Advanced combinatorial testing of software and systems using covering arrays

- o Advanced
  - o High strength *t*-way testing
  - o Support complex constraints
- o Made possible by use of covering arrays

Raghu Kacker
Applied and Computational Division
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg Maryland 20899 USA

Team: Rick Kuhn, Yu Lei, Jim Lawrence, Vincent Hu

National Institute of Standards and Technology http://www.nist.gov

- US federal research laboratory founded in 1901
- About 3000 staff including 3 Nobel laureates
- Laboratory Programs
  - Materials Measurement Laboratory
  - Physical Measurement Laboratory
  - Engineering Laboratory
  - Information Technology Laboratory
  - Center for Nano-scale Science and Technology
  - Center for Neutron Research
- Innovation & Industry Services
  - Baldrige Performance Excellence Program
  - Hollings Manufacturing Extension Partnership
  - Technology Innovation Program

Outline

- Discuss development of Combinatorial Testing (CT) as adaptation of Design of Experiments (DoE) methods
- Special aspects of CT for software and systems
- Limitations of Orthogonal Arrays (OAs), benefits of Covering Arrays (CAs) for generating combinatorial test suites for testing software and systems

Combinatorial testing is a variation of Design of Experiments (DoE) adapted for testing software

- Example of DoE: Five test factors
  - Viscosity {a} with 2 values {0, 1}
  - Feed rate {b} with 2 values {0, 1}
  - Spin Speed {c} with 2 values {0, 1}
  - Pressure {d} with 2 values {0, 1}
  - Materials {e} with 4 types {0, 1, 2, 3}
- Combinatorial test structure $2^4 x 4^1$
  - Number of possible test cases: $2^4 x 4^1 = 64$
- Object: evaluate only "main effects" of five factors
- Possible to evaluate main effects from 8 test cases only determined using orthogonal array OA(8, $2^4 \times 4^1$, 2)

DoE based on orthogonal array: OA(8, $2^4 \times 4^1$, 2)

Strength 2: every two columns contain all pairs exactly once or exactly twice

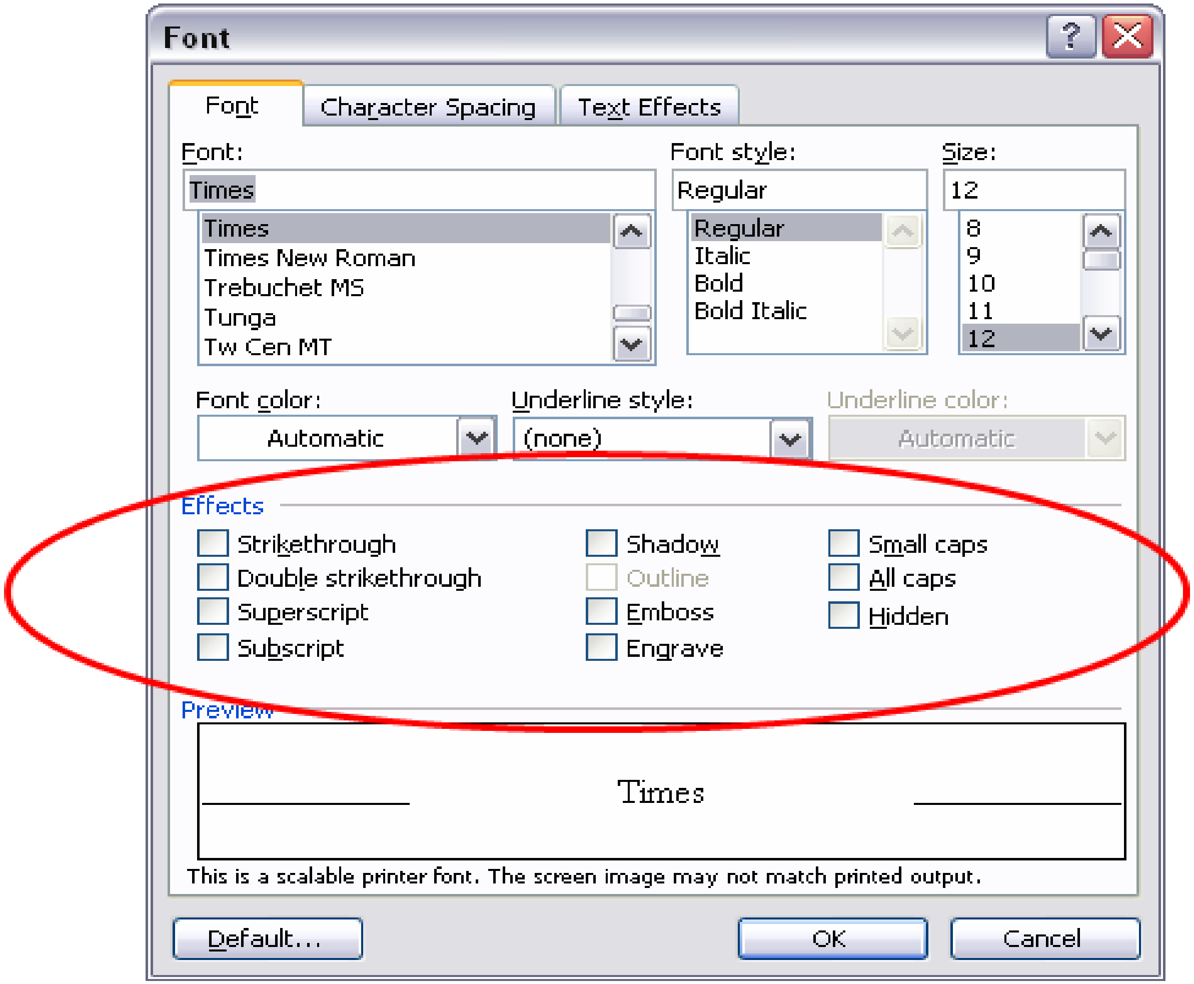
	a	b	c	d	e	data
1.	0	0	0	0	0	y_1
2.	1	1	1	1	0	y_2
3.	0	0	1	1	1	y_3
4.	1	1	0	0	1	y_4
5.	0	1	0	1	2	y_5
6.	1	0	1	0	2	y_6
7.	0	1	1	0	3	y_7
8.	1	0	0	1	3	y_8

