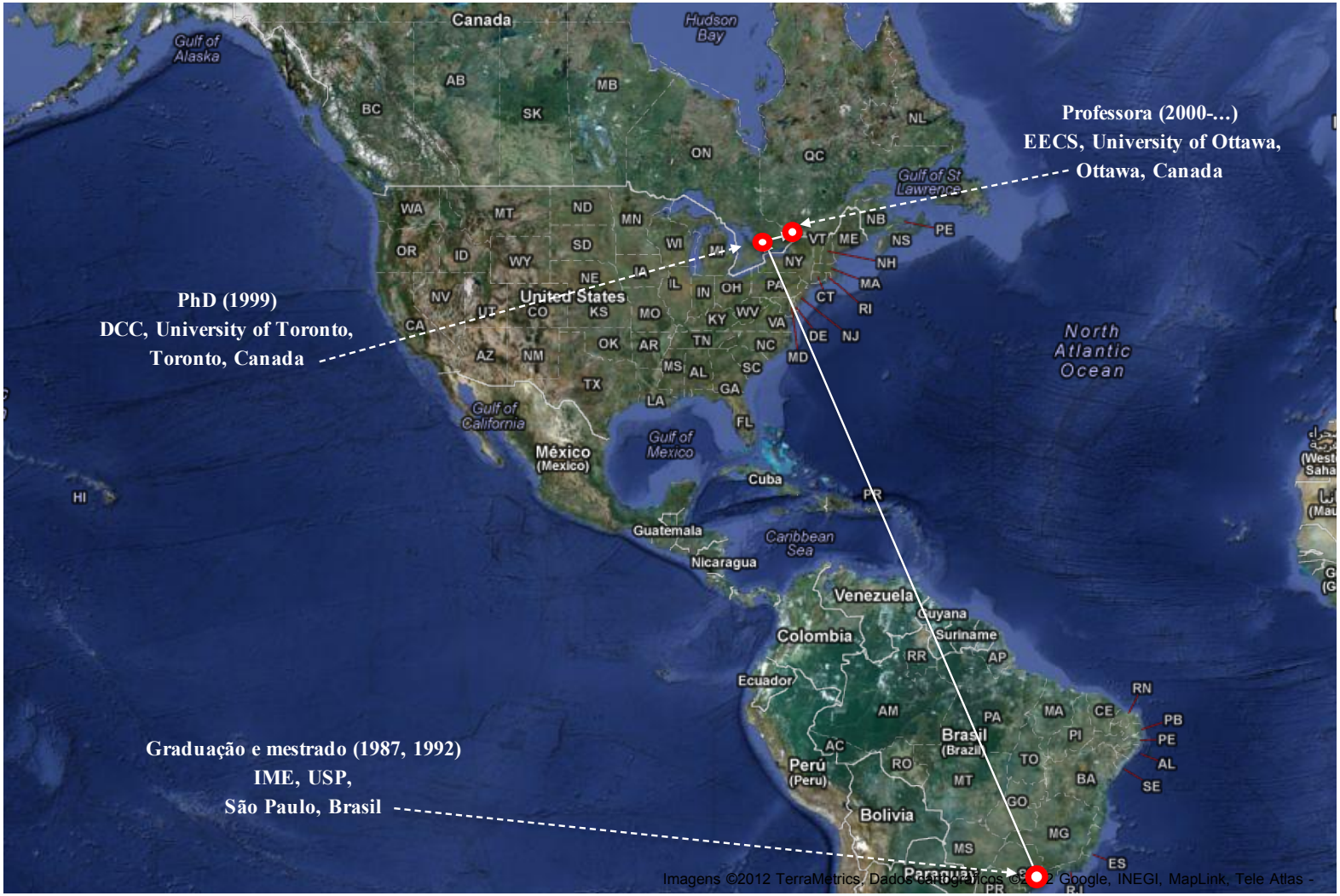
Teste Combinatório de Software

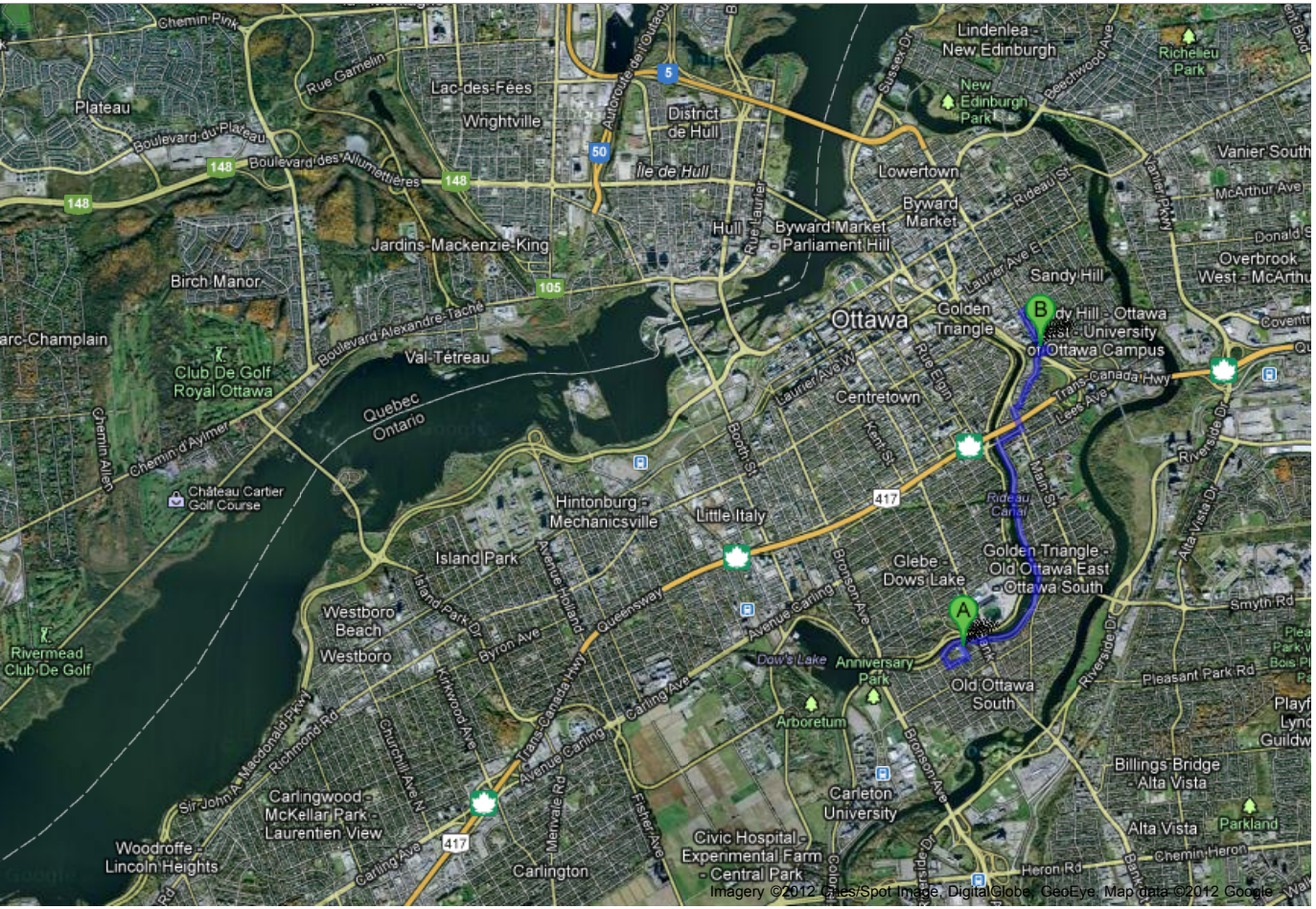
Lucia Moura
School of Electrical Engineering and Computer Science
University of Ottawa
lucia@eecs.uottawa.ca

Palestra no SECCOM, INE, UFSC, 2012

Trajetória geográfica e acadêmica



Ottawa, Canadá



EECS

University of Ottawa

Teste Combinatório de Software

Queremos testar um **sistema**:

