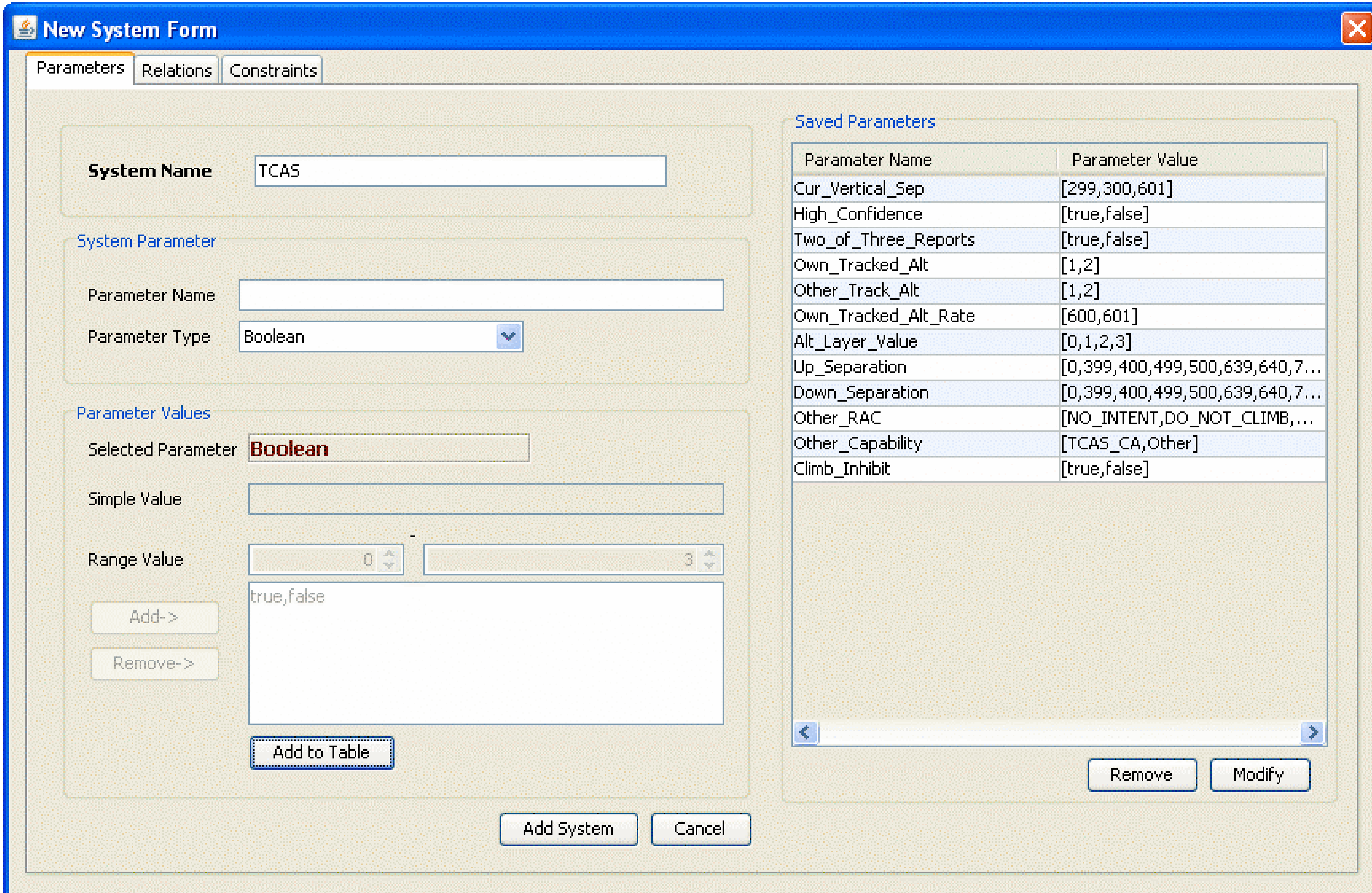
National Institute of Standards and Technology