- Associate factors with columns, test values {0, 1}, {0, 1, 2, 3} with entries
- Rows of OA specify 8 test cases
- Every test value paired with each value of every other factor
- Main effect of factor a: $(y_2+y_4+y_6+y_8)/4 - (y_1+y_3+y_5+y_7)/4$
- All test values of every other factor represented in each average of four

DoE balanced, software test suite need not be

- DoE plans can be expressed in matrix form
  - Columns: test factors, Entries: test values, Rows: tests cases
- In DoE “main effects” and “interaction effects” linear contrasts of response data
  - Binary factors: difference of two averages of half data
  - Main effect of factor a: $(y_2+y_4+y_6+y_8)/4 - (y_1+y_3+y_5+y_7)/4$
  - For main effects to be meaningful, DoE must be balanced
- In testing software and systems “interaction” means “joint combinatorial effect of two or more factors”
- CT suite for testing software need not be balanced because DoE type “main effects” not relevant, statistical models not used in data analysis

Example: Font effects on word processing



Factors values and test cases

- Each factors (font effects) can be turned on or off
  - Ten binary test factors with test values {0, 1}
- Combinatorial test structure 2^{10}
- Possible test cases 2^{10} = 1024 too many to test
- Suppose no failure involves more than 3 factors jointly
  - Sufficient to test all triplets of factor values
- Number of triplets = $\binom{10}{3}2^3$ = 960
- How many test cases needed to test all 960 triples?
- How to determine those test cases?

All 960 triples can be covered by13 test cases determined using covering array CA(13, 2^{10}, 3)

	Factors									
Rows	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	0	1	0	0	0	0	1
4	1	0	1	1	0	1	0	1	0	0
5	1	0	0	0	1	1	1	0	0	0
6	0	1	1	0	0	1	0	0	1	0
7	0	0	1	0	1	0	1	1	1	0
8	1	1	0	1	0	0	1	0	1	0
9	0	0	0	1	1	1	0	0	1	1
10	0	0	1	1	0	0	1	0	0	1
11	0	1	0	1	1	0	0	1	0	0
12	1	0	0	0	0	0	0	1	1	1
13	0	1	0	0	0	1	1	1	0	1

Early history of combinatorial testing for software and systems

- Mandl (1985) “Use of orthogonal Latin squares for testing Ada compiler” often cited first publication
  - Special case of orthogonal arrays
- Japan/mid-1980s OAs used for testing hardware-software systems: Tatsumi (1987), Tatsumi et al (1987)
- USA/late-1980s descendent orgs of AT&T (Bell Labs, Bellcore-Telcordia) exploring use of OAs for combinatorial testing; developing tools based on OAs: Brownlie et al (1992), Burroughs et al (1994)
- In1990s use of OAs for testing of computer and communication hardware-software systems expanded

