Combinatorial Methods for Discrete Event Simulation of a Grid Computer Network

Rick Kuhn
Computer Security Division
National Institute of Standards and Technology
Gaithersburg, MD
kuhn@nist.gov

ModSim World, 14 Oct 09

Overview

- NIST is a US Government agency
- **The nation's measurement and testing laboratory** – 3,000 scientists, engineers, and support staff including 3 Nobel laureates
- Research in physics, chemistry, materials, manufacturing, computer science, including
  - network security
  - combinatorial methods and testing

Question: can combinatorial methods help us find attacks on networks?

Experiment: find deadlock configurations with grid computer network simulator. Compare:

- random simulation inputs
- covering arrays of 2-way, 3-way, 4-way combinations

Automated Combinatorial Testing

Goals – reduce testing cost, improve cost-benefit ratio

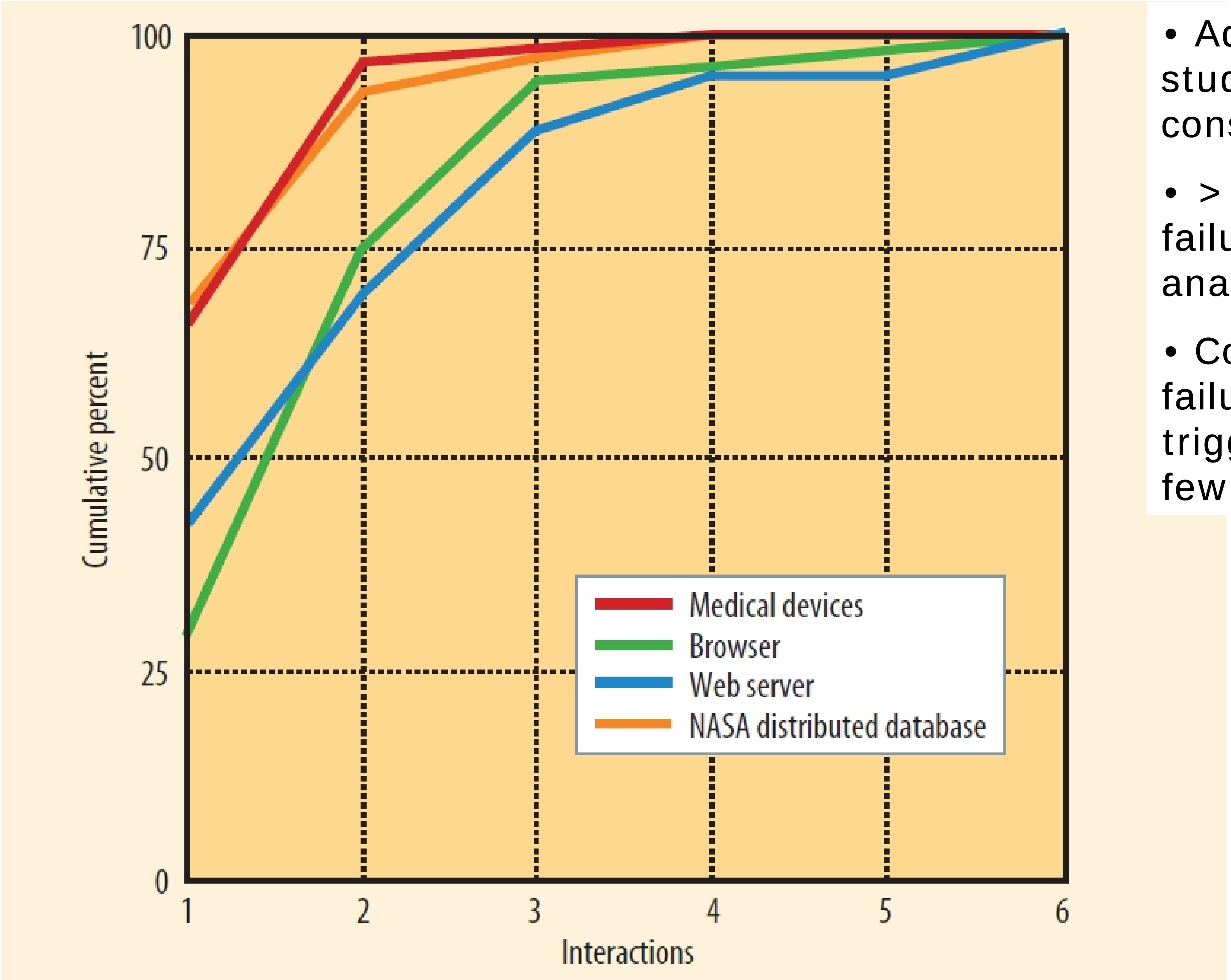
Accomplishments – huge increase in performance, scalability, 200+ users, most major IT firms and others