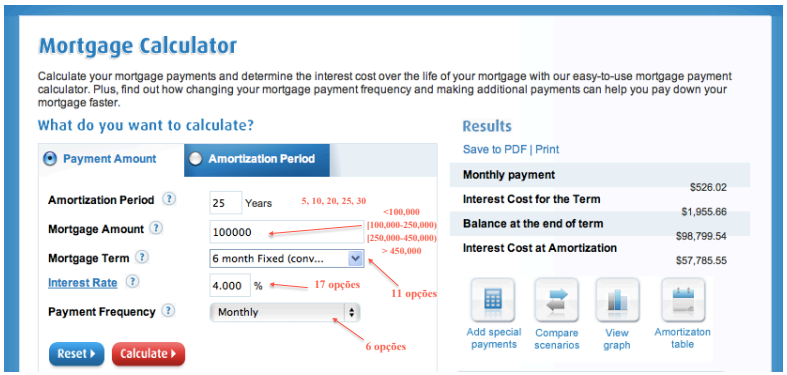
- um programa
- um circuito
- um pacote que integra vários softwares
- várias plataformas diferentes em que um pacote tenha que funcionar
- um software altamente configurável
- uma gui interface
- uma aplicação na "cloud"

Queremos uma **bateria de testes** que dê uma boa **cobertura** do sistema para detectar **erros/bugs/falhas**.

Teste Combinatório de Software

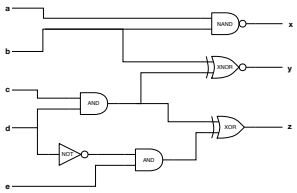
Primeiro isolamos os **parâmetros** do sistema e possíveis **valores**

- os parâmetros de entrada de um programa e seus valores



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

- as entradas de um circuito: 5 entradas binárias



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

- componentes de uma plataforma e suas configurações

	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)

$(3, 3, 3, 3)$

Teste combinatório de pares (pairwise testing)

Testando um sistema com $k = 4$ componentes cada um com $v = 3$ valores:

	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)

Testar todas as possibilidades: $3^4 = 81$ testes.

Teste combinatório de **pares** de parâmetros pode ser feito com 9 testes.

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

(examplo de Colbourn 2004)

Arranjo de cobertura com forca $t = 2$, $k = 4$ parametros, $v = 3$ valores cada cobre todas as interacoes dois a dois com $N = 9$ tests.

Teste combinatório de pares (pairwise testing)

Arranjo de cobertura:
força $t = 2$, $k = 5$ parâmetros, valores $(3, 2, 2, 2, 3)$, $N = 10$ testes

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)

testar todas as possibilidades $(t = 5)$: $3^2 \times 2^3 = 72$ testes
teste de pares $(t = 2)$: 10 testes

Teste combinatório de pares (pairwise testing)

Arranjo de cobertura:
força $t = 2$, $k = 5$ parâmetros, valores $(3, 2, 2, 2, 3)$, $N = 10$ testes

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)

testar todas as possibilidades $(t = 5)$: $3^2 \times 2^3 = 72$ testes
teste de pares $(t = 2)$: 10 testes

Por que usar teste de pares (pairwise testing)?

- **Economia: usamos um número mínimo de testes.**
  exemplo: $k = 20$ parâmetros com $v = 10$ valores cada.
  testar todas as combinações:10^{20} testes(em geral $= v^k$)
  teste de pares:155 testes (em geral em $O(v^2 \log k)$)
- **Robustez: temos uma boa cobertura de erros na prática.**
  a maior parte dos erros em software (75%-80%) são causados por certos valores ou por interação de 2 valores.
  "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)

  cobertura de pares implicam outras medidas de cobertura.
  "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 testcases." (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

Aumentando a força da cobertura (t-way coverage)

- podemos usar outros valores intermediários de força entre $t = 2$ (pairwise) e $t = k$ (testar todo o espaço de parâmetros).
- o "tradeoff" é que aumentando o t, aumenta-se a robustez, mas também aumenta-se o número de testes
- estudos mostram que normalmente $t \in [2, 6]$ é suficiente para detectar todos os errors

Kuhn, Wallace e Gallo (2004)