Tools for generating combinatorial test suites

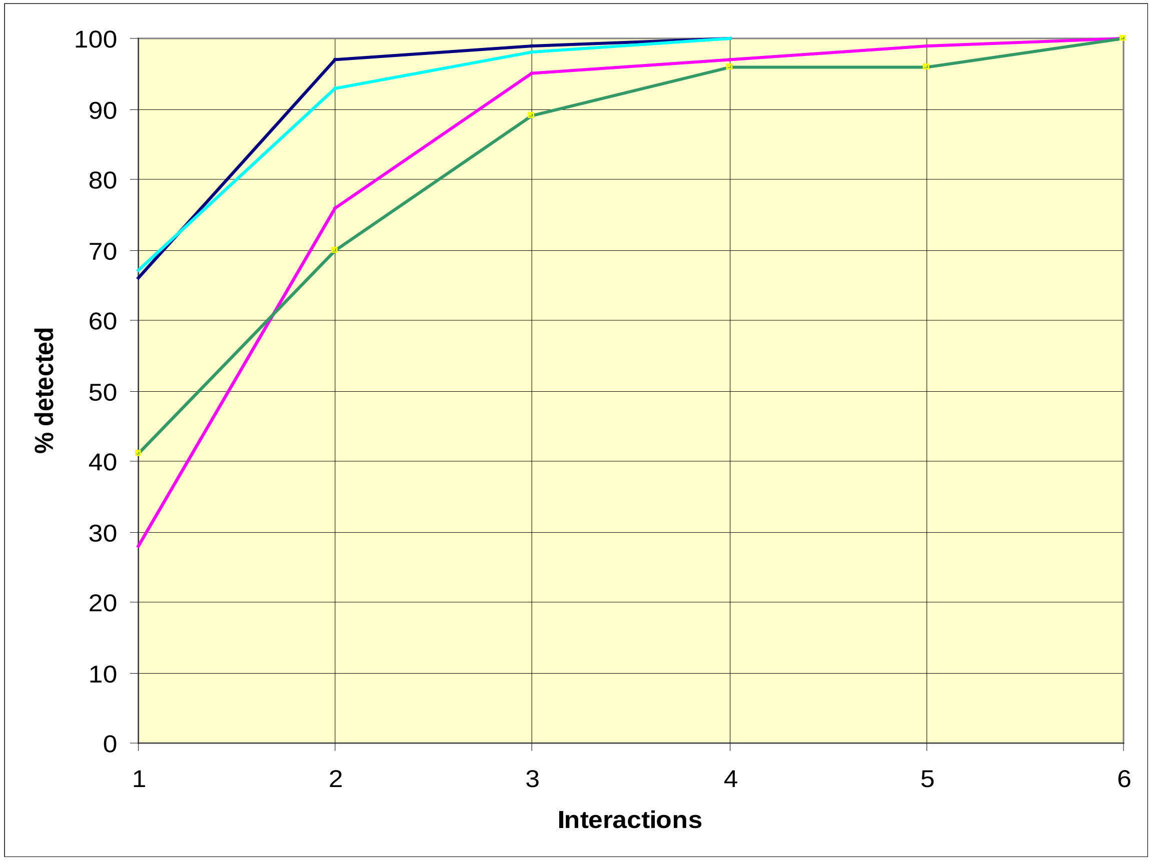
- Early tools for generating test suites for pairwise testing
  - OATS (Phadke AT&T) 1990s (not public)
  - CATS (Sherwood AT&T) 1990s (not public)
  - AETG (Cohen et al Telcordia) 1997 (commercial)
  - IPO (Yu Lei NCSU) 1998 (not public)
- Czerwonka (Microsoft) lists 34 tools (www.pairwise.org)

- Tconfig	- CTS	- Jenny
- TestCover	- DDA	- AllPairs
- AllPairs[McDowell]	- PICT	- EXACT
- IPO-s		

- ACTS (NIST/UTA): freely distributed
  - Primary algorithm: IPOG generalization of 1998 IPO (Yu Lei UTA)

NIST investigated actual faults to determine what kind of testing would have detected them

Medical devices recall data, Browser, Server, NASA distributed database, Network security system



