Combinatorial Testing and Covering Arrays

Software and Network Testing

We want to test a **system**:

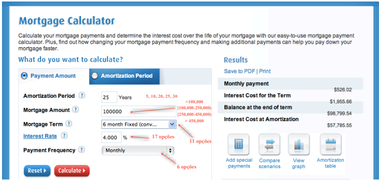
- a program
- a circuit
- a package that integrates several pieces of software
- different platforms where a package needs to run correctly
- a highly configurable software
- a GUI interface
- a cloud application

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



Mortgage Calculator

Calculate your mortgage payments and determine the interest cost over the life of your mortgage with our easy-to-use mortgage payment calculator. Plus, find out how changing your mortgage payment frequency and making additional payments can help you pay down your mortgage faster.

What do you want to calculate?

● Payment Amount ○ Amortization Period

Amortization Period: 25 Years — 5, 10, 20, 25, 30

Mortgage Amount: 100000 — <100,000 / [100,000-250,000) / [250,000-450,000) / > 450,000

Mortgage Term: 6 month Fixed (conv... — 11 opções

Interest Rate: 4.000 % — 17 opções

Payment Frequency: Monthly — 6 opções

Reset ▸ Calculate ▸

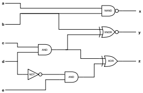
Results

Save to PDF | Print

Monthly payment	$526.02
Interest Cost for the Term	$1,955.66
Balance at the end of term	$98,799.54
Interest Cost at Amortization	$57,785.55

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

- 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)	Windows(0)	LAN(0)	Local (0)
	IE(1)	Macintosh(1)	PPP(1)	Networked(1)
	Other(2)	Linux(2)	ISDN(2)	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)	Windows(0)	LAN(0)	Local (0)
	IE(1)	Macintosh(1)	PPP(1)	Networked(1)
	Other(2)	Linux(2)	ISDN(2)	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

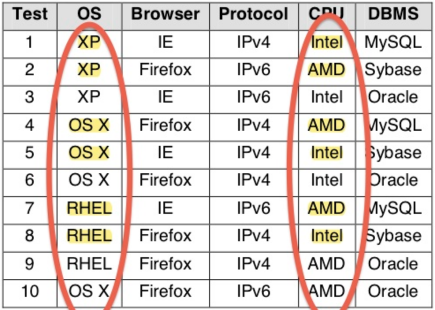
Test	OS	Browser	Protocol	CPU	DBMS
1	XP	IE	IPv4	Intel	MySQL
2	XP	Firefox	IPv6	AMD	Sybase
3	XP	IE	IPv6	Intel	Oracle
4	OS X	Firefox	IPv4	AMD	MySQL
5	OS X	IE	IPv4	Intel	Sybase
6	OS X	Firefox	IPv4	Intel	Oracle
7	RHEL	IE	IPv6	AMD	MySQL
8	RHEL	Firefox	IPv4	Intel	Sybase
9	RHEL	Firefox	IPv4	AMD	Oracle
10	OS X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

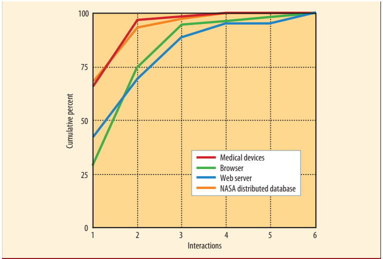


Figure 2. Cumulative error detection rate for fault-triggering conditions. Many faults were caused by a single parameter value, a smaller proportion resulted from an interaction between two parameter values, and progressively fewer were triggered by three-, four-, five, and six -way interactions.

Kuhn, Wallace e Gallo (2004)

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.

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.