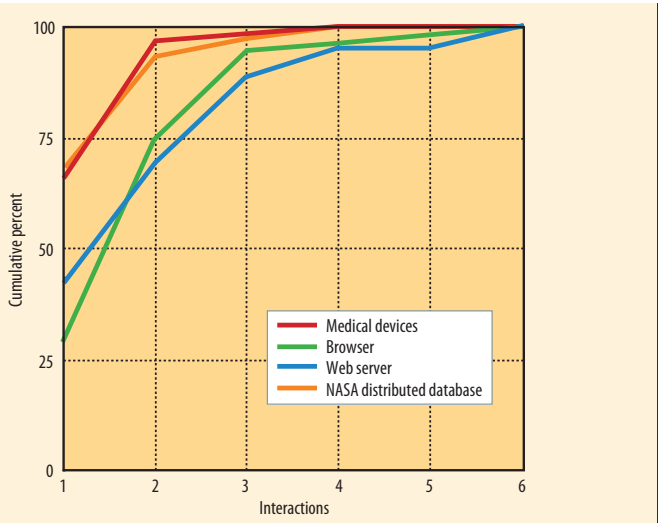


Figure 2. Cumulative error detection rate for fault-triggering conditions. Many faults were caused by a single parameter value, a smaller proportion resulted from an interaction between two parameter values, and progressively fewer were triggered by three-, four-, five, and six -way interactions.

Teste combinatório de software, arranjos de cobertura

Teste combinatório t-way requer arranjos de cobertura de força t
força $t = 3$; $v = 2$ símbolos; $k = 10$ colunas; $N = 13$ linhas

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

Definicao: Arranjos de Cobertura

Um *arranjo de cobertura* de força t, k fatores, v símbolos por fator e tamanho N, denotado $CA(N; t, k, v)$, é uma matrix $N \times k$ com símbolos de um alfabeto v-ário G tal que em cada sub-arranjo $t \times N$, cada t-tupla em G^t é coberta pelo menos uma vez.

Teste combinatório de software, arranjos de cobertura

Teste combinatório t-way requer arranjos de cobertura de força t
força $t = 3$; $v = 2$ símbolos; $k = 10$ colunas; $N = 13$ linhas

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

Definicao: Arranjos de Cobertura

Um *arranjo de cobertura* de força t, k fatores, v símbolos por fator e tamanho N, denotado $CA(N; t, k, v)$, é uma matrix $N \times k$ com símbolos de um alfabeto v-ário G tal que em cada sub-arranjo $t \times N$, cada t-tupla em G^t é coberta pelo menos uma vez.

Minimização de arranjos de cobertura

Dados t (força), k (número de parâmetros) and v (#valores).

Minimize N (#tests)

$CAN(t,k,v) = \min\{N : \ existe\ um\ CA(N;t,k,v)\}.$

Crescimento logarímitico de arranjos de cobertura

- $CAN(t=2,k,v=2) = \{\min N : \binom{N-1}{\lceil N/2 \rceil} \geq k\} = \log k(1+o(1))$ (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)

Minimização de arranjos de cobertura e crescimento logarítmo

Dados t (força), k (número de componentes) e v (#valores).

Minimize N (#testes)

$CAN(t,k,v) = \min\{N : \ existe\ um\ CA(N;t,k,v)\}.$

Para v e t fixos $CAN(t,k,v) = O(\log k)$.

Use o método guloso de densidade (Bryce & Colbourn 2007).
Método guloso de um-teste-por-vez garante $N = O(\log k)$.

Ótimo para teste de software: #testes cresce com o log do #parâmetros!

Construção de arranjos de cobertura mínimos

- **métodos combinatórios: recursivos e diretos**
  Survey: Charlie Colbourn, "Combinatorial Aspects of Covering Arrays", 2004 (34 paginas)
- **algoritmos**
  - **métodos gulosos**:
    • AETG (D. Cohen, Dalal, Fredman, Patton 1996, 1997), um-teste-por-vez, tenta aproximar garantia logarítmica
    • método guloso de densidade (Bryce e Colbourn 2007), um-teste-por-vez, garantia logarítmica
    • IPOG algoritm (J. Lei), ACTS tool/NIST (Khun e Kacker): alternância de crescimento de linhas e colunas
  - **métodos de busca heurística**
    • tabu search: Zekaoui (2006), Torres-Jimenez (2012)
    • simulated annealing: M. Cohen (2003-2008), Torres-Jimenez (2010-2012)

Construção de arranjos de cobertura

- **Métodos práticos, mais flexíveis:**
  métodos gulosos (rápidos, número de testes não é otimizado)
  busca heurística (mais lentos, resultados de melhor qualidade)
- **Método para obter os melhores arranjos de cobertura possível:**
  selecione os melhores resultados, usando combinação de:
  bons ingredientes (construções diretas e buscas heurísticas)
  + e as melhores construções combinatórias recursivas

  Ver a tabela mantida por Colbourn com os melhores tamanhos de arranjos de cobertura conhecidos.

