Input testing:
Partition and combinatorial testing

- Angelo Gargantini –

- Corso Testing e verifica del software

Outline

- Input testing

- Partition testing

  - A method to apply partition testing
  - How to choose variable values

- Combinatorial interaction of parameters

  - What is combinatorial testing
  - Efficiency: It can detect faults

- Generation techniques

  - IPO, AETG, IPOS

- Adding constraints

  - Logic approach, using SAT/SMT solving

What is Input testing

▸ "input space" testing or testing based on the **interfaces**

▸ No internal information about the system under test is considered, but only the information about the inputs

▸ It can be <u>model based testing</u>

  ▸ Model of the inputs

▸ <u>Program</u> based testing

  ▸ The program is analyzed to extract the parameters

    ▸ E.g. the parameters of a method. …

Advantages of Input based testing

▸ Can be equally applied at several levels of testing

  ▸ Unit
  ▸ Integration
  ▸ System

▸ Relatively easy to apply

  ▸ Test generation is simple, simpler than structure based testing or fault based

▸ Easy to adjust the procedure to get more or fewer tests

▸ No implementation knowledge is needed

  ▸ just the input space
  ▸ Usable even if the complete code/model is not accessible
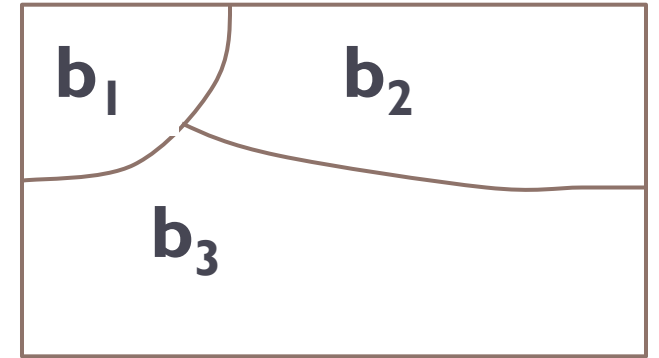
Partition testing

Problems …

▸ The <u>input domain</u> to a program contains all the possible inputs to that program

  ▸ For even small programs, the input domain is so large that it might as well be <u>infinite</u>

▸ Testing is fundamentally about <u>choosing finite sets</u> of values from the input domain

Solution: Input partitioning

▸ Domain for each input parameter is <u>partitioned into regions</u>

> ▸ The domain is substituted by an enumeration

Partitioning Domains

▸ <u>Domain</u> *D*

▸ <u>Partition scheme</u> *q* of *D*

▸ The partition *q* defines a <u>set of blocks</u>, $Bq = b_1, b_2, b_3, \ldots b_q$

● The partition must satisfy two properties:

1) blocks must be pairwise disjoint (no overlap)

2) together the blocks cover the domain D (complete)

$$bi \cap bj = \Phi, \forall\, i \neq j, bi, bj \in Bq$$

$$\bigcup_{b \in Bq} b = D$$

Using Partitions – Assumptions

▸ Choose a value from each partition

▸ Each value is assumed to be equally useful for testing

▸ Application to testing

 ▸ Find <u>characteristics</u> in the inputs : parameters, semantic descriptions, …
 ▸ <u>Partition</u> each characteristics
 ▸ <u>Choose tests</u> by combining values from characteristics

▸ Example <u>Characteristics</u>

 ▸ Input X is null -> true or false
 ▸ Order of the input file F -> sorted, inverse sorted, arbitrary
 ▸ Min separation of two aircraft -> integer 0 … 1000
 ▸ Input device -> DVD, CD, VCR, computer

Choosing Partitions

▸ Choosing (or defining) partitions seems easy, but is easy to get wrong

▸ Consider a file the contains word in some "order"

b1 = sorted in ascending order
b2 = sorted in descending order
b3 = arbitrary order

but … something's fishy …

What if the file is of length 1?

The file will be in all three blocks …
That is, disjointness is not satisfied

Solution:

Each characteristic should address just one property

b1 and b2

File F sorted ascending
  - b1 = true
  - b2 = false
File F sorted descending
  - b1 = true
  - b2 = false

Properties of Partitions

▸ If the partitions are not complete or disjoint, that means the partitions have not been considered carefully enough

▸ They should be reviewed carefully, like any design attempt

▸ Different alternatives should be considered

Example for program based testing

- Java

```
enum Color { RED, GREEN,BLU}

void foo(long x,  Color c, boolean value)
```

Color and boolean domain already partitioned. What about long domain?

Example of partition, from Boundary Value Analysis

- **MAX_VALUE**
  A constant holding the maximum value a long can have, 2^{63-1}.

- **MIN_VALUE**
  A constant holding the minimum value a long can have, -263.

