Combinatorial Testing and Covering Arrays

Lucia Moura
School of Electrical Engineering and Computer Science
University of Ottawa
lucia@eecs.uottawa.ca

Winter 2017

Software and Network Testing

We want to test a **system**:

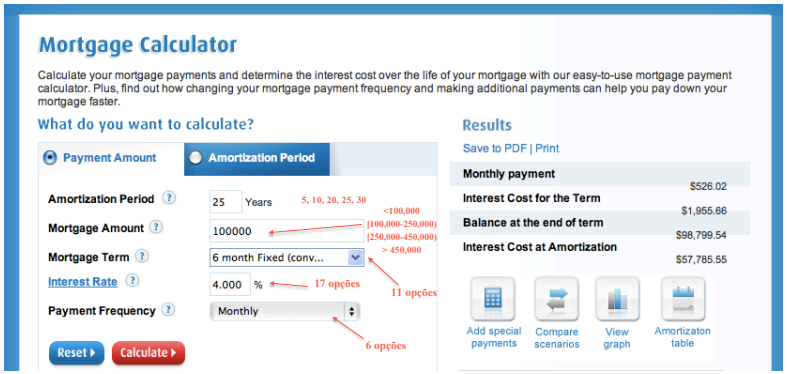
- a program
- a circuit
- a package that integrates several pieces of software
- different platforms where a package needs to run correctly
- a highly configurable software
- a GUI interface
- a cloud application

We would like a **test suite** that gives a good **coverage** of the input parameter space in order to detect the maximum number of **errors/bugs/faults**.

Combinatorial Software Testing

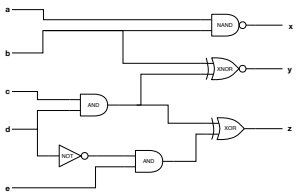
First we isolate the system **parameters** and its possible **values**

- the input parameters of a program and its possible values



$(5, 4, 11, 17, 6)$

- the inputs of a circuit: 5 binary inputs



$(2, 2, 2, 2, 2)$

- the components of a platform and its configurations

	Component			
	Web Browser	**Operating System**	**Connection Type**	**Printer Config**
Config:	Netscape(0) IE(1) Other(2)	Windows(0) Macintosh(1) Linux(2)	LAN(0) PPP(1) ISDN(2)	Local (0) Networked(1) Screen(2)

$(3, 3, 3, 3)$

Pairwise Testing

Testing a system with $k = 4$ components each having $v = 3$ values:

	Component			
	Web Browser	**Operating System**	**Connection Type**	**Printer Config**
Config:	Netscape(0) IE(1) Other(2)	Windows(0) Macintosh(1) Linux(2)	LAN(0) PPP(1) ISDN(2)	Local (0) Networked(1) Screen(2)

Test all possibilities: $3^4 = 81$ tests.

Pairwise testing can be done with only 9 tests.

Test Case	Browser	OS	Connection	Printer
1	NetScape	Windows	LAN	Local
2	NetScape	Linux	ISDN	Networked
3	NetScape	Macintosh	PPP	Screen
4	IE	Windows	ISDN	Screen
5	IE	Macintosh	LAN	Networked
6	IE	Linux	PPP	Local
7	Other	Windows	PPP	Networked
8	Other	Linux	LAN	Screen
9	Other	Macintosh	ISDN	Local

(example from Colbourn 2004)

Covering Arrays with strength $t = 2$, $k = 4$ parameters, $v = 3$ values for each, can cover all pairwise interactions with $N = 9$ tests.