Exemplo de bons ingredientes p/ usar em construção recursiva

- arranjos ortogonais: $CA(N = q^2; t = 2, k \leq q + 1, v = q)$
  (método clássico usando corpos finitos F_q)

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

(N otimo)

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

(N ótimo ou perto do ótimo)

Exemplo de uma boa construção recursiva: Produtos

neste exemplo: parâmetro $t = 2$

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)

Generalizações de Arranjos de Cobertura

Queremos construções eficazes baseadas em teoria combinatória para lidar com restrições adicionais em teste de software:

- **Número de valores variado** (v varia com a coluna: $v_1, v_2, \ldots, v_k$)
- **Configurações proibidas**
  ACs evitando pares proibidos (CAFEs) e generalizações
- **Variações na força**
  ACs de força variável.
- **Capacidade de determinação dos erros**
  locating arrays, detecting arrays, error-locating arrays (ELAs)
- **Outras generalizações**

Número de valores variados

$$MCA(N = 10; t = 2, k = 5, v = (3, 2, 2, 2, 3))$$

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

- (exemplo de Khun, Kacker and Lei 2010)

Construções combinatórias:
Moura, Stardom, Stevens, Williams (2003)
Colbourn, Martirosyan, Mullen, Shasha, Sherwood, Yucas (2005)

Número de valores variados

$$MCA(N = 10; t = 2, k = 5, v = (3, 2, 2, 2, 3))$$

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

(exemplo de Khun, Kacker and Lei 2010)

Construções combinatórias:
Moura, Stardom, Stevens, Williams (2003)
Colbourn, Martirosyan, Mullen, Shasha, Sherwood, Yucas (2005)

Configurações proibidas/CAFEs

Internet explorer (IE) so pode ser usado com Windows (XP) Pares proibidos: (OS X, IE) (RHL,IE)

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	Firefox	IPv4	Intel	Sybase
6	OS X	Firefox	IPv6	AMD	Oracle
7	RHL	Firefox	IPv6	Intel	MySQL
8	RHL	Firefox	IPv4	Intel	Oracle
9	XP	IE	IPv4	AMD	Sybase

(exemplo de Khun, Kacker and Lei 2010)

Modelo: grafo c/ 3+2+2+2+3 vertices, 2 arestas: {OS X, IE} {RHL,IE}

Construções combinatórias:

- Danziger, Mendelsohn, Moura, Stevens (2008, 2009)
- Maltais, Moura (2010)

Aplicações:

- Cohen, Dwyer, Shi (2007, 2008)

Força variável: AC em grafos

Testes cobrindo relações de entrada-saida (nota: a matriz está transposta)

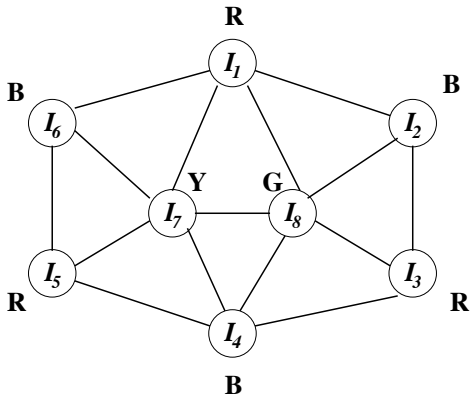


Figure 1. The graph for the problem instance where $\mathcal{I} = \{I_1, I_2, \ldots, I_8\}$ and $\mathcal{O} = \{\{I_1, I_2, I_8\}, \{I_2, I_3\}, \{I_3, I_4, I_8\}, \{I_4, I_5, I_7\}, \{I_5, I_6\}, \{I_1, I_6, I_7\}, \{I_7, I_8\}\}$.