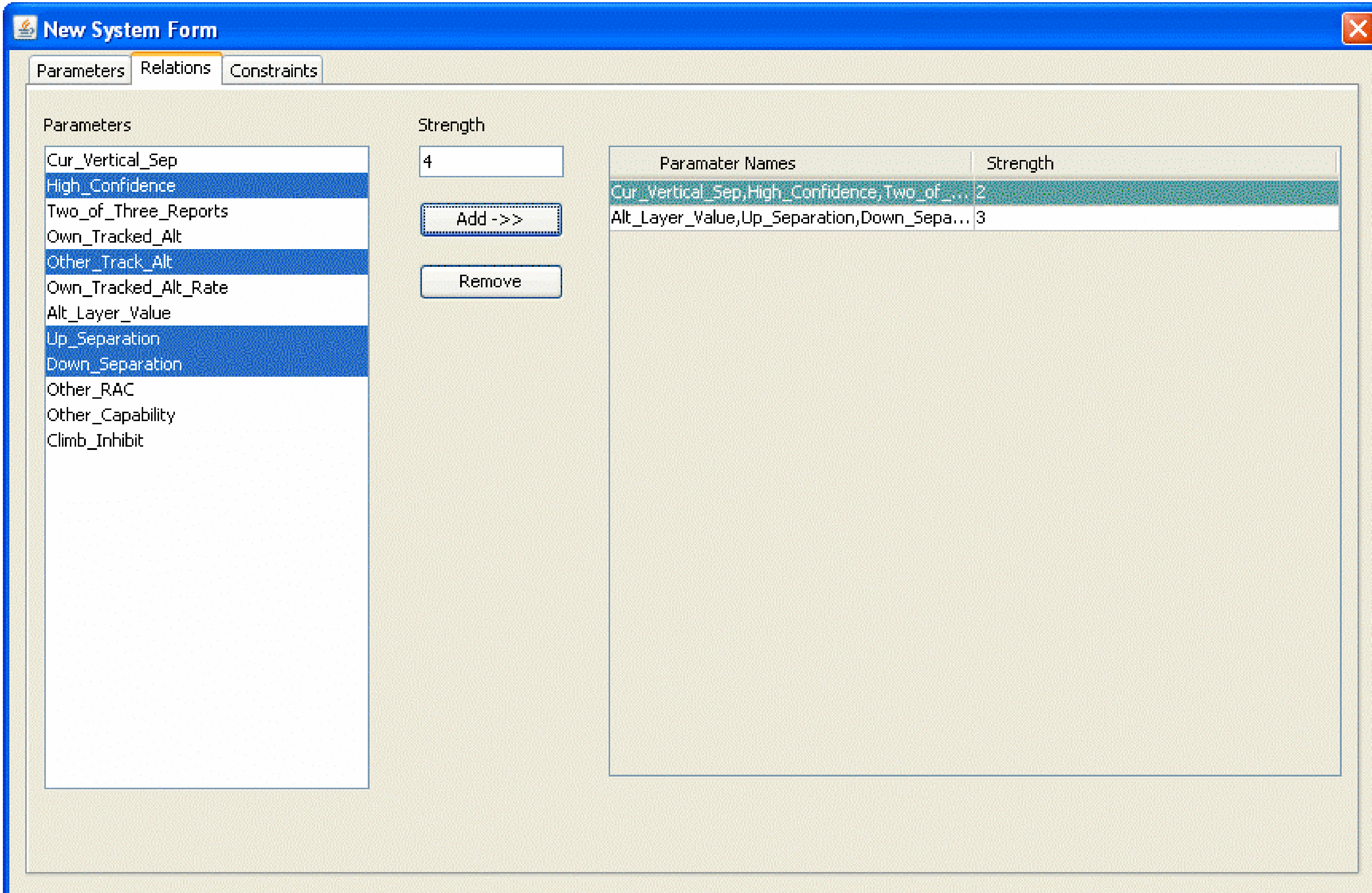
	CUR_V...	HIGH_...	TWO_...	OWN_...	OTHER...	OWN_...	ALT_L...	UP_SE...	DOWN...	OTHE...	OTHER...	CLIMB.
1	299	true	true	1	1	600	0	0	0	NO_INT...	TCAS_TA	true
2	300	false	false	2	2	601	1	0	399	DO_NO...	OTHER	false
3	601	true	false	1	2	600	2	0	400	DO_NO...	OTHER	true
4	299	false	true	2	1	601	3	0	499	DO_NO...	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