Pairwise Testing

Covering array:
strength $t = 2$, $k = 5$ paramters, values $(3, 2, 2, 2, 3)$, $N = 10$ tests

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	RHEL	IE	IPv6	AMD	MySQL
8	RHEL	Firefox	IPv4	Intel	Sybase
9	RHEL	Firefox	IPv4	AMD	Oracle
10	OS X	Firefox	IPv6	AMD	Oracle

(example taken from Khun, Kacker and Lei 2010)

testing all possibilities $(t = 5)$: $3^2 \times 2^3 = 72$ tests
pairwise testing $(t = 2)$: 10 tests

Pairwise Testing

Covering array:
strength $t = 2$, $k = 5$ paramters, values $(3, 2, 2, 2, 3)$, $N = 10$ tests

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	RHEL	IE	IPv6	AMD	MySQL
8	RHEL	Firefox	IPv4	Intel	Sybase
9	RHEL	Firefox	IPv4	AMD	Oracle
10	OS X	Firefox	IPv6	AMD	Oracle

(example taken from Khun, Kacker and Lei 2010)

testing all possibilities $(t = 5)$: $3^2 \times 2^3 = 72$ tests
pairwise testing $(t = 2)$: 10 tests

Why to use pairwise testing?

- **Economy: we use a minimal number of tests.**
  example: $k = 20$ parameters with $v = 10$ values each.
  testing all combinations: 10^{20} tests (in general $= v^k$)
  pairwise testing: 155 tests (in general $O(v \log k)$)
- **Robustness: we have good coverage in practice.**
  most software errors (75%-80%) are caused by certain parameter values or by the interaction of two of values.
  "Evaluating FDA recall class failures in medical devices... 98% showed that the problem could have been detected by testing the device with all pairs of parameter settings." (Wallace and Kuhn, 2001)
  Cohen, Dalal, Fredman, Patton (1996) - AETG software
  Dalal, Karunanithi, Leaton, Patton, Horowicz (1999)
  Kuhn and Reilly (2002)

  covering pairs imply other coverage measures.
  "Our initial trial of this was on a subset Nortel's internal e-mail system where we able cover 97% of branches with less than 100 valid and invalid testcases, as opposed to 27 trillion exhaustive test cases." (Burr and Young, 1998)
  "The block coverage obtained for [pairwise] was comparable with that achieved by exhaustively testing all factor combinations ..." (Dunietz et al., 1997)
  Cohen, Dalal, Fredman, Patton (1996, 1997) - AETG software

Increasing the coverage strength (t-way coverage)

- we can use intermediate strength values between $t = 2$ (pairwise) and $t = k$ (testing full parameter space).
- the "tradeoff" is that increasing t, we increase robustness, but also the number of tests
- studies show that usually $t \in [2, 6]$ is sufficient to detect all the software errors

Kuhn, Wallace e Gallo (2004)

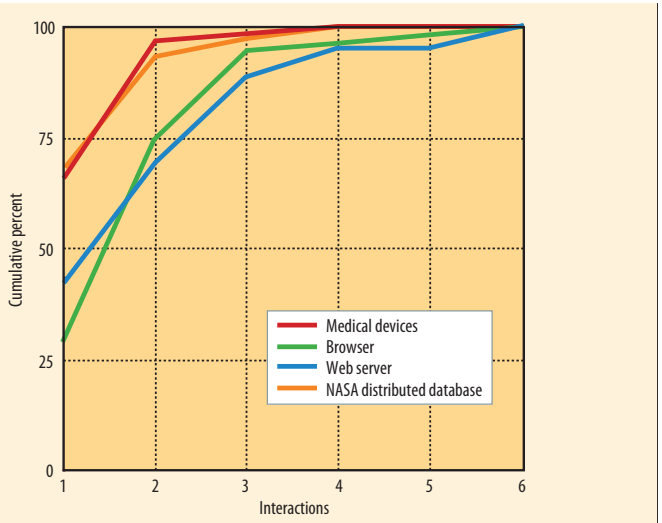


Figure 2. Cumulative error detection rate for fault-triggering conditions. Many faults were caused by a single parameter value, a smaller proportion resulted from an interaction between two parameter values, and progressively fewer were triggered by three-, four-, five, and six -way interactions.