Also **non-testing applications** – modelling and simulation, genome

Software Failure Analysis

- NIST studied software failures in a variety of fields including 15 years of FDA medical device recall data
- What triggers software failures?
  - logic errors?
  - calculation errors?
  - inadequate input checking?
  - 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

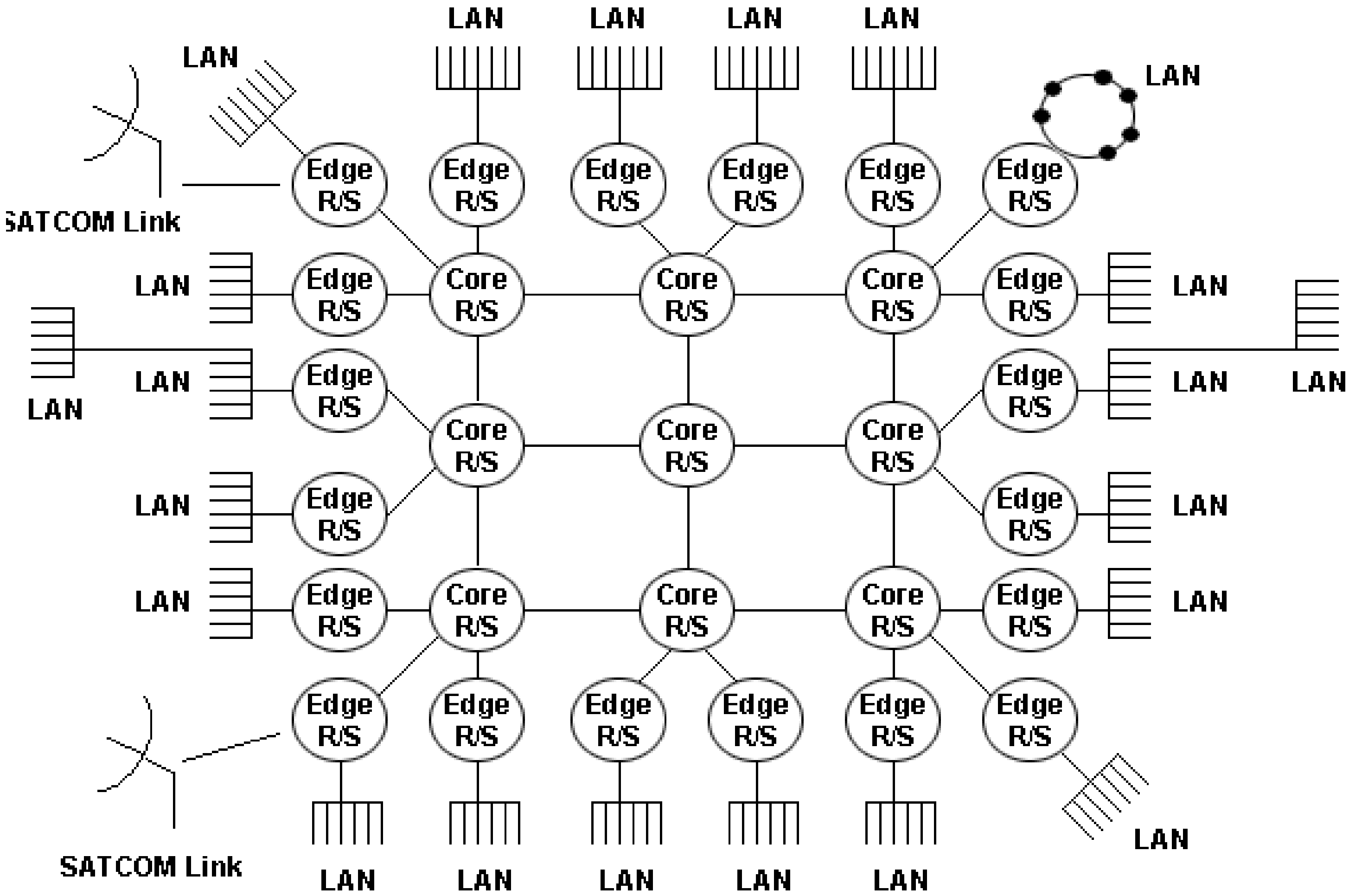
Failure-triggering Interactions



- Additional studies consistent
- > 4,000 failure reports analyzed
- Conclusion: failures triggered by few variables

How About Network Failure?