Defining a new system

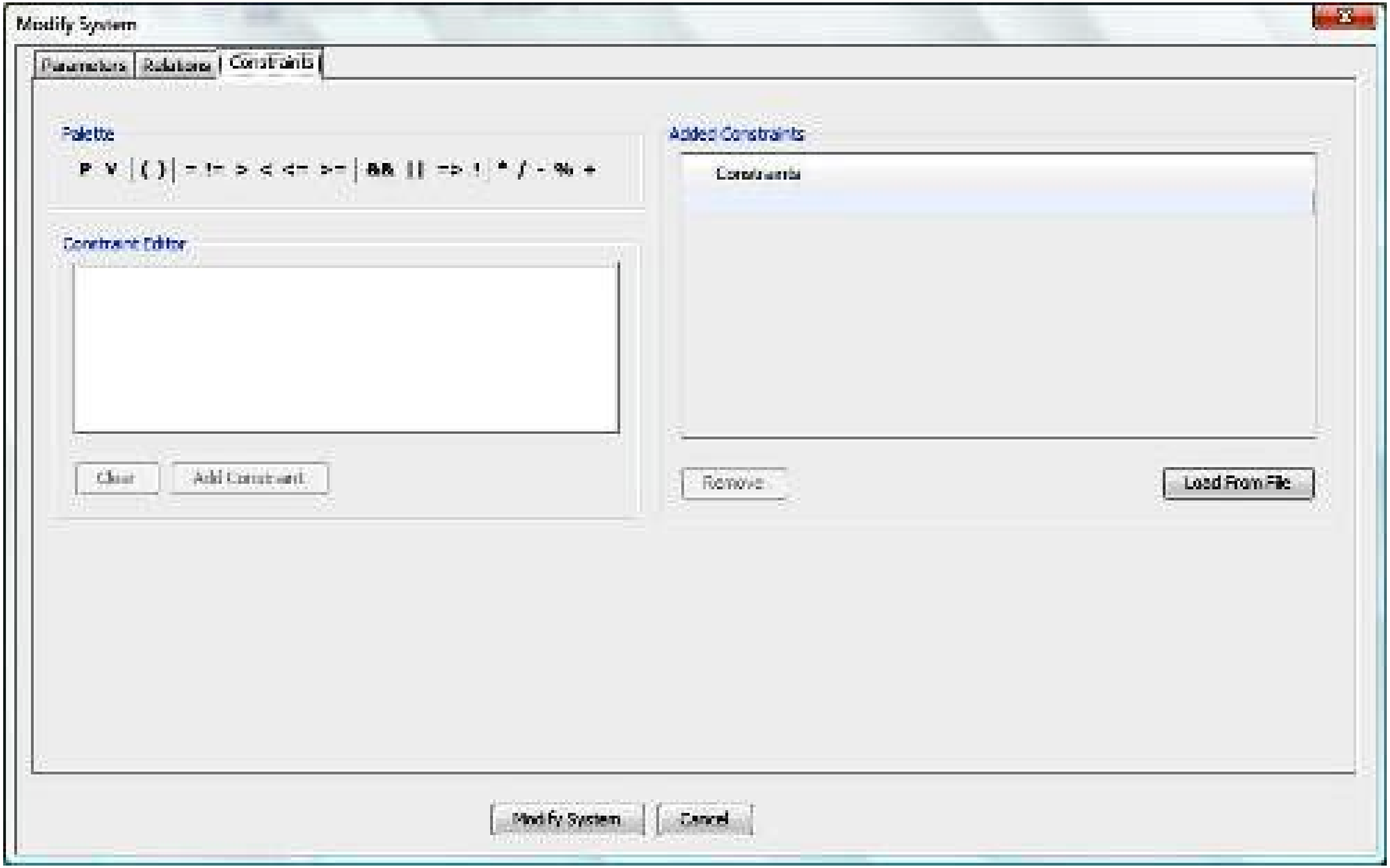
National Institute of Standards and Technology