I_1	1	1	1	0	0	0	1	0
I_2	1	1	0	1	0	1	0	0
I_3	1	1	1	0	0	0	1	0
I_4	1	1	0	1	0	1	0	0
I_5	1	1	1	0	0	0	1	0
I_6	1	1	0	1	0	1	0	0
I_7	1	0	1	1	1	0	0	0
I_8	1	0	0	0	0	1	1	1

(exemplo de Cheng, Dumitrescu, Schroeder 2003)

Coloração de grafos, homomorfismos de grafos

- Meagher, tese de PhD (2005)
- Meagher, Stevens (2005), Meagher, Moura, Zekaoui (2007)
- Cheng, Dumitrescu, Schroeder (2003), Cheng (2007)

Força variável: AC em grafos

Testes cobrindo relações de entrada-saida (nota: a matriz está transposta)

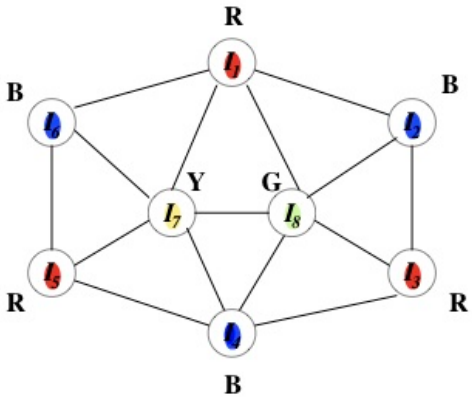


Figure 1. The graph for the problem instance where $\mathcal{I} = \{I_1, I_2, \ldots, I_8\}$ and $\mathcal{O} = \{\{I_1, I_2, I_8\}, \{I_2, I_3\}, \{I_3, I_4, I_8\}, \{I_4, I_5, I_7\}, \{I_5, I_6\}, \{I_1, I_6, I_7\}, \{I_7, I_8\}\}$.

I_1	1	1	1	0	0	0	1	0
I_2	1	1	0	1	0	1	0	0
I_3	1	1	1	0	0	0	1	0
I_4	1	1	0	1	0	1	0	0
I_5	1	1	1	0	0	0	1	0
I_6	1	1	0	1	0	1	0	0
I_7	1	0	1	1	1	0	0	0
I_8	1	0	0	0	0	1	1	1

(exemplo de Cheng, Dumitrescu, Schroeder 2003)

Coloração de grafos, homomorfismos de grafos

- Meagher, tese de PhD (2005)
- Meagher, Stevens (2005), Meagher, Moura, Zekaoui (2007)
- Cheng, Dumitrescu, Schroeder (2003), Cheng (2007)

ACs de força variável

VCAs = ACs em hipergrafos

f_1	f_2	f_3	f_4	f_5	f_6
1	1	1	3	2	1
1	1	2	1	3	2
1	1	3	2	1	3
1	2	1	2	2	4
1	2	2	3	3	5
1	2	3	1	1	6
1	3	1	1	2	7
1	3	2	2	3	8
1	3	3	3	1	9

f_1	f_2	f_3	f_4	f_5	f_6
2	1	1	1	1	5
2	1	2	2	2	9
2	1	3	3	3	7
2	2	1	3	1	8
2	2	2	1	2	3
2	2	3	2	3	1
2	3	1	2	1	2
2	3	2	3	2	6
2	3	3	1	3	4

f_1	f_2	f_3	f_4	f_5	f_6
3	1	1	2	3	6
3	1	2	3	1	4
3	1	3	1	2	8
3	2	1	1	3	9
3	2	2	2	1	7
3	2	3	3	2	2
3	3	1	3	3	3
3	3	2	1	1	1
3	3	3	2	2	5

Figure 2: A VCA$(27; \Lambda, 3^5 9^1)$ for $\Lambda = \{[5] \times \{6\}\} \cup \left(\binom{[5]}{3} \setminus 135\right)$: the rows are divided into three columns for compactness.

Se escolhe níveis de cobertura diferentes para diferentes conjuntos de parâmetros.