NIST National Institute of Standards and Technology



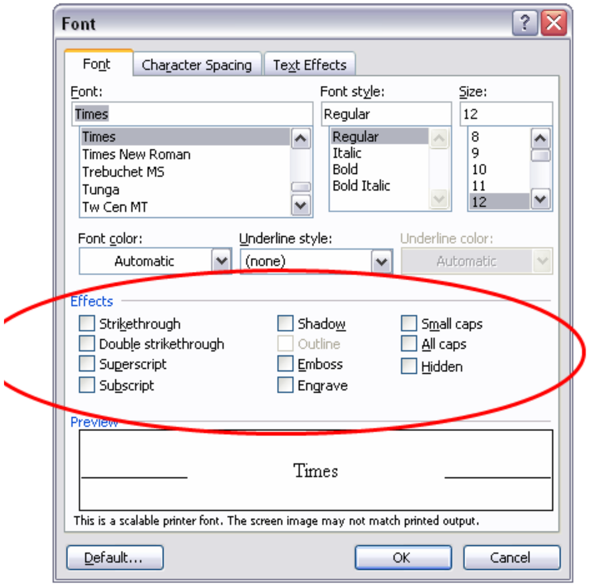
Can we use these ideas to induce network failure?

What we need: a Covering Array

Each row is a test:

Each column is a parameter:

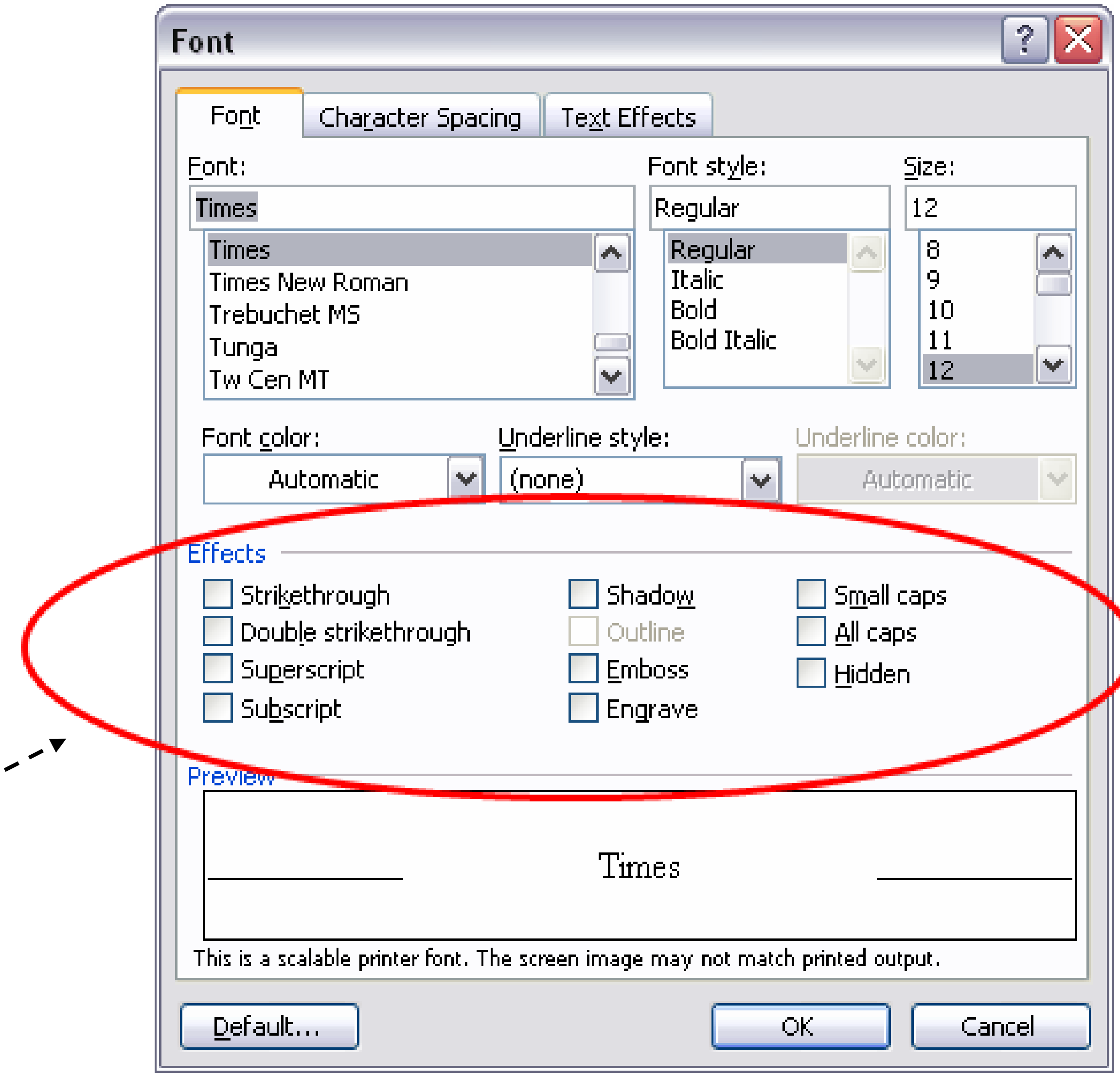
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	1
1	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0
1	1	0	1	0	0	1	0	1	0
0	0	0	1	1	1	0	0	1	1
0	0	1	1	0	0	1	0	0	1
0	1	0	1	1	0	0	1	0	0
1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	1	1	1	0	1



