Combinatorial Methods for Event Sequence Testing

D. Richard Kuhn[1], James M. Higdon[2],
James F. Lawrence[1,3], Raghu N. Kacker[1], Yu Lei[4]

[1]National Institute of Standards & Technology
Gaithersburg, MD

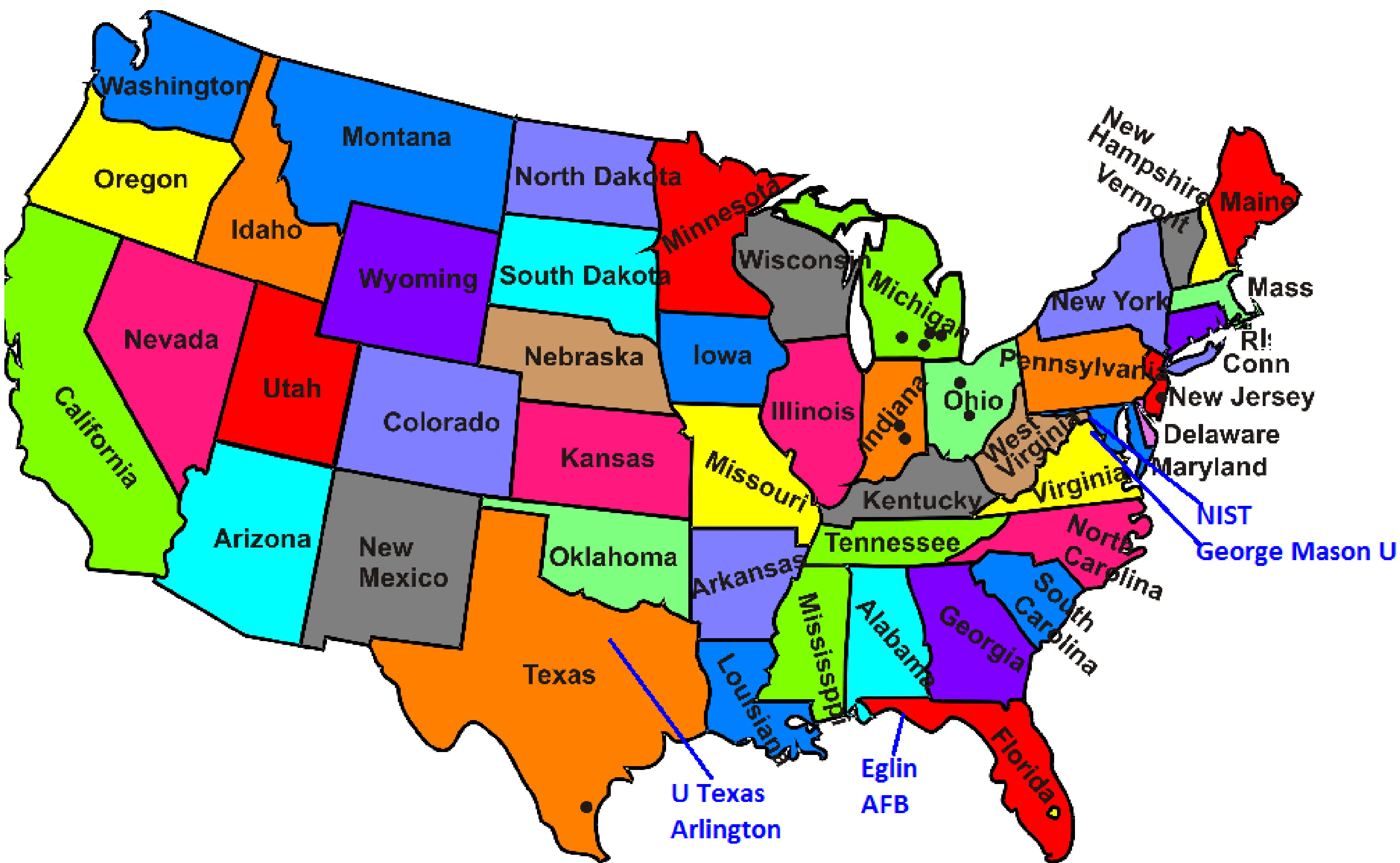
[2]US Air Force
Jacobs Technology, TEAS contract,
46th Test Squadron,
Eglin AFB, FL

[3]Dept. of Mathematics
George Mason Univ.
Fairfax, VA

[4]Dept. of Computer Science
University of Texas Arlington, TX

What is NIST and why are we doing this project?