- Cohen, Colbourn, Collofello, Gibbons, Mugridge (2003)
- Raaphorst, Moura and Stevens (2011, 2012)

Como localizar as interações que causam falha?

Example:

0	0	0	0	F
0	1	1	0	F
0	2	2	2	P
1	0	2	1	P
1	1	0	2	P
1	2	1	0	P
2	0	1	2	P
2	1	2	0	P
2	2	0	1	P
0	0	1	1	P

Vários possíveis conjuntos de interações errôneas !

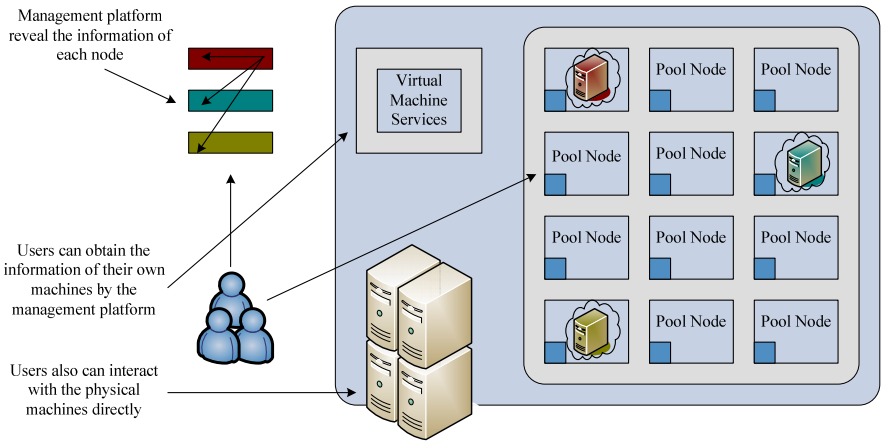
Como construir um arranjo que nos indique as interações que causam erros?

Arranjos usados para localização de erros

- Yilmaz, Cohen and Porter (2006):
  arranjos de cobertura+ árvores de classificação.
- Colbourn and McClary (2008):
  locating arrays, detecting arrays.
- Martinez, Moura, Stevens and Panario (2010):
  error-locating arrays (ELAs)
- Chodoriwsky, Moura (2012): generalizacao de algoritmo adaptativo para $t \geq 3$

Outras generalizações: aplicação em cloud computing

Jun &Meng ("Software Testing based on cloud computing" , 2011) discutem o modelo de usar **cloud computing** para teste de software.



The main process of cloud testing as follows:
1) Users login cloud testing provider's website and register user information;
2) Users apply for the test platform resources, applications need to describe the configuration requirements of virtual machine environment, Such as operating system version, browser version, memory size, hard disk size, hard disk speed, network bandwidth, firewall, etc.;
3) The service provider of cloud testing review the application and configure the test platform;
4) Under the rental agreement, the user (online) pre-paid service fee;
5) Users login the cloud testing platform, until the end of the testing;
6) According to the flow or time to pay the actual costs, the lease ends.

Uma idéia que proponho nesta palestra:

- usuário usa arranjos de cobertura no passo 2 (pode ser inviável); ou
- sofisticar o passo 2: o usuário só fornece as componentes e configurações desejadas para o teste e a força desejada; o cloud testing provider gera o arranjo de cobertura adequado
  (restrição: configurações das maquinas virtuais na cloud)

Discussão final

- Teste combinatório de software é util e efetivo.
- Existem tools disponíveis para uso imediato em aplicações:
  • ACTS do NIST (EUA) $t \leq 6$ (open source, gratuito)
  • Hexawise: comercial $t \leq 6$ (gratuito para uso acadêmico, nonprofit e compania com até 5 usuários, $\geq$\$50.000/ano)
  • Testcover.com: gerador automático ($t = 2$) na cloud/software as a service (SaaS); (assinatura: \$100/mês)
- Existe pesquisa ativa na área de algorítmos e construções combinatórias para otimizar o número de testes em arranjos de cobertura.
  Temos tido algum sucesso em lidar com restrições adicionais.
- Existe pesquisa ativa na área de teste de software avaliando a efetividade e adaptando teste combinatório de software aos mais diversos tipos de aplicações.