All triples in only 13 tests

National Institute of Standards and Technology

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

- 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

Modeling & Simulation Application

- "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)

Simulation Input Parameters

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

Combinatorial vs. Random

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 one that very few random tests could find:

$1/ 31{,}457{,}280 = 3.2 \times 10^{-8}$

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

Risks:

- accidental deadlock configuration: low
- deadlock configuration discovered by attacker: **high**

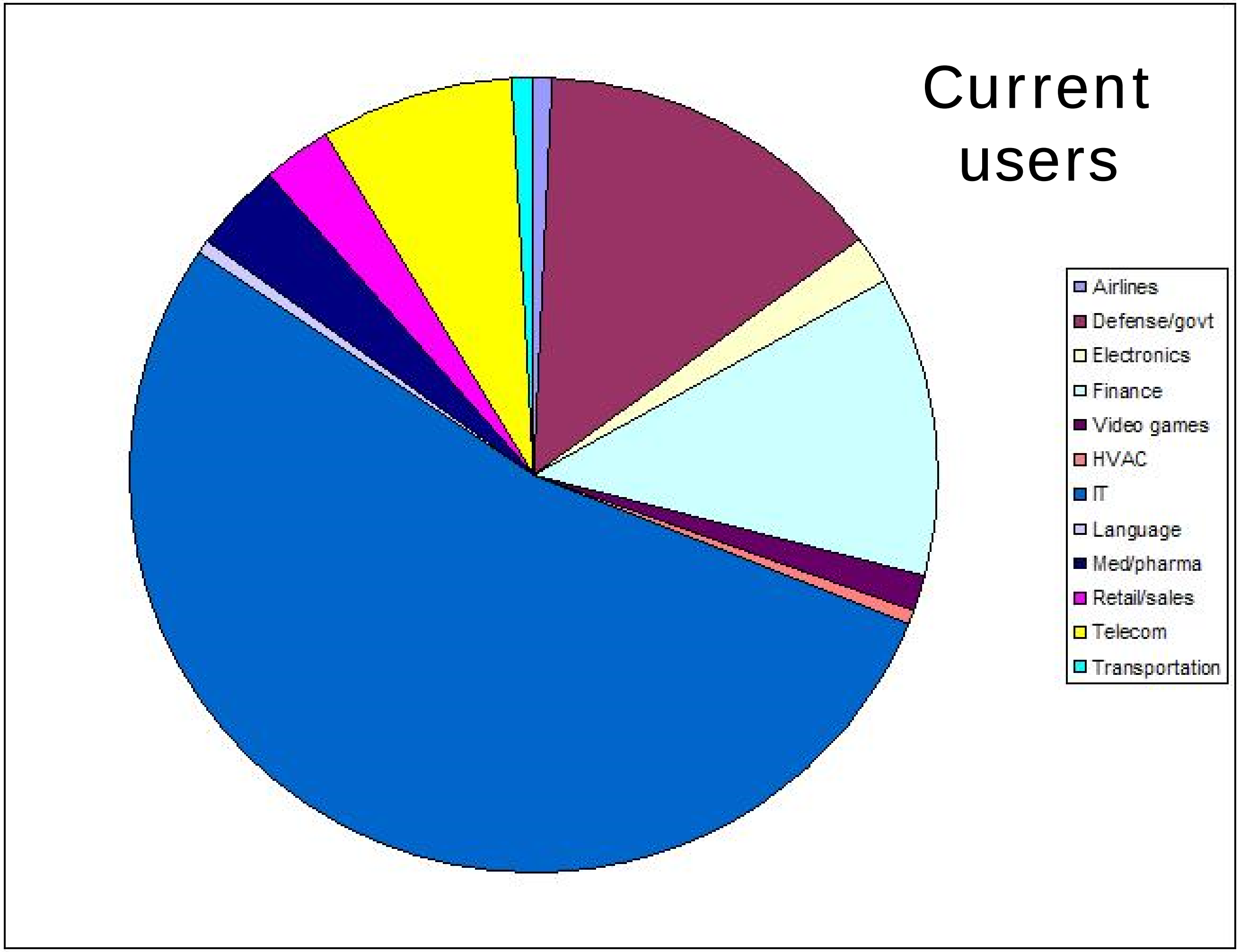
How many random tests do we need to equal combinatorial results?