- **BETWEEN MAX E MIN?**

Partition for long

Partitions in 5 subsets

MIN_VALUE < x < 0

0 < x < MAX_VALUE

MIN_VALUE

0

MAX_VALUE

Testing e verifica del SW 22/23
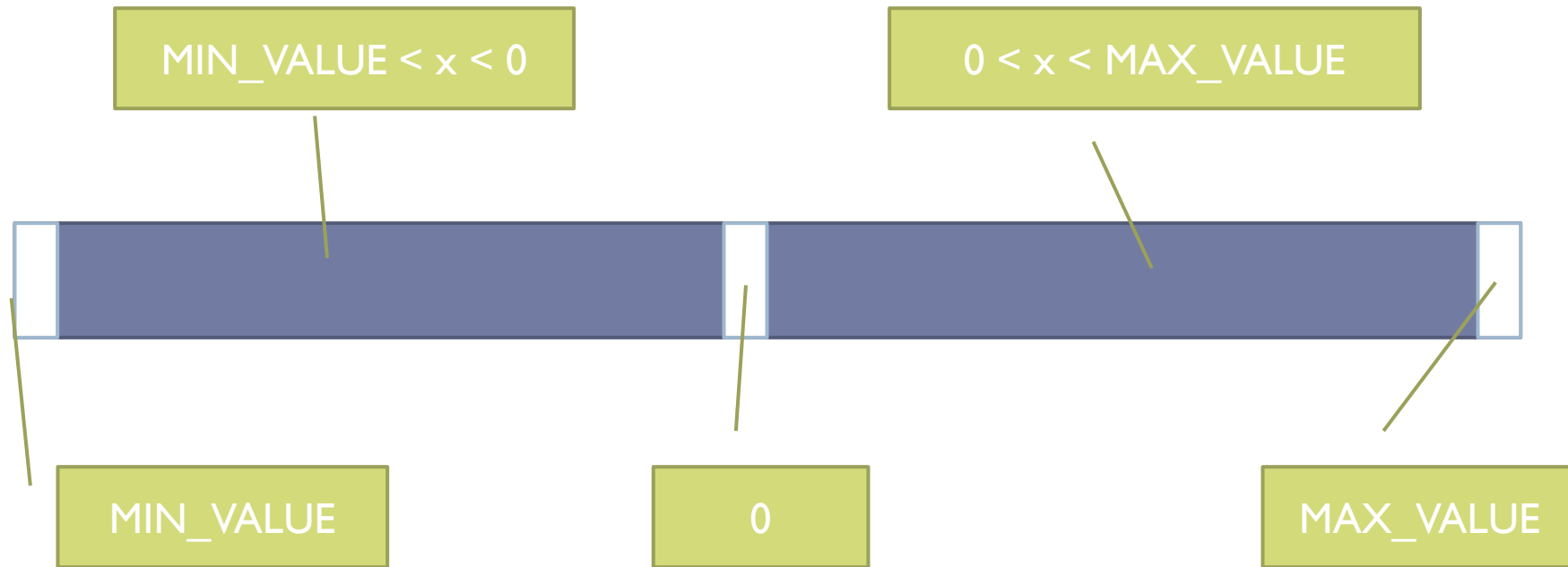
What to do when a Domain is a product of domains

▸ When our domain D = D1 x D2 x D3 …

▸ example: foo(int x, enum y, string s)
  D = int x enum x string

▸ Three possible choices:

  – Partition D as a whole

  – Partition Di and make the cartesian product

  – Partition Di and apply combinatorial testing

Partition of cartesian product

▸ Given two domains D1 and D2

▸ Let P1 a partition for D1 and P2 a partition for D2

▸ Partitions can be multiplied to obtain again partitions

▸ D1 x D2 can be partitioned by P1 X P2

▸ P1 x P2 will contain all the combinations of P1 and P2

D1	d11	(d11,d21)	..	..
	d12	...	..	..
	d13	...	..	..
		d21	d22	d23
		D2		

Product of partitions, application

▸ **For more than one input:**

```
/** given three sides return the type
    of the triangle*/
TriType Triang(int Side1,int Side2,int Side3)
```

▸ **If one splits every input in 5 subsets, the input is partitioned in 5 x 5 x 5 = 125 subsets ….**

Partition testing

- Several methods are based on partition testing [see books by Myers, and Beizer]:

1. Equivalent Partition

2. Domain Testing

3. Boundary Value Analysis

4. Category Partition [Ostrand Balcer 1988]:
   - Identify the parameters and variables and their choices
   - Generate all combinations (test frames)

Partition does not solve the problem!