- US Government agency, whose mission is to support US industry through developing better measurement and test methods
- 3,000 scientists, engineers, and staff including 3 Nobel laureates



Why: USAF laptop app testing

Problem: connect many peripherals, order of connection may affect application

Combinatorial Sequence Testing

- Suppose we want to see if a system works correctly regardless of the order of events. How can this be done efficiently?
- Failure reports often say something like: 'failure occurred when A started if B is not already connected'.
- Can we produce compact tests such that all t-way sequences covered (possibly with interleaving events)?

Event	Description
a	connect autonomous vehicle
b	connect autonomous aircraft 1
c	connect satellite link
d	connect router
e	connect autonomous aircraft 2
f	connect range finder

Sequence Covering Array

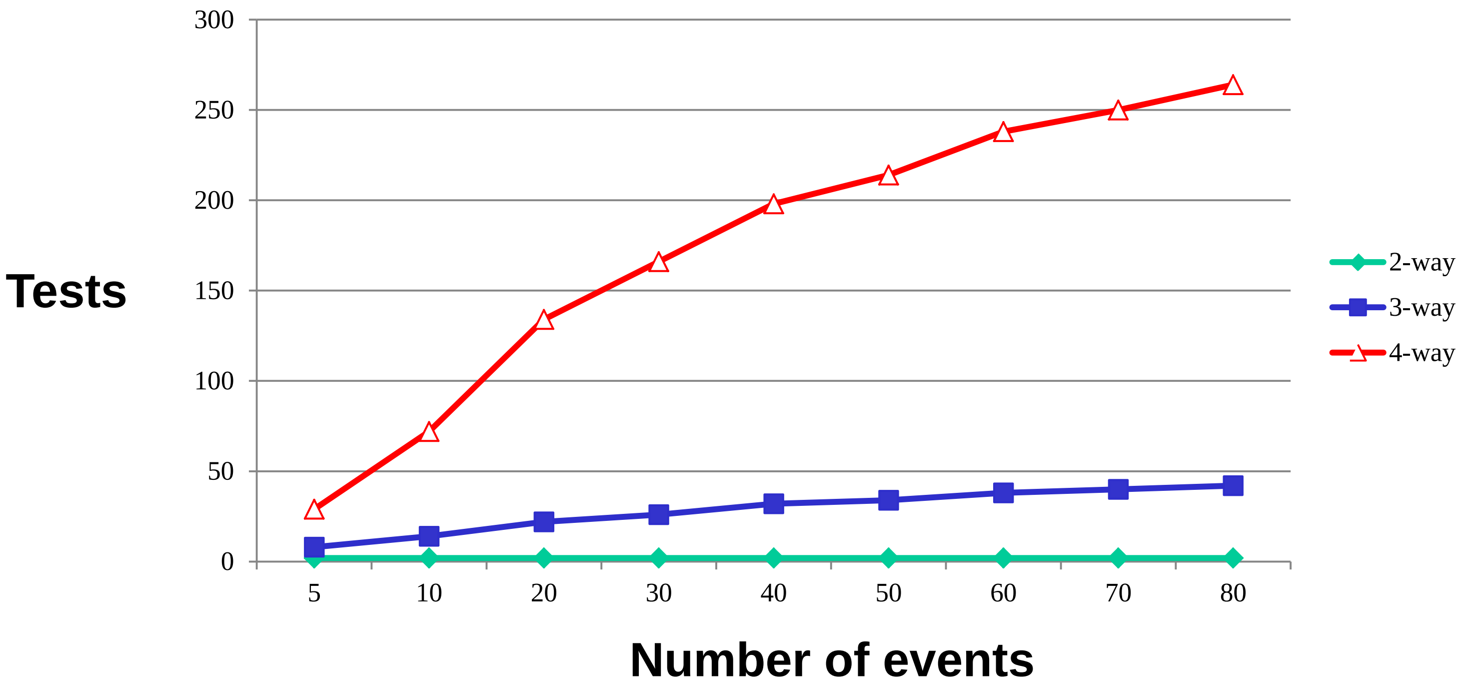
- With 6 events, all sequences = 6! = 720 tests
- Only 10 tests needed for all 3-way sequences, results <u>even better for larger numbers of events</u>
- Example: .*c.*f.*b.* covered. Any such 3-way seq covered.