Var	Vals/ var	2-way Tests		3-way Tests		4-way Tests	
		IPOG Tests	Ratio	IPOG Tests	Ratio	IPOG Tests	Ratio
10	2	10	1.80	20	3.05	42	3.57
10	4	30	4.83	151	6.05	657	3.43
10	6	66	5.80	532	3.73	3843	3.48
10	8	117	4.26	1214	4.46	12010	4.39
10	10	172	4.70	2367	4.94	29231	4.71
15	2	10	2.00	24	2.17	58	2.24
15	4	33	3.67	179	3.75	940	2.73
15	6	77	3.82	663	3.79	5243	3.26
15	8	125	4.41	1551	4.36	16554	3.66
15	10	199	4.72	3000	5.08	40233	3.97
20	2	12	1.92	27	2.59	66	2.12
20	4	37	3.78	209	2.98	1126	3.35
20	6	86	3.35	757	3.39	6291	2.99
20	8	142	4.44	1785	4.73	19882	3.00
20	10	215	4.78	3463	4.04	48374	3.25
25	2	12	2.83	30	2.33	74	2.35
25	4	39	3.08	233	3.39	1320	2.67
25	6	89	3.67	839	3.44	7126	2.75
25	8	148	5.71	1971	3.76	22529	2.72
25	10	229	4.50	3823	4.32	54856	3.50
Ratio Avg.			3.90		3.82		3.21

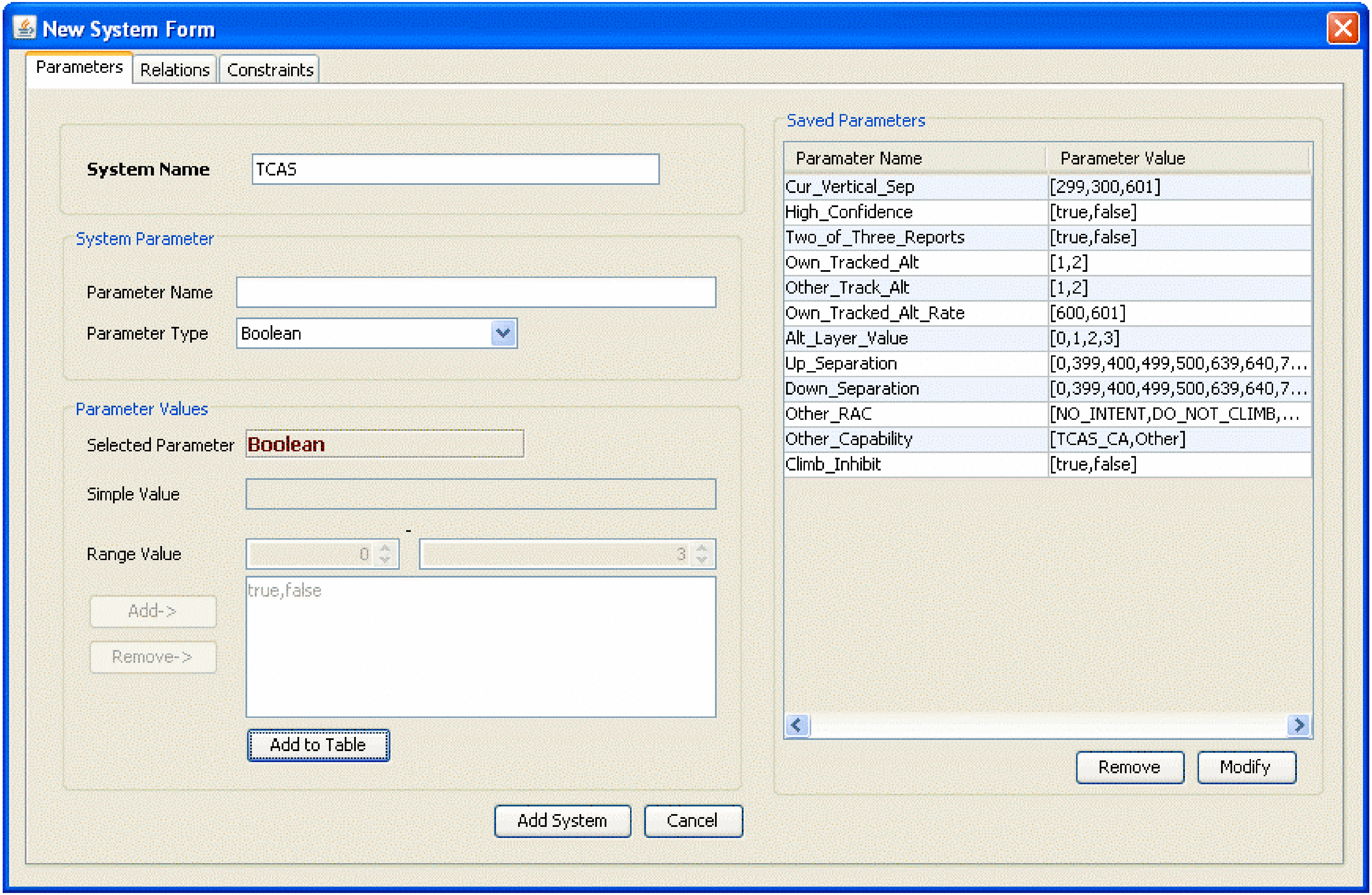
Answer: **3x** to **4x** as many
and still would not guarantee detection