- Category partition testing gave us
  - Systematic approach: Identify characteristics and values (the creative step),
  - generate combinations (the mechanical step)
- While equivalence partitioning offers a set of guidelines to design test cases, it suffers from two shortcomings:
  1. It raises the possibility of a large number of sub-domains in the partition.
     - Test suite size grows very rapidly with number of categories. Can we use a non-exhaustive approach?
  2. It lacks guidelines on how to select inputs from various sub-domains in the partition.
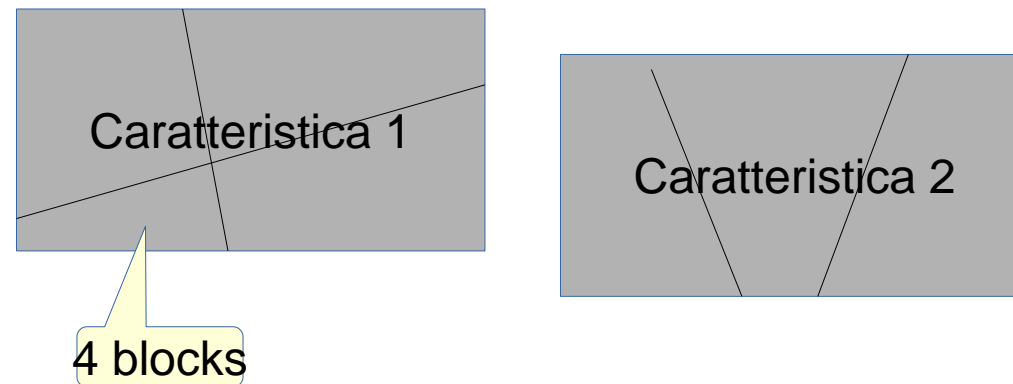
INPUT DOMAIN MODELLING

How to select the inputs and their values

A. Input Domain Modeling

- Step 1 : Identify testable functions
  - Individual methods have one testable function
  - In a class, each method has the same characteristics
  - Programs have more complicated characteristics—modeling documents such as UML use cases can be used to design characteristics
  - Systems of integrated hardware and software components can use devices, operating systems, hardware platforms, browsers, etc

- Step 2 : Find all the parameters
  - Often fairly straightforward, even mechanical
  - Important to be complete
  - Methods : Parameters and state (non-local) variables used
  - Components : Parameters to methods and state variables
  - System : All inputs, including files and databases

Modeling the Input Domain (cont)

▸ **Step 3** : Model the input domain

  ▸ The domain is scoped by the parameters

  ▸ The structure is defined in terms of characteristics

  ▸ Each characteristic is partitioned into sets of blocks

  ▸ Each block represents a set of values

  ▸ This is the most creative design step in applying ISP

Caratteristica 1

4 blocks

Caratteristica 2

STEP 3: Modeling the Input Domain

▸ Partitioning characteristics into blocks and values is a very <u>creative engineering</u> step

▸ More blocks means more tests

▸ The partitioning often flows directly from the definition of characteristics and both steps are sometimes done together

　▸ Should evaluate them separately – sometimes fewer characteristics can be used with more blocks and vice versa

▸ Strategies for identifying values :

　▸ Include valid, invalid and special values

　▸ Sub-partition some blocks

　▸ Explore boundaries of domains

　▸ Include values that represent "normal use"

　▸ Try to balance the number of blocks in each characteristic

　▸ Check for completeness and disjointness

Two Approaches to Input Domain Modeling (IDM)

I. Interface-based **approach**

   ▸ Develops characteristics directly from individual input parameters

   ▸ Simplest application

   ▸ Can be partially automated in some situations

I. Functionality-based **approach**

   ▸ Develops characteristics fro a behavioral view of the program under test

   ▸ Harder to develop—requires more design effort

   ▸ May result in better tests, or fewer tests that are as effective

1. Interface-Based Approach

▸ Mechanically consider each parameter in isolation

▸ This is an easy modeling technique and relies mostly on syntax

▸ Some domain and semantic information won't be used

    ▸ Could lead to an incomplete IDM

▸ Ignores relationships among parameters

Consider TriTyp

Three *int* parameters

IDM for each parameter is identical

Reasonable characteristic : *Relation of side with zero*

2. Functionality-Based Approach

‣ Identify characteristics that correspond to the intended functionality

‣ Requires more design effort from tester

‣ Can incorporate domain and semantic knowledge

‣ Can use relationships among parameters

‣ Modeling can be based on requirements, not implementation

‣ The same parameter may appear in multiple characteristics, so it's harder to translate values to test cases

Consider TriTyp again

The three parameters represent a *triangle*

IDM can combine all parameters

Reasonable characteristic : *Type of triangle*

Interface-Based IDM – TriTyp

- TriTyp, had one testable function and three integer inputs

First Characterization of TriTyp's Inputs			
Characteristic	b_1	b_2	b_3
q_1 = "Relation of Side 1 to 0"	greater than 0	equal to 0	less than 0
q_2 = "Relation of Side 2 to 0"	greater than 0	equal to 0	less than 0
q_3 = "Relation of Side 3 to 0"	greater than 0	equal to 0	less than 0

▸ A maximum of 3*3*3 = 27 tests

▸ Some triangles are valid, some are invalid

▸ Refining the characterization can lead to more tests …

Second Characterization of TriTyp's Inputs

Characteristic	b_1	b_2	b_3	b_4
q_1 = "Refinement of q_1"	greater than 1	equal to 1	equal to 0	less than 0
q_2 = "Refinement of q_2"	greater than 1	equal to 1	equal to 0	less than 0
q_3 = "Refinement of q_3"	greater than 1	equal to 1	equal to 0	less than 0