Pairwise testing may not be adequate

- Kuhn et al (2001, 2002, 2004)
  - 2-way testing could detect 65 % to 97 % faults
  - 3-way testing could detect 89 % to 99 % faults
  - 4-way testing could detect 96 % to 100 % faults
  - 5-way testing could detect 96 % to 100 % faults
  - 6-way testing could detect 100 % faults in all cases investigated
- Kera Bell (2006, NCSU) arrived similar conclusion
- Empirical conclusion: pairwise (2-way) testing useful but may not be adequate; 6-way testing may be adequate

Combinatorial high strength (*t*-way) testing

- Dynamic verification of input-output system
  - against its known expected behavior
  - on test suite of test cases selected such that
  - all *t*-way combinations are exercised with the
  - object of discovering faults in system
- Earlier combinatorial test suites based on orthogonal arrays of strength 2 useful for pairwise (2-way) testing
- Now tools available for high strength *t*-way testing
  - ACTS (NIST/UTA) 2009
  - Primary algorithm is IPOG, generalization of IPO for t > 2
  - ACTS has built-in support of constraints
  - http://csrc.nist.gov/groups/SNS/acts/index.html
  - Freely downloaded by over 800 organizations and individuals

Special aspects of CT for software and systems-1

- System Under Test (SUT) must be exercised (dynamic verification)
- CT does not require access to source code
- Expected behavior (oracle) for each test case be known
  - determined from functionality and/or other information
- In CT actual behavior is compared against expected for each test case with final result of pass or fail
- Objective of CT to reveal faults; a failure indicates fault, a fault always results in failure
- Repeat of a *t*-way combination gives same result so no need to repeat *t*-way combinations in test suite

Special aspects of CT for software and systems-2

- Numbers of test values of factors may be different
- A test case is combination of one value for each factor
- Certain test cases invalid, incorporate constraints
- From pass/fail data identify *t*-way combinations which trigger failure among actual test cases (fault localization)
- No statistical model used in data analysis: test plan need not be balanced like classical DoE
- Choice of factors and test values highly critical for effectiveness of combinatorial testing
  - Information about nature of faults to be detected helpful

Orthogonal arrays

- Fixed-value OA(N, v^k, t) has four parameters N, k, v, t : It is a matrix such that every t-columns contain all t-tuples <u>the same number of times</u>
  - For OAs strength t is generally 2
  - Index of OA is number of times every t-tuple appears
  - Another notation OA(N, k, v, t)
- Mixed-value orthogonal array OA(N,$v_1^{k1}v_2^{k2}\ldots v_n^{kn}$, t) is a variation of fixed value OA where $k1$ columns have v_1 distinct values, $k2$ columns have v_2 values, ..., kn columns have v_n values $k = k1 + k2 +\ldots + kn$

Covering arrays

- Fixed-value CA(N, v^k, t) has four parameters N, k, v, t : It is a matrix such that every t-columns contain all t-tuples <u>at least once</u>
  - For CAs strength t can be any integer k or less
  - OA(N, v^k, t)) of index one is covering array with min test cases
  - However OA of index 1 are rare
  - Most CA are unbalanced
  - Another notation CA(N, k, v, t)
- Mixed-value covering array CA(N,$v_1^{k1}v_2^{k2}\ldots v_n^{kn}$, t) is a variation of fixed value CA where $k1$ columns have v_1 distinct values, $k2$ columns have v_2 values, ..., kn columns have v_n values and $k = k1 + k2 + \ldots + kn$

Combinatorial structure $2^4 \times 3^1$, need strength $t = 2$
OA for $2^4 \times 3^1$ dose not exist

OA(8, $2^4 4^1$, 2)

	a	b	c	d	e
1.	0	0	0	0	0
2.	1	1	1	1	0
3.	0	0	1	1	1
4.	1	1	0	0	1
5.	0	1	0	1	2
6.	1	0	1	0	2
7.	0	1	1	0	~~3~~ 2
8.	1	0	0	1	~~3~~ 2

CA(8, $2^4 3^1$, 2)

	a	b	c	d	e
1.	0	0	0	0	0
2.	1	1	1	1	0
3.	0	0	1	1	1
4.	1	1	0	0	1
5.	0	1	0	1	2
6.	1	0	1	0	2

OAs useful but have limitations

- OAs do not exist for many combinatorial test structures
  - Construction requires advanced mathematics
  - http://www2.research.att.com/~njas/oadir/
- Most OAs of strength t = 2; some t = 3 recent
- Most fixed-value; some mixed value OAs recent
- Combinatorial test structure fitted to suitable OA
  - We saw how OA(8, $2^4 \times 4^1$, 2) can be used for $2^4 \times 3^1$
- Constraints destroy balance property of OA