Tools

- Covering array generator
- Coverage analysis - what is the combinatorial coverage of existing test set?
- .Net configuration file generator
- Fault location - currently underway

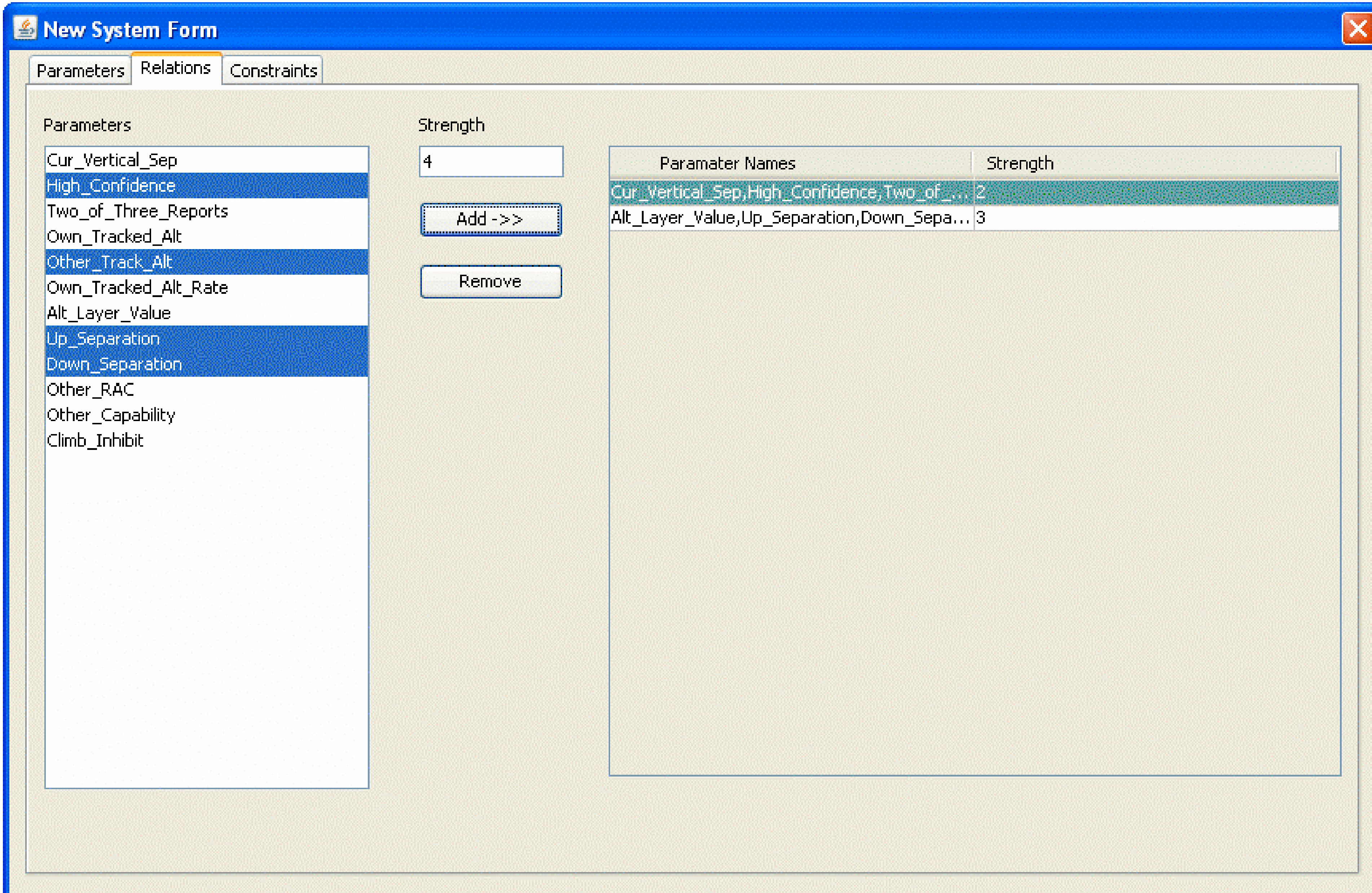


Defining a new system

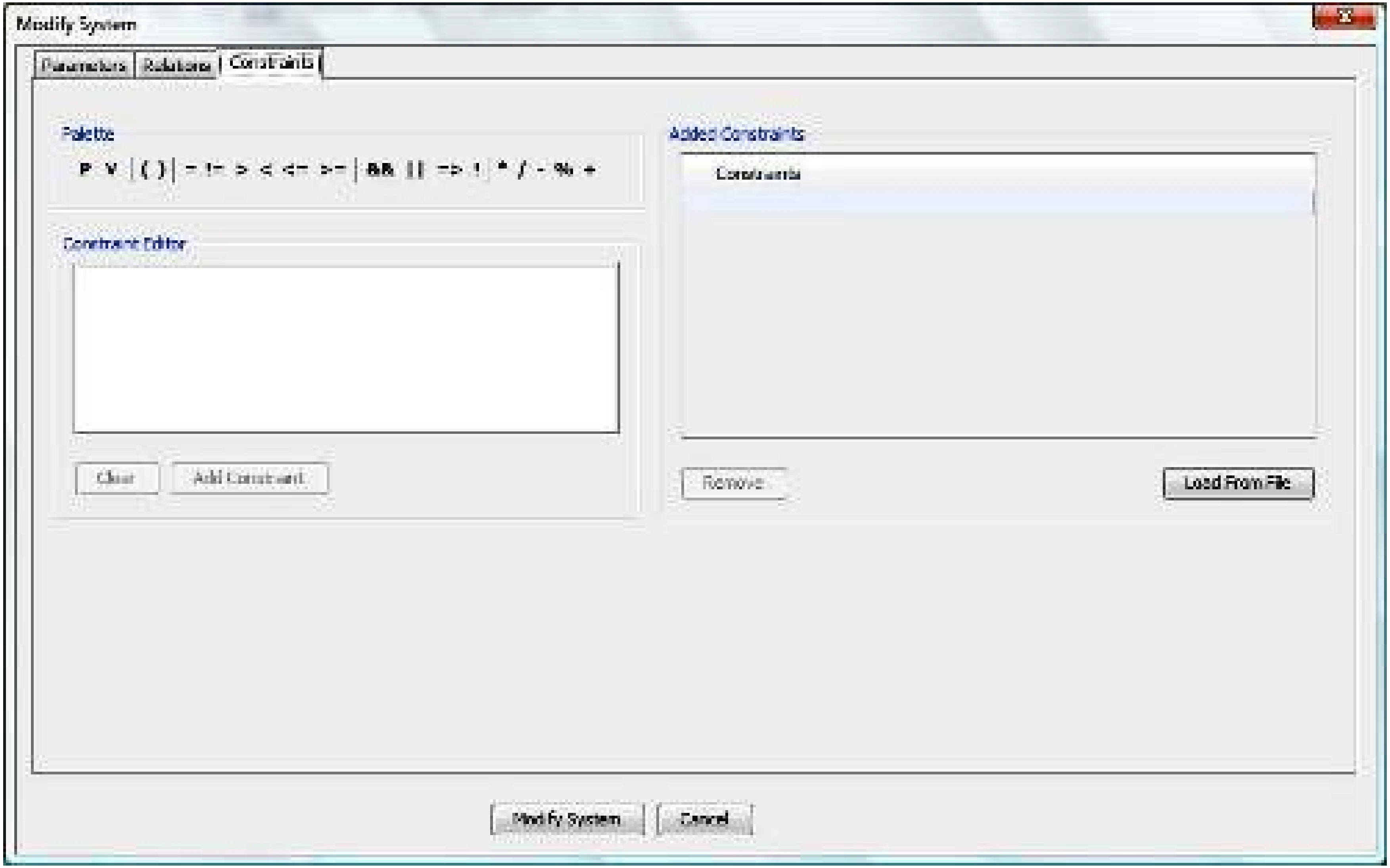