▸ A maximum of 4*4*4 = 64 tests

▸ This is only complete because the inputs are integers (0 .. 1)

Possible values for partition q1

Characteristic	b_1	b_2	b_3	b_4
Side1	2	1	0	-1

Test boundary conditions

Functionality-Based IDM – TriTyp

▸ First two characterizations are based on <u>syntax</u>–parameters and their type

▸ A <u>semantic</u> level characterization could use the fact that the three integers represent a triangle

Geometric Characterization of TriTyp's Inputs				
Characteristic	b_1	b_2	b_3	b_4
q_1 = "Geometric Classification"	scalene	isosceles	equilateral	invalid

• Oops … something's fishy … equilateral is also isosceles !
• We need to refine the example to make characteristics valid

Correct Geometric Characterization of TriTyp's Inputs				
Characteristic	b_1	b_2	b_3	b_4
q_1 = "Geometric Classification"	scalene	isosceles, not equilateral	equilateral	invalid

At the end of the first 3 steps

Set of parameters

- Each paramater has a list of possible values

    - Each value represent a block

- Each value can represent a partition

There may be constraints over the possible values

Combination Strategies criteria

▸ Step 4 : Apply a test criterion to choose combinations of values

    One parameter for each characteristic

    A test input has a value for each parameter

    Choosing all combinations is usually infeasible

    Coverage criteria allow subsets to be chosen

▸ Step 5 : Refine combinations of blocks into test inputs

    Choose appropriate values from each parameter

Choosing Combinations of Values

We assume to have partitioned the domain in characteristics and partitioned each characteristic in blocks

Step 4 – Choosing Combinations of Values

▸ Once parameters and values are defined, the next step is to <u>choose test values</u>

  – We use <u>criteria</u> – to choose <u>effective</u> subsets

▸ The most obvious criterion is to choose all combinations …

> <u>All Combinations (ACoC)</u> : All combinations of **values** from all **parameters** must be used.

- Number of tests is the product of the number of values in each parameter :

$$\prod_{i=i}^{Q} B_i$$

- The second characterization of TriTyp results in 4*4*4 = <u>64 tests</u> – too many ?

Example

- Parameter1: color : {Green, White, Red}

- parameter2: age [0...100] : blocks [0..6] [7..40][41..100]

- parameter3: isFemale : {true, false}

- ACoC all the combinations of values for each parameter
  - (not the combinations of all the values)

- Combination1: Green, 5,true ….

- If every parameter has only a value is like taking all the combinations of values.

ISP Criteria – Each Choice

▸ 64 tests for TriTyp is almost certainly way too many

▸ One criterion comes from the idea that we should try at least one value from each parameter

> Each Choice (EC) : Each value from each parameter be used in at least one test case.

• Number of tests is the number of values in the largest parameter

$$\max_{i=1}^{Q} \; size(B_i)$$

For TriTyp

▸ Three inputs side1, side2, side3

▸ Four values each 2, 1, 0, -1

▸ A test with 4 test is enough:

    2, 2, 2

    1, 1, 1

    0, 0, 0

    -1, -1, -1

Each Choice (EC)

- Characteristic1: color : {Green, White, Red}

- Characteristic2: age [0...100] : blocks [0..6] [7..40][41..100]

- Characteristic3: isFemale : {true, false}

- Each Choice (EC)
  - 3 Combination:
    Green, 5,true

    ...

- Is there something in the middle?

▸ AoC "Too many" ! - EC too few

▸ 34 switches = 2^{34} = 1.7×10^{10} possible inputs = 1.7×10^{10} tests ????

▸ Too much – some assumptions

- What if we knew that **one** single switch always causes the fault?
  2 tests would be enough to find if the system is correct:
  - all off, all on
  What if we knew no failure involves more than 2 **switch** settings interacting?

- We should test the interaction of 2 switches

Tra eachchoice e allcombinations

▸ Esempio introduttivo a combinatorial

    – ## Per comodità ogni blocco ha un valore

CLIENT	WEB SERVER	PAYMENT	DATABASE
FIREFOX	WEB SPHERE	MASTER CARD	DB/2
IE	APACHE	VISA	ORACLE
OPERA	.NET		

- All combinations…
- Each choice

- Se volessi testare tutti a coppia- coppia
  - Per vedere se l'interazione tra due può dare errore
  - Esempio FIREFOX su WEBSPHERE, ma anche FIRE

ISP Criteria – Pair-Wise

Another approach asks values to be combined with other values

Pair-Wise (PW) : Each value from each parameter must be combined with every value for each other parameter.