Test	Sequence					
1	a	b	c	d	e	f
2	f	e	d	c	b	a
3	d	e	f	a	b	c
4	c	b	a	f	e	d
5	b	f	a	d	c	e
6	e	c	d	a	f	b
7	a	e	f	c	b	d
8	d	b	c	f	e	a
9	c	e	a	d	b	f
10	f	b	d	a	e	c

NIST National Institute of Standards and Technology

Sequence Covering Array Properties

- 2-way sequences require only 2 tests
  (write events in any order, then reverse)
- For > 2-way, number of tests grows with log *n*, for *n* events
- Simple greedy algorithm produces compact test set
- Not previously described in CS or math literature



Constructing Sequence Covering Arrays

- Conventional covering array algorithm could be used if range of each variable = n for n variables, and constraints prevent use of each value more than once, thus not efficient
- Direct construction also possible, starting from two tests for t=2 and creating a new test for each variable vi of n, w/ vi followed by array for remaining v-1 variables
- Sequence extension is another alternative: for initial array of m events, m<n, check if each t-way sequence covered; if not extend a test w/ up to t events
- Greedy algorithm is fast, simple, and produces good results
- Naïve greedy algorithm improved with a simple reversal of each generated test, giving "two for the price of one"
- Some newer algorithms produce smaller array at t=3, but problematic at t=4 and above

Greedy Algorithm

- Standard greedy approach, with an optimization step
- Allows exclusion of specified sequences

if (constraint on sequence *x..y*) symmetry = false; **else** symmetry = true;
while (all *t-way sequence*s not marked in *chk*) {
tc := set of *N* test candidates with random values of each of the *n* parameters
$test_1$:= test *T* from set *tc* such that
T covers the greatest number of sequences not marked as covered in *chk*
&& .*x.*y.* not matched in *T*
for each new sequence covered in $test_1$, set bit in set *chk* to 1;
ts := *ts* U $test_1$;
if (symmetry && all *t-way sequence*s not marked in *chk*) {
    $test_2$:= reverse($test_1$);
    ts := *ts* U $test_2$;
   for each new sequence cover in $test_2$, set bit in set *chk* to 1; }
  }

Algorithm analysis

- Time $O(n^t)$
- Storage $O(n^t)$
- Practical to produce tests for up to 100 events in seconds to minutes on standard desktop
- Interesting properties:
- Reversal step produces = number of previously uncovered sequences as test being reversed
- Number of tests grows with log n
- Where K(n,t) = fewest tests for t-way seq of n events
  - K(n,t) >= t!
  - K(n,3) >= CAN(n-1,2) i.e., a 3-way SCA for n events at least as large as 2-way array for n-1 symbols (Jim L)

Using Sequence Covering Arrays

- Laptop application with multiple input and output peripherals
- Seven steps plus boot: open app, run scan, connect peripherals P1 – P5
- Operation requires cooperation among peripherals
- About 7,000 possible valid sequences
- Testing requires manual, physical connection of devices
- Originally tested using Latin Squares approach:
  - Each event appears once
  - Each event at every possible location in sequence
  - OK for some configurations, but produces too many tests