Variable interaction strength

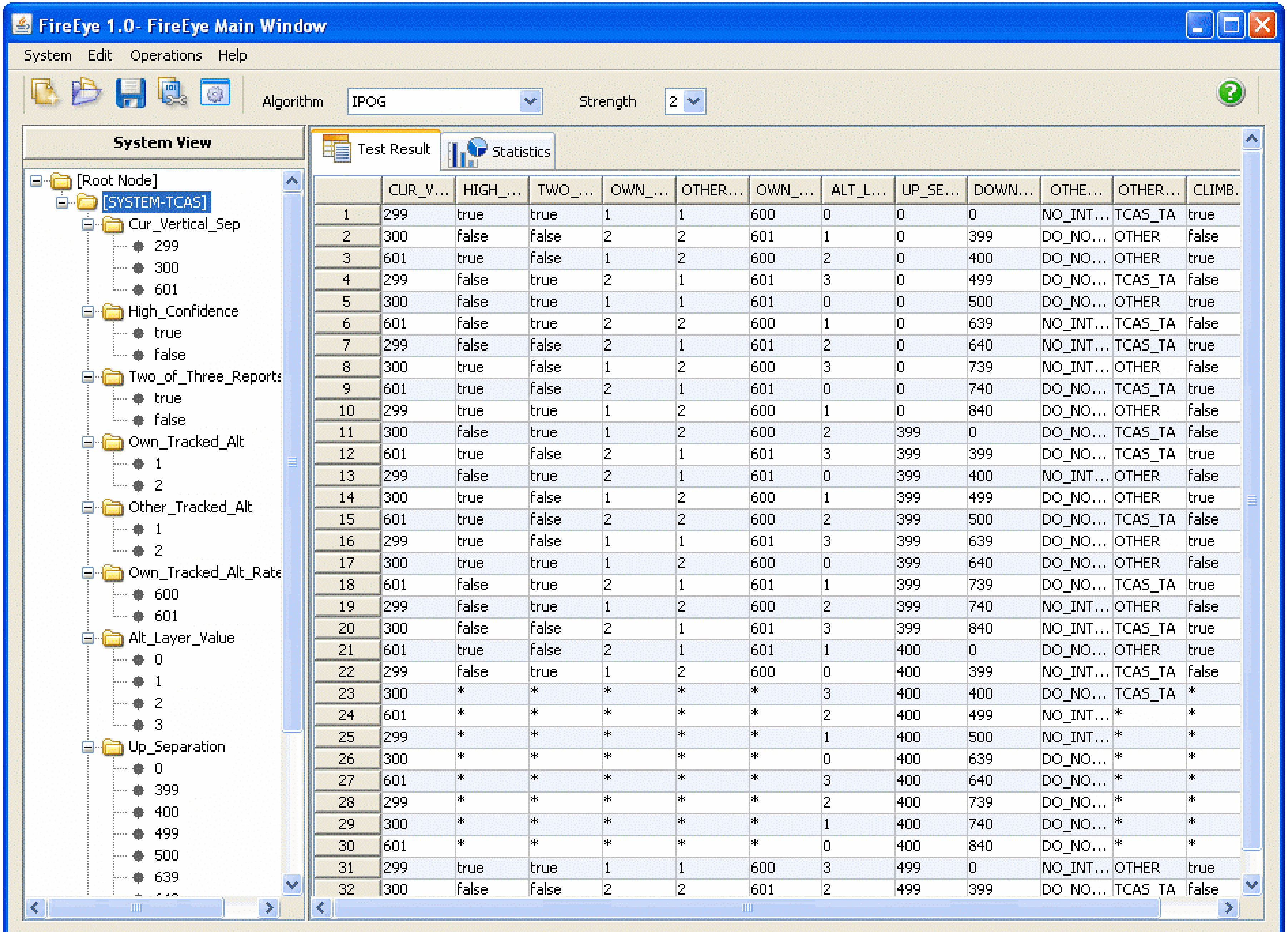
NIST
National Institute of Standards and Technology



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

Summary

NIST
National Institute of Standards and Technology

- Empirical research suggests that all or nearly 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
- Beta release of tools available, to be open source

Please contact us if you are interested!

Rick Kuhn — kuhn@nist.gov

Raghu Kacker — raghu.kacker@nist.gov

http://csrc.nist.gov/acts (Or just search “combinatorial testing” !)