For TriTyp:

2, 2, 2	2, 1, 1	2, 0, 0	2, -1, -1
1, 2, 1	1, 1, 0	1, 0, -1	1, -1, 2
0, 2, 0	0, 1, -1	0, 0, 2	0, -1, 1
-1, 2, -1	-1, 1, 2	-1, 0, 1	-1, -1, 0

Example

- Characteristic1: color : {Green, White, Red}

- Characteristic2: age [0...100] : blocks [0..6] [7..40][41..100]

- Characteristic3: isFemale : {true, false}

- <u>PW. chose two characteristics: every combination of values of these two characteristics is covered</u>

- Color and age: every pair must be covered

- Age and isFemale must be covered

-

Combinatorial approach

- Pairwise combination instead of exhaustive

    - Generate combinations that efficiently cover all pairs of values

    - Rationale: most failures are triggered by single values or combinations of a few values. Covering pairs (triples,…) reduces the number of test cases, but reveals most faults

Example

Display Mode	Color	Screen size
full-graphics	Monochrome	Hand-held
Low resolution	16-bit	Laptop
text-only	True-color	Full-size

3 variables with 3 values each: 33 = 27 possible combinations

Combinatorial testing with much fewer tests

Test Suite - example

- **pairwise testing can be achieved by only 9 tests**

Test	Color	Display Mode	Screen Size
1	Monochrome	Full-graphics	Hand-held
2	16-bit	Text-only	Laptop
3	True-color	Full-graphics	Hand-held
4	…	…	…

One test covers many combinations:
e.g. Test 1 covers 3 pairs:
(**Monochrome, Full-graphics**)
(**Monochrome, Hand-held**)
(**Full-graphics, Hand-held**)

Other figures:
2^{100} combinations with 10 tests;
10^{20} -> 200 tests; …

ISP Criteria – Pair-Wise

Number of tests?

- Number of tests is at least the product of two largest characteristics

$$(\text{Max } _{i=1}^{Q} |B_i|) * (\text{Max } _{j=1, j!=i}^{Q} |B_j|)$$

ISP Criteria –T-Wise

A natural extension is to require combinations of t values instead of 2

> **t-Wise (TW)** : A value from each block for each group of t characteristics must be combined.

- Number of tests is at least the product of t largest characteristics
- If all characteristics are the same size, the formula is

$$\left(\text{Max}_{i=1}^{Q} \ \text{size}(B_i) \right)^{t}$$

- If t is the number of characteristics Q, then all combinations
- That is … *Q-wise = AC*
- *t*-wise is expensive and benefits are not clear

ISP Coverage Criteria Subsumption
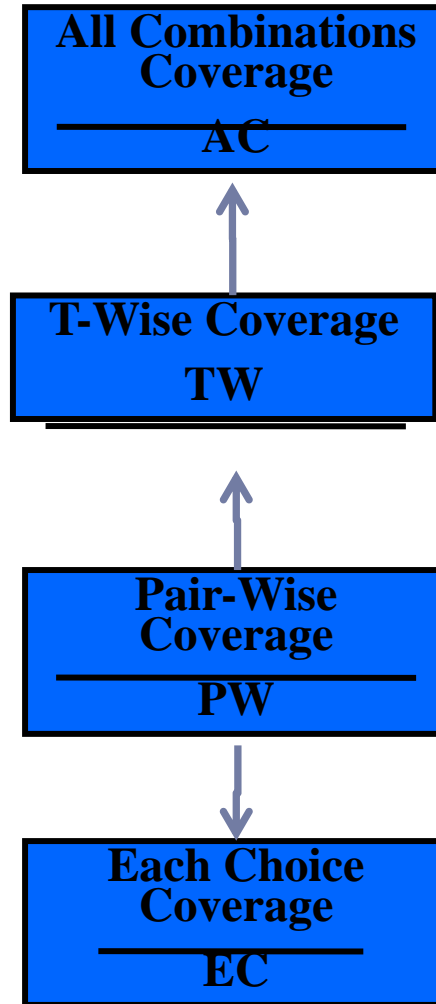
Exercise

example system: component based application

CLIENT	WEB SERVER	PAYMENT	DATABASE
FIREFOX	WEB SPHERE	MASTER CARD	DB/2
IE	APACHE	VISA	ORACLE
OPERA	.NET	AMEX	ACCESS

One test is missing ...

▸

▸ ▸ $3^4 = 81$ exhaustive
test cases

▸ 9 test cases

CLIENT	WEB SERVER	PAYMENT	DATABASE
FIREFOX	WEB SPHERE	MASTER CARD	DB/2
FIREFOX	.NET	AMEX	ORACLE
FIREFOX	APACHE	VISA	ACCESS
IE	WEB SPHERE	AMEX	ACCESS
IE	APACHE	MASTER CARD	ORACLE
IE	.NET	VISA	DB/2
OPERA	WEB SPHERE	VISA	ORACLE
OPERA	.NET	MASTER CARD	ACCESS
OPERA	APACHE	AMEX	DB/2

Altri criteri

Esistono anche altri criteri come il

Base Choice Coverage (BCC): A base choice block is chosen for each characteristic, and a base test is formed by using the base choice for each characteristic. Subsequent tests are chosen by holding all but one base choice constant and using each non-base choice in each other characteristic.