Application to Test Problem

- Tested system using 7-event sequence covering array, 3-way sequences, 17 tests

Original Test	Test	Step1	Step2	Step3	Step4	Step5	Step6	Step7	Step8
1	1	Boot	P-1 (USB-RIGHT)	P-2 (USB-BACK)	P-3 (USB-LEFT)	P-4	P-5	Application	Scan
2	2	Boot	Application	Scan	P-5	P-4	P-3 (USB-RIGHT)	P-2 (USB-BACK)	P-1 (USB-LEFT)
3	3	Boot	P-3 (USB-RIGHT)	P-2 (USB-LEFT)	P-1 (USB-BACK)	Application	Scan	P-5	P-4
4	4	Boot	P-4	P-5	Application	Scan	P-1 (USB-RIGHT)	P-2 (USB-LEFT)	P-3 (USB-BACK)
5	5	Boot	P-5	P-2 (USB-RIGHT)	Application	P-1 (USB-BACK)	P-4	P-3 (USB-LEFT)	Scan
6A	6	Boot	Application	P-3 (USB-BACK)	P-4	P-1 (USB-LEFT)	Scan	P-2 (USB-RIGHT)	P-5
6B	7	Boot	Application	Scan	P-3 (USB-LEFT)	P-4	P-1 (USB-RIGHT)	P-2 (USB-BACK)	P-5
6C	8	Boot	P-3 (USB-RIGHT)	P-4	P-1 (USB-LEFT)	Application	Scan	P-2 (USB-BACK)	P-5
6D	9	Boot	P-3 (USB-RIGHT)	Application	P-4	Scan	P-1 (USB-BACK)	P-2 (USB-LEFT)	P-5
7	10	Boot	P-1 (USB-RIGHT)	Application	P-5	Scan	P-3 (USB-BACK)	P-2 (USB-LEFT)	P-4
8A	11	Boot	P-4	P-2 (USB-RIGHT)	P-3 (USB-LEFT)	Application	Scan	P-5	P-1 (USB-BACK)
8B	12	Boot	P-4	P-2 (USB-RIGHT)	P-3 (USB-BACK)	P-5	Application	Scan	P-1 (USB-LEFT)
9	13	Boot	Application	P-3 (USB-LEFT)	Scan	P-1 (USB-RIGHT)	P-4	P-5	P-2 (USB-BACK)
10A	14	Boot	P-2 (USB-BACK)	P-5	P-4	P-1 (USB-LEFT)	P-3 (USB-RIGHT)	Application	Scan
10B	15	Boot	P-2 (USB-LEFT)	P-5	P-4	P-1 (USB-BACK)	Application	Scan	P-3 (USB-RIGHT)
11	16	Boot	P-3 (USB-BACK)	P-1 (USB-RIGHT)	P-4	P-5	Application	P-2 (USB-LEFT)	Scan
12A	17	Boot	Application	Scan	P-2 (USB-RIGHT)	P-5	P-4	P-1 (USB-BACK)	P-3 (USB-LEFT)
12B	18	Boot	P-2 (USB-RIGHT)	Application	Scan	P-5	P-4	P-1 (USB-LEFT)	P-3 (USB-BACK)
NA	19	P-5	P-4	P-3 (USB-LEFT)	P-2 (USB-RIGHT)	P-1 (USB-BACK)	Boot	Application	Scan

Results

- Manually configured tests to deal with constraints
- Found errors that would not have been detected with 2-way sequences, and unlikely to have been found with scenario-based testing
- Added the ability to incorporate constraints, based on experience with this test project

Summary

- Sequence covering arrays developed to address a need in practical testing
- Useful for testing order of events in sequential systems
- Applicable to GUI testing
- Anticipate applicability to concurrent systems
- Improves test effectiveness
- Reduces cost with fewer tests
- Tool now incorporates constraints

Events	3-seq Tests	4-seq Tests
5	8	26
6	10	36
7	12	46
8	12	50
9	14	58
10	14	66
11	14	70
12	16	78
13	16	86
14	16	90
15	18	96
16	18	100
17	20	108
18	20	112
19	22	114
20	22	120
21	22	126
22	22	128
23	24	134
24	24	136
25	24	140
26	24	142
27	26	148
28	26	150
29	26	154
30	26	156
40	32	182
50	34	204
60	38	222
70	40	238
80	42	250

Number of tests, 3-way and 4-way