Benefits of CAs for generating test suites

- CAs available for any combinatorial test structure
- CAs available for any required strength (*t*-way) testing
- For a combinatorial test structure if OA exists then CA of same or fewer test runs can be obtained
- When numbers of factors large, CAs of few tests exist
- Generally CAs not balanced (like OAs) not needed in software testing
- Certain tests invalid, constraints can be incorporated
  - Coverage defined relative to valid test cases

Test suite for 2-way testing based on covering array

An application must run on various 3-OS, 2-Browser, 2-Protocol, 2-CPU type, and 3-DBMS, combinatorial test structure: $2^3 3^2$

Test	OS	Browser	Protocol	CPU	DBMS
1	XP	IE	IPv4	Intel	MySQL
2	XP	Firefox	IPv6	AMD	Sybase
3	XP	IE	IPv6	Intel	Oracle
4	OS X	Firefox	IPv4	AMD	MySQL
5	OS X	IE	IPv4	Intel	Sybase
6	OS X	Firefox	IPv4	Intel	Oracle
7	RHL	IE	IPv6	AMD	MySQL
8	RHL	Firefox	IPv4	Intel	Sybase
9	RHL	Firefox	IPv4	AMD	Oracle
10	OS X	Firefox	IPv6	AMD	Oracle

All pairs of values of five factors covered by 10 test cases

Size of test suites for various values of *t* based on CA

Combinatorial test structure: $2^3 3^2$

t	# Test cases	% of Exhaustive
2	10	14
3	18	25
4	36	50
5	72	100

Android smart phone configuration options
Combinatorial test structure: $3^3 4^4 5^2$

Factors	Test Values	Number
HARDKEYBOARDHIDDEN	NO, UNDEFINED, YES	3
KEYBOARDHIDDEN	NO, UNDEFINED, YES	3
KEYBOARD	12KEY, NOKEYS, QWERTY, UNDEFINED	4
NAVIGATIONHIDDEN	NO, UNDEFINED, YES	3
NAVIGATION	DPAD, NONAV, TRACKBALL, UNDEFINED, WHEEL	5
ORIENTATION	LANDSCAPE, PORTRAIT, SQUARE, UNDEFINED	4
SCREENLAYOUT_LONG	MASK, NO, UNDEFINED, YES	4
SCREENLAYOUT_SIZE	LARGE, MASK, NORMAL, SMALL, UNDEFINED	5
TOUCHSCREEN	FINGER, NOTOUCH, STYLUS, UNDEFINED	4

Size of test suites for various values of *t* based on CA

Combinatorial test structure: $3^3 4^4 5^2$

t	# Test Cases	% of Exhaustive
2	29	0.02
3	137	0.08
4	625	0.4
5	2532	1.5
6	9168	5.3

Some comments on Combinatorial *t*-way testing

- CT one of many complementary testing methods
- CT can reveal faults, not guarantee their absence (in this sense software testing is about risk management)
- CT can reveal many types of faults
- CT can be used in unit, integration, system testing
- CT better than random (fewer test runs); may be better than human generated test suites (better coverage)

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

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

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	18s	NA	**>1 d**	4580	43.54	NA	>1 day	313056	1549
6	10941	65.03	NA	>1 day	11625	470	NA	>1 day	1070048	12600

Combinatorial testing is a generic methodology

- Software testing
  - Test input space, test configuration space
- Computer/network security
  - Network deadlock detection, buffer overflow
  - http://csrc.nist.gov/groups/SNS/acts/index.html
- Testing Access Control Policy Systems
  - Security, privacy (e.g. health records)
  - http://csrc.nist.gov/groups/SNS/acpt/index.html
- Explore search space for study of gene regulations
  - http://www.plantphysiol.org/content/127/4/1590.full
- Optimization of simulation models of manufacturing
  - http://publications.lib.chalmers.se/cpl/record/index.xsql?pubid=103117

Summary

- Combinatorial testing is a variation of DoE adapted for testing software and hardware-software systems
- Early use was limited to pairwise (2-way) testing
- Investigations of actual faults suggests that up to 6-way testing may be needed
- Combinatorial *t*-way testing for *t* up to 6 is possible by use of covering arrays
- ACTS is useful tool for generating *t*-way test suites based on CAs, supports constraints
- Combinatorial testing useful when testing expressed in terms of factors, discrete test values, critical event happens when certain *t*-way combination encounters