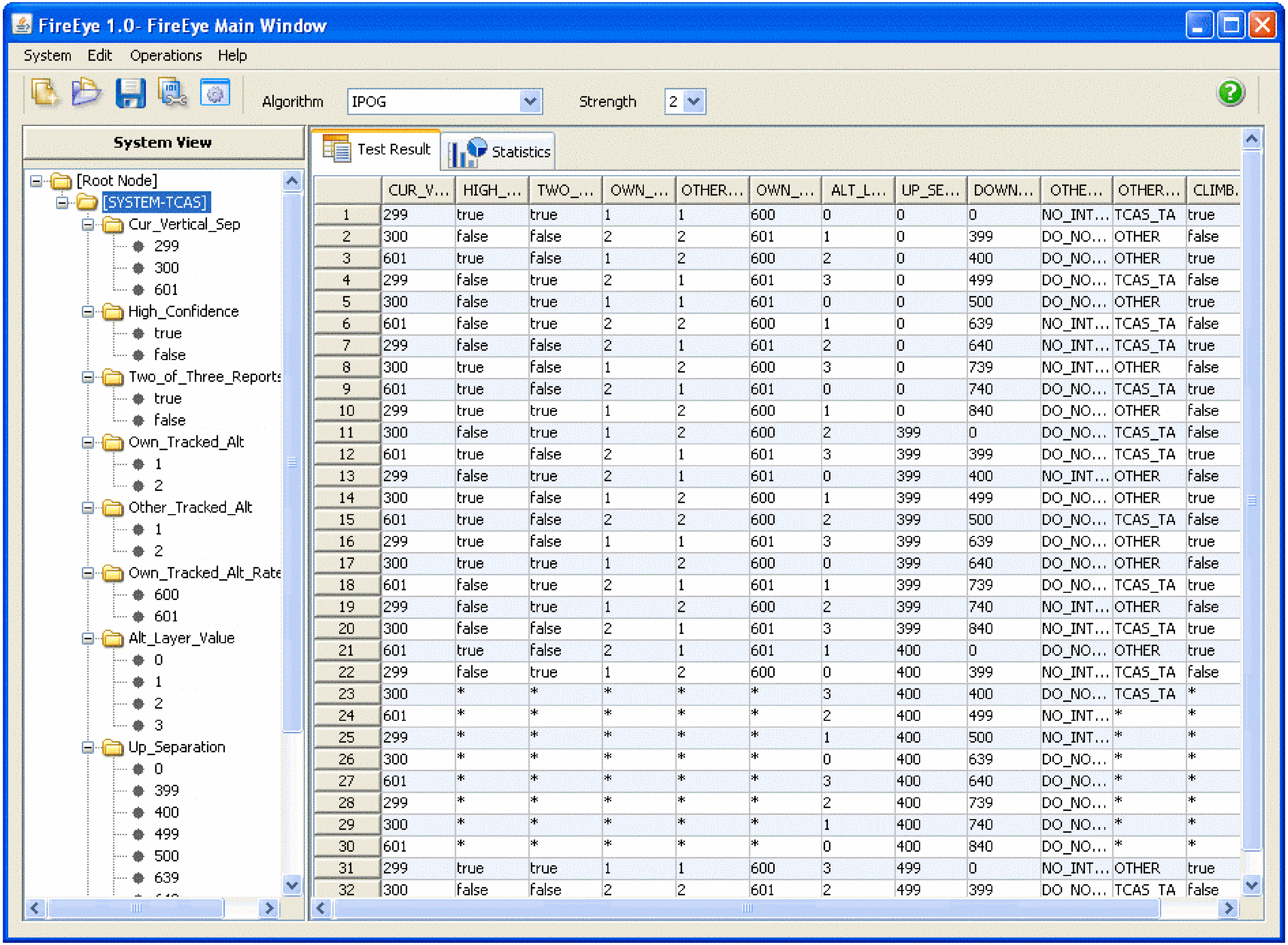
Variable interaction strength



Constraints



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

Output

Output formats:
- XML
- Numeric
- CSV
- Excel

Post-process output using Perl scripts, etc.

Output options

```
Degree of interaction
coverage: 2
Number of parameters: 12
Number of tests: 100

-----------------------------

0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 0 1 1 1 1
2 0 1 0 1 0 2 0 2 2 1 0
0 1 0 1 0 1 3 0 3 1 0 1
1 1 0 0 0 1 0 0 4 2 1 0
2 1 0 1 1 0 1 0 5 0 0 1
0 1 1 1 0 1 2 0 6 0 0 0
1 0 1 0 1 0 3 0 7 0 1 1
2 0 1 1 0 1 0 0 8 1 0 0
0 0 0 0 1 0 1 0 9 2 1 1
1 1 0 0 1 0 2 1 0 1 0 1
```

Etc.

```
Degree of interaction coverage: 2
Number of parameters: 12
Maximum number of values per
parameter: 10
Number of configurations: 100
-----------------------------------
Configuration #1:

1 = Cur_Vertical_Sep=299
2 = High_Confidence=true
3 = Two_of_Three_Reports=true
4 = Own_Tracked_Alt=1
5 = Other_Tracked_Alt=1
6 = Own_Tracked_Alt_Rate=600
7 = Alt_Layer_Value=0
8 = Up_Separation=0
9 = Down_Separation=0
10 = Other_RAC=NO_INTENT
11 = Other_Capability=TCAS_CA
12 = Climb_Inhibit=true
```

What if I want to try this?

- **Start small**
  - Apply pairwise or 3-way combinations to some modules
  - Compare tests developed with test sets for similar previous modules
  - Use combination coverage analysis to see how many t-way combinations covered by old test sets
  - Use existing test set but apply to combinations of input configurations
  - Add assertions to existing code

Summary

NIST
National Institute of Standards and Technology

- 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 available, 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

(Or just search “combinatorial testing”. We’re #1!)