BCC prevede un test (base) + una variazione per ogni blocco di ogni partizione

- Esempio – triangolo base choice

- -1,0,1,2 per ogni lato, e base 2,2,2 ...

Estensione

Multiple base choice.

Multiple Base Choices (MBCC): At least one, and possibly more, base choice blocks are chosen for each characteristic, and base tests are formed by using each base choice for each characteristic at least once. Subsequent tests are chosen by holding all but one base choice constant for each base test and using each non-base choice in each other characteristic.

Più semplici da generare che i combinatorial test suite

A small simplification

- In case every characteristic is an input variable

- That takes the value in a domain and each block is a **single** value in this domain

- Example input1: {A,B,C}, input {LOW,MIDDLE,HIGH}

- Each choice: every possible combination of values is given to all the variables

- One choice: every variable takes at least once

- Pairwise: every pair of values is taken by at least one test

Generation techniques per combinatorial testing

Generation techniques families

Algebraic

- Based on some mathematical/algebraic properties

Search based, greedy, based on heuristics

- Because the problem of generating a minimum test suit for combinatorial testing is NP-complete, most methods and tools use a greedy approach

Logic based

- Based on SAT/SMT solving and model checking

Classification

Algebraic methods that are mainly developed by mathematicians

- Latin squares, Orthogonal arrays, Covering arrays

- Recursive Construction

Search-Based methods that are mainly developed by computer scientists

- AETG (from Telcordia), TCG (from JPL/NASA), DDA (from ASU), PairTest/Fireeye (from NIST)

- Incremental construction

Search-Based vs Algebraic Methods

Algebraic methods:

- Advantages: very fast, and often produces optimal results

- Disadvantages: limited applicability, difficult to support parameter relations and constraints

- E.g. most work only if all the parameters have the same domain size

Search-based methods:

- Advantages: no restrictions on the input model, and very flexible, e.g., relatively easier to support parameter relations and constraints

- Disadvantages: explicit search takes time, the resulting test sets are not optimal

Greedy methods

Greedy methods

Parameter based

- One column at the time

  - IPO

  - IPOS – still room to improve

Test case based

- Add one test at the time

  - AETG

IPO: In-Parameter-Order

Originally presented in:

Yu Lei, K. C. Tai, "In-Parameter-Order: A Test Generation Strategy for Pairwise Testing," High-Assurance Systems Engineering, IEEE International Symposium on, p. 254, Third IEEE International High-Assurance Systems Engineering Symposium, 1998
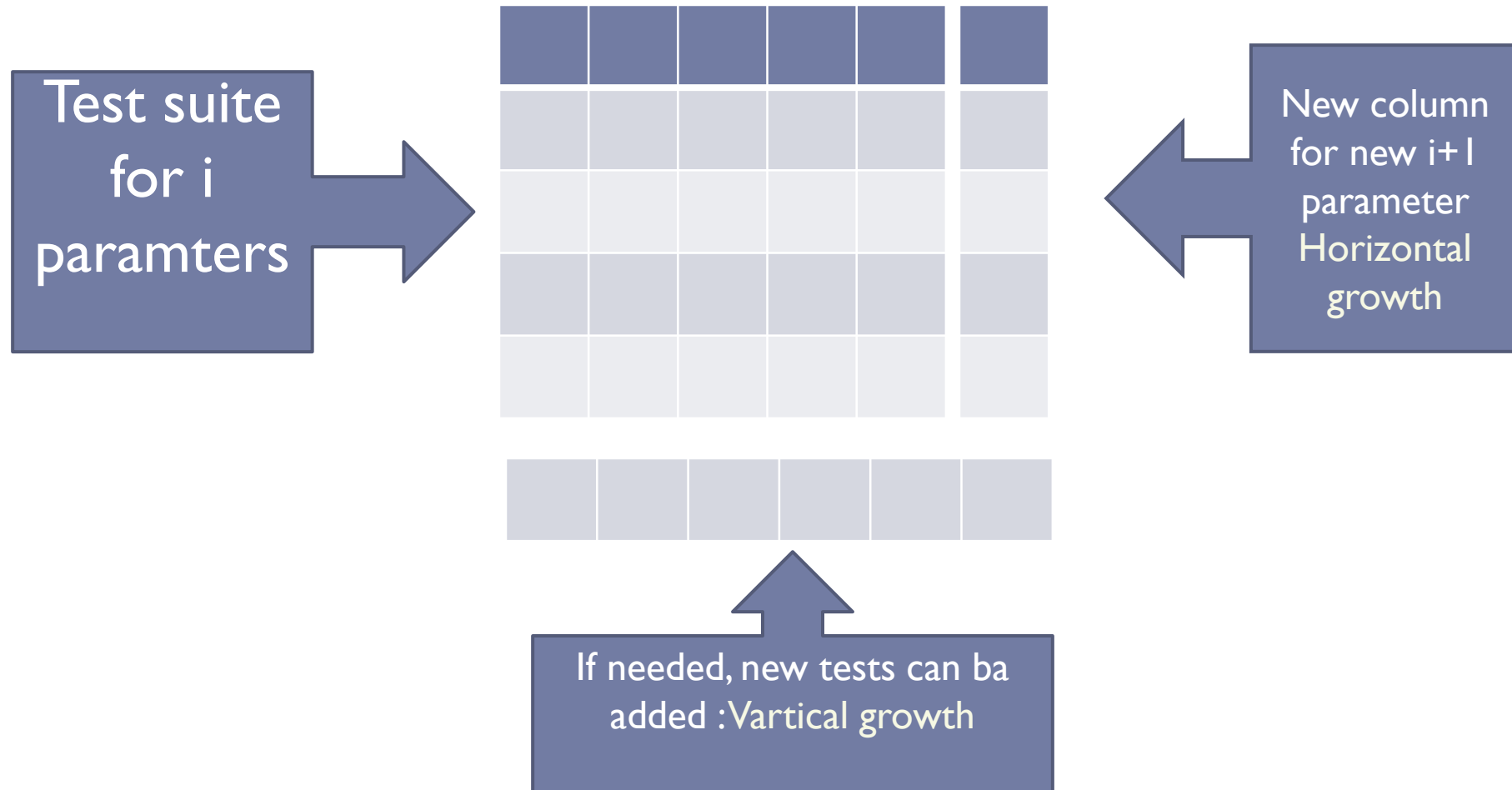
Several extensions