Covering Arrays

t-way combinatorial testing requires covering arrays of strength t
strength $t = 3$; $v = 2$ symbols; $k = 10$ columns; $N = 13$ rows

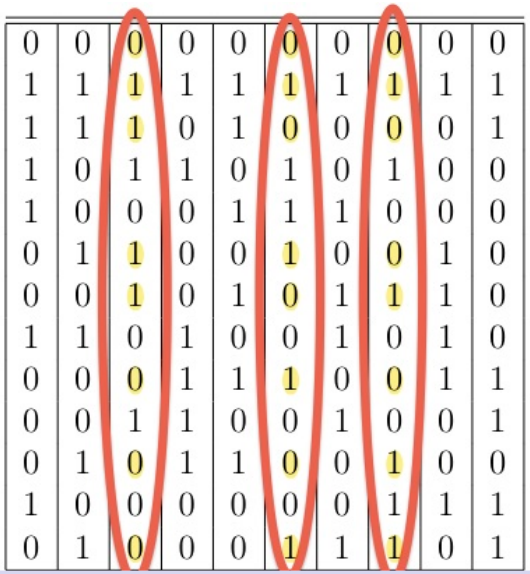
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	1
1	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0
1	1	0	1	0	0	1	0	1	0
0	0	0	1	1	1	0	0	1	1
0	0	1	1	0	0	1	0	0	1
0	1	0	1	1	0	0	1	0	0
1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	1	1	1	0	1

Definition (Covering Arrays)

A *covering array* of strength t, k factors, v symbols per factor and size N, denoted $CA(N; t, k, v)$, is an $N \times k$ matrix with symbols from a v-ary alphabet G such that in each $t \times N$ subarray, each t-tuple in G^t is covered *at least* once.

Covering Arrays

t-way combinatorial testing requires covering arrays of strength t
strength $t = 3$; $v = 2$ symbols; $k = 10$ columns; $N = 13$ rows



Definition (Covering Arrays)

A *covering array* of strength t, k factors, v symbols per factor and size N, denoted $CA(N; t, k, v)$, is an $N \times k$ matrix with symbols from a v-ary alphabet G such that in each $t \times N$ subarray, each t-tuple in G^t is covered *at least* once.

Covering Array Minimization

Given t (strength), k (number of paramters) and v (#values).

Minimize N (#tests)

$CAN(t,k,v) = \min\{N : \text{ there exists a } CA(N;t,k,v)\}.$

Covering array logarithmic growth

- $CAN(t=2,k,v=2) = \{\min N : \binom{N-1}{\lceil N/2 \rceil} \geq k\} = \log k(1+o(1))$ (Katona 1973, Kleitman and Spencer 1973)
- $t=2$, $v>2$ fixed, $k \to \infty$: $CAN(t=2,k,v) = \frac{v}{2}\log k(1+o(1))$ (Gargano, Korner and Vaccaro 1994)
- $CAN(t,k,v=2) \leq 2^t t^{O(\log t)} \log k$ (Naor et al 1993,1996,1998)
- $CAN(t,k,v) \leq v^t(t-1)\log k(1+o(1))$ (Godbole, Skipper and Sunley 1996)

Covering array minimization and logarithmic growth

Given t (strength), k (number of parameters) and v (#values).

Minimize N (#tests)

$CAN(t, k, v) = \min\{N : \text{ there exists a } CA(N; t, k, v)\}.$

For fixed v and t $CAN(t, k, v) = O(\log k)$.

Use the greedy density method (Bryce & Colbourn 2007).
One-test-at-a-time greedy method that garantees $N = O(\log k)$.

Excellent for software testing: #tests grows with the log of the #parameters!

