Partition and combinatorial testing

Angelo Gargantini –
Corso Testing e verifica del software

Outline

- What is combinatorial testing
  - Efficiency: It can detect faults
- Partition testing
  - A method to apply partition testing
  - How to choose variable values
- Combinatorial interaction of parameters
- Generation techniques
  - IPO, AETG, IPOS
- Adding constraints
  - Logic approach, using SAT/SMT solving

What is Combinatorial testing

- It can be classified “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 model based testing
  - Model of the inputs
- Program 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

Combinatorial testing is effective

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

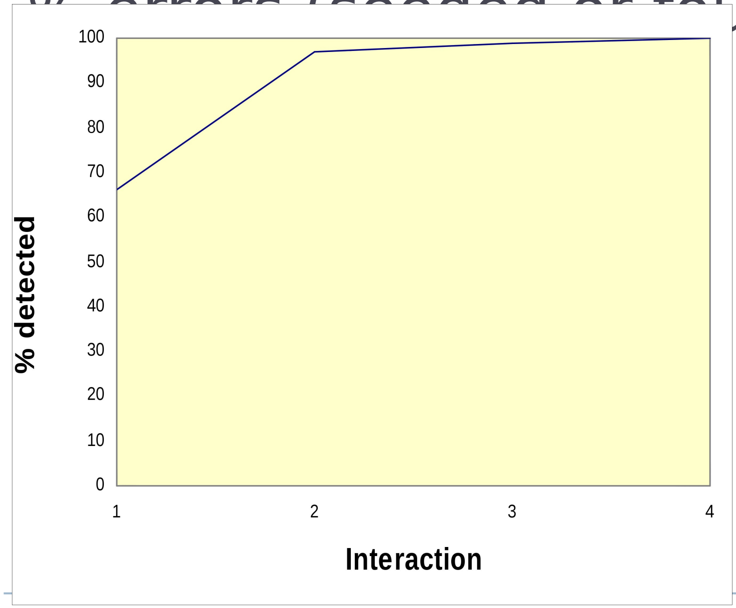
Effectiveness -1

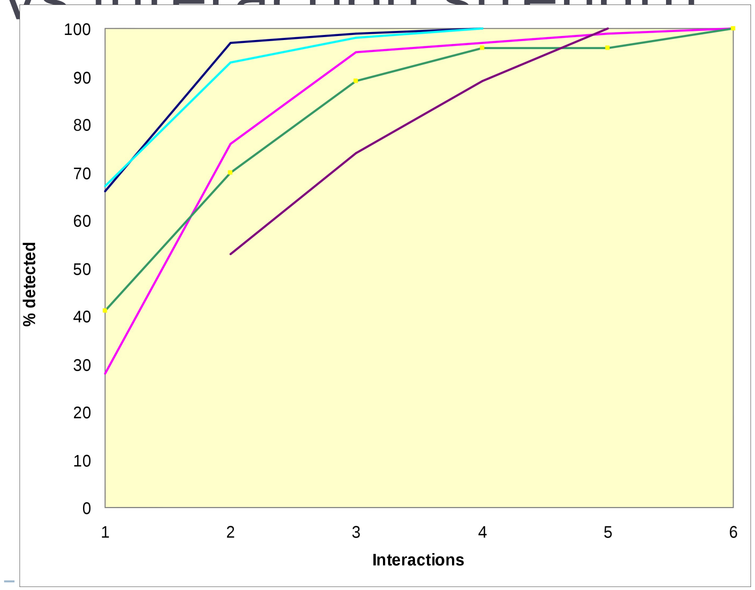
- *Compared to a traditional company that would use the quasi-exhaustive strategy, the Combinatorial design method (CDM) strategy would reduce its system level test schedule by sixty-eight percent (68%) and save sixty-seven percent (67%) in labor costs associated with the testing.*

- 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





Effectiveness - 3

- More experiments are needed
- New experiments are welcome!

Combinatorial testing is better than structural testing ?

Combinatorial testing is better than random testing ?