NIST

      http://csrc.nist.gov/groups/SNS/acts/

TOOL: FireEye, now ACTS

IPO: In-Parameter-Order

- Builds a t-way test set in an incremental manner

  - A t-way test set is first constructed for the first t parameters, simply considering their combinations

  - Then, the test set is extended to generate a t-way test set for the first t + 1 parameters

  - The test set is repeatedly extended for each additional parameter.

- Two steps involved in each extension for a new parameter:

  - Horizontal growth: extends each existing test by adding one value of the new parameter

  - Vertical growth: adds new tests, if necessary

Adding parameters

Test suite for i paramters →

New column for new i+1 parameter Horizontal growth ←

↑ **If needed, new tests can ba added :Vartical growth**

Strategy In-Parameter-Order

/* step 1: for the first t parameters p1, p2 , ..., pt*/

T := {(v1, v2, ..., vt) | v1, v2, ..., vt are values of p1, p2, ..., pt } // cartesian product

if n = t then stop;

/* step 2: for the remaining parameters */

for parameter pi, i = t + 1, ..., n do

  begin /* add parameter pi */

    /* 2a: horizontal growth */

    for each test (v1, v2, ..., vi-1) in T do

      replace it with (v1, v2, ..., vi-1, vi), where vi is a value of pi

    /* 2b: vertical growth */

    while T does not cover all the interactions between pi and

      each of p1, p2, ..., pi-1 do

      add a new test for p1, p2, ..., pi to T;

  end

Example

Consider a system with the following parameters and values:

– parameter A has values WIN and LIN

– parameter B has values INT and AMD, and

– parameter C has values IPV4, IPV6

Pairwise testing t = 2

Step 1: the first t parameters

if a test suite wants to cover all the t-combinations of t parameters, it must contain all the possible combinations

$t = 2$

parameter A has values WIN and LIN

parameter B has values 1NT and AMD

Initial test suite (CA):

A	B
WIN	INT
WIN	AMD
LIN	INT
LIN	AMD

Step 2: adding a new parameter

Add the tests to cover the t+1 th parameter.

Add a column to the CA for the new parameter

For the values of the new

<p style="color:red">Horizontal Growth</p>

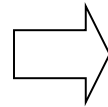| A	B
WIN	INT
WIN	AMD
LIN	INT
LIN	AMD

➡

A	B	C
WIN	INT	IPV4
WIN	AMD	IPV6
LIN	INT	IPV4
LIN	AMD	IPV6

Step 2 b

Check if all the tuples are covered, in case add new rows (vertical growth)

Vertical Growth

A	B	C
WIN	INT	IPV4
WIN	AMD	IPV6
LIN	INT	IPV4
LIN	AMD	IPV6

⟹

A	B	C
WIN	INT	IPV4
WIN	AMD	IPV6
LIN	INT	IPV4
LIN	AMD	IPV6
LIN	AMD	IPV4
LIN	INT	IPV6

missing :

AMD, IPV4

INT, IPV6

Exercise

parameter A has values A1 and A2
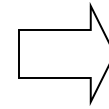
parameter B has values B1 and B2, and

parameter C has values C1, C2, and C3

Example (2)

A	B
A1	B1
A1	B2
A2	B1
A2	B2

➡

A	B	C
A1	B1	C1
A1	B2	C2
A2	B1	C3
A2	B2	C1

Horizontal Growth

➡
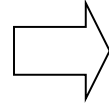
A	B	C
A1	B1	C1
A1	B2	C2
A2	B1	C3
A2	B2	C1
A2	B1	C2
A1	B2	C3

Vertical Growth

Open problems

When adding a new column, how to chose the values?