Construction of (minimum/small) covering arrays

- **combinatorial methods: recursive and direct**
  Survey: Charlie Colbourn, "Combinatorial Aspects of Covering Arrays", 2004 (34 pages)
- **algorithms**
  - **greedy methods**:
    • AETG (D. Cohen, Dalal, Fredman, Patton 1996, 1997), one-test-at-a-time, tries to approximate logarithmic growth
    • greedy density method (Bryce e Colbourn 2007), one-test-at-a-time, logarithmic guaranty
    • IPOG algoritm (J. Lei), ACTS tool/NIST (Khun and Kacker): alternates row growth and column growth
  - **heuristic methods**
    • tabu search: Zekaoui (2006), Torres-Jimenez (2012)
    • simulated annealing: M. Cohen (2003-2008), Torres-Jimenez (2010-2012)

Covering Array Construction

- **Practical, more flexible methods:**
  greedy methods (fast, number of tests is not optimized)
  heuristic search (slower, number of tests is smaller)
- **Method to get the best possible covering arrays:**
  select the best results, using a combination of:
  good ingredients (direct constructions or heuristic searches)
  + the best recursive constructions

  See table maintained by Colbourn with the best known sizes of covering arrays.

Example of good ingredients to use in recursive constructions

- orthogonal arrays: $CA(N = q^2; t = 2, k \leq q + 1, v = q)$
  (Bush method using finite fields F_q)

$$\begin{bmatrix} 0000 \\ 0122 \\ 1220 \\ 2202 \\ 2021 \\ 0211 \\ 2110 \\ 1101 \\ 1012 \end{bmatrix}$$

(optimal N)

- method using LFSR for $t = 3$:
  $CA(N = 2q^3 - 1; t = 3, k \leq q^2 + q + 1, v = q)$
  (Raaphorst, Moura, Stevens 2012)

(optimal or near optimal N)

Example of a good recursive construction: Product

in this example: parameter $t = 2$

CA(4,3)

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

size=9

CA(3,3) with 3 disjoint rows:

OD(3,3)

0	0	0
1	1	1
2	2	2
0	1	2
0	2	1
1	0	2
1	2	0
2	0	1
2	1	0

size=6

CA(12=4*3,3)

1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	2	0	1	1	2	0	1	1	2
0	2	2	1	0	2	2	1	0	2	2	1
1	0	2	2	1	0	2	2	1	0	2	2
1	1	0	1	1	1	0	1	1	1	0	1
1	2	1	0	1	2	1	0	1	2	1	0
2	0	1	1	2	0	1	1	2	0	1	1
2	1	2	0	2	1	2	0	2	1	2	0
2	2	0	2	2	2	0	2	2	2	0	2
0	0	0	0	1	1	1	1	2	2	2	2
0	0	0	0	2	2	2	2	1	1	1	1
1	1	1	1	0	0	0	0	2	2	2	2
1	1	1	1	2	2	2	2	0	0	0	0
2	2	2	2	0	0	0	0	1	1	1	1
2	2	2	2	1	1	1	1	0	0	0	0

size=9+6=15

CA(N1,k1,g)+OD(N2,k2,g)= CA(N1+N2,k1*k2,g)

Current State

- Combinatorial sofware testing is useful and effective.
- There are ready-to-use tools for use in applications:
  • ACTS by NIST (EUA) $t \leq 6$ (open source, free)
  • Hexawise: comercial $t \leq 6$ (SaaS, free for academic use, nonprofit e companies up to 5 users; otherwise annual fee)
  • Testcover.com: automatic generator ($t = 2$) (SaaS, subscription: \$100/month)
- There is active research in the area of algorithms and combinatorial constructions to optimize the number of tests (rows) in covering arrays.
  There are some efforts to deal with additional restrictions.
- There is active research in the area of software testing evaluating the effectiveness and adapting combinatorial software testing to many types of applications.

An in-depth view of covering array constructions

- The rest of our study of covering arrays will use a survey talk entitled "Covering Arrays and Generalizations" (2006).
- We will refer to the following sections covered there:
  - Introduction and summary of results.
  - Constructions: using OAs, blocksize recursive (product), direct construction of binary CAs.
  - Covering array on graphs.
  - Other generalizations.

References

- C. Colbourn, Combinatorial Aspects of Covering Arrays, *Le Matematiche* (Catania), 2004. (survey article)
- L. Moura, Covering Arrays and Generalizations, *Survey Talk, UPC seminar,* November 2006.