Partition testing

Problems …

- The input domain 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 infinite
- Testing is fundamentally about choosing finite sets of values from the input domain

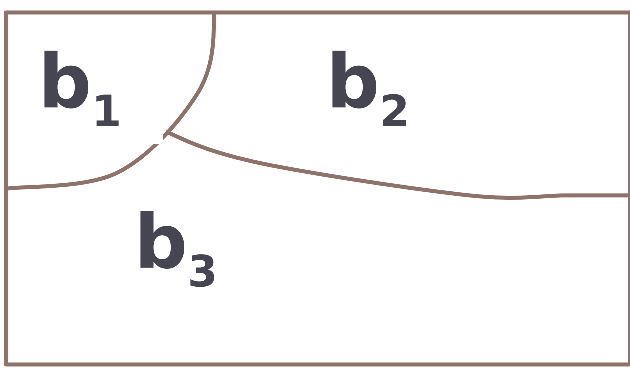
Solution: Input partitioning

- Domain for each input parameter is partitioned into regions
  - The domain is substituted by an enumeration

Partitioning Domains

- Domain *D*
- Partition scheme *q* of *D*
- The partition *q* defines a set of blocks, $Bq = b_{1}, b_{2}, \ldots b_{Q}$
- The partition must satisfy two properties :

1. blocks must be *pairwise disjoint* (no overlap)



domain

$b_i \cap b_j = \Phi, \forall i \neq j, b_i, b_j \in Bq$

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

Using Partitions – Assumptions

- Choose a value from each partition
- Each value is assumed to be equally useful for testing
- Application to testing
  - Find characteristics in the inputs : parameters, semantic descriptions, …
  - Partition each characteristics
  - Choose tests by combining values from characteristics
- Example Characteristics
  - 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"

b_1 = 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)
```

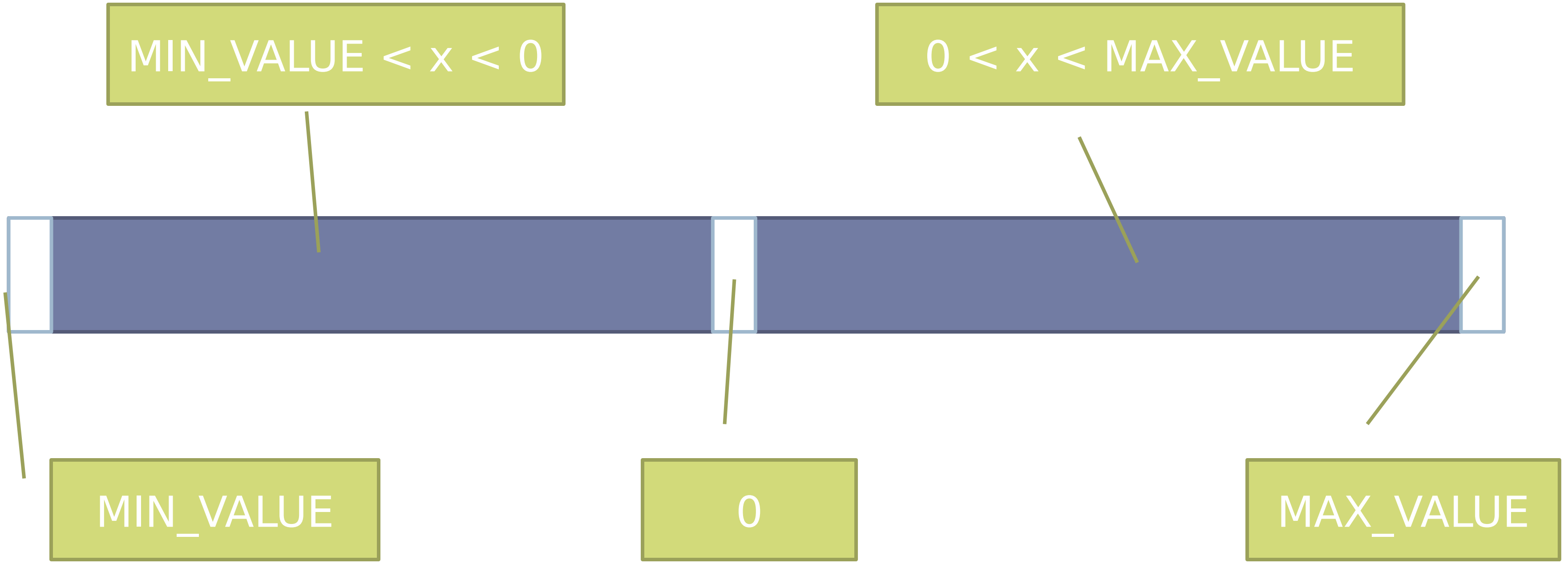
Color and boolean domain already partitioned. What about long domain?

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



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

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.

From Partition testing to combinatorial testing

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

STEP 3: Modeling the Input Domain

- Partitioning characteristics into blocks and values is a very creative engineering 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)

1. Interface-based approach
   - Develops characteristics directly from individual input parameters
   - Simplest application
   - Can be partially automated in some situations

2. 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

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 inte

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 syntax–parameters and their type
- A semantic level characterization could use the fact that the three i

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

Combination Strategies criteria

- Step 4 : Apply a test criterion to choose combinations of values
  - A test input has a value for each parameter
  - One block for each characteristic
  - 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 