When adding a new row, how to choose the new row?

Combinatorial testing is effective

What is Combinatorial testing

▶ It can be classified "input space" testing or testing based on the **interfaces**

  ▶ Systematic testing of parameters to check the interaction
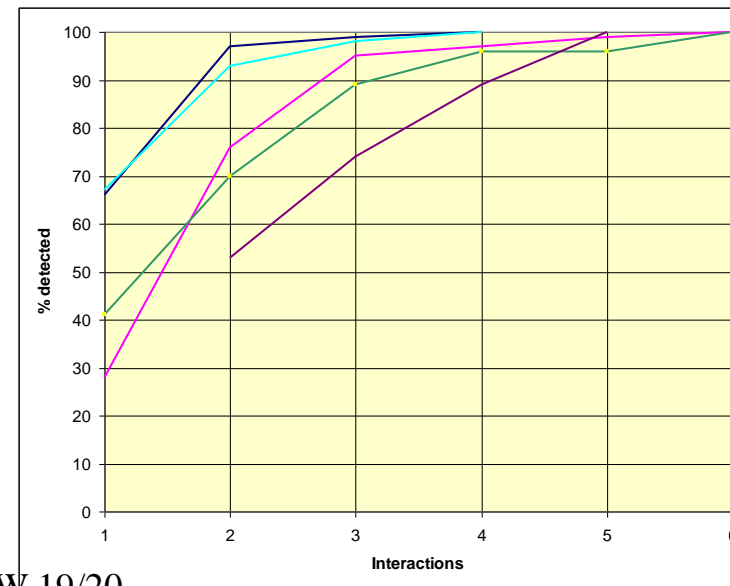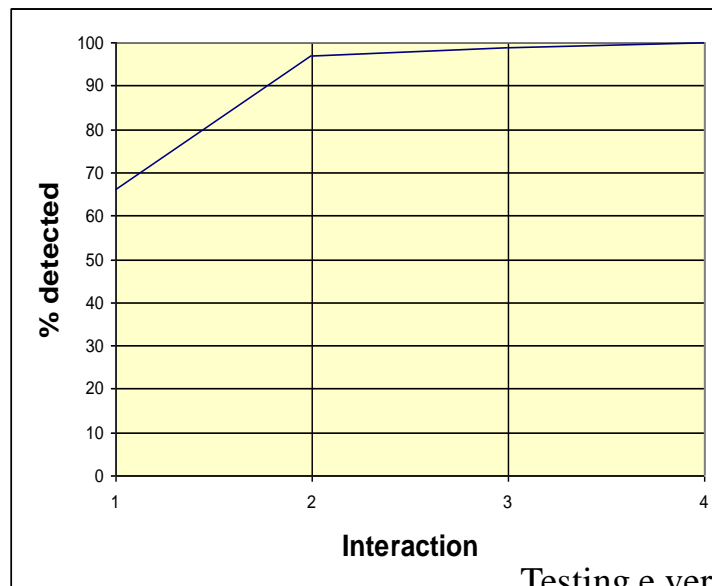
CIT effectiveness

▸ Experiments show that CIT is

▸ **effective**

  ▸ finds faults that traditional testing may be not able to find

▸ **efficient**

  ▸ A low degree of interaction between inputs can already discover most faults

    ▸ Pairwise is the most used

  ▸ Never with interaction > 6

Effectiveness -1

▸ *Compared to a traditional company that would use the quasi-exhaustive strategy, the combinatorial design method (CDM) strategy would reduce its system level test schedule by sixty-eight percent (68%) and save sixty-seven percent (67%) in labor costs associated with the testing.*

▸ Reference: Raytheon (2000). Jerry Huller. *Reducing Time to Market with Combinatorial Design Method Testing.*

Effectiveness -2 - Kuhn @ NIST

▸ Maximum interactions for fault triggering was 6

▸ Reasonable evidence that maximum interaction strength for fault triggering is relatively small

  ▸ % errors (seeded or found) vs interaction strength for several application:

Effectiveness - 3

▸ More experiments are needed

▸ New experiments are welcome!

Combinatorial testing is better than structural testing ?

Combinatorial testing is better than random testing ?

VINCOLI TRA I PARAMETRI

- Molte volte I parametri non sono indipendenti ma esistono dei vincoli

- Esempi

  - Giorno e mese

    - Se mese = FEB, giorno < 29 ….

Vincoli

CLIENT	SISTEMA OP.	ARCH
FIREFOX	WINDOWS	X86-32bit
IE	LINUX	X86-64bit
OPERA	MACOS	ARM

- Constraints:

  – CLIENT = IE => SISTEMAOP = WINDOWS

  – not (SO = MACOS and ARCH = ARM)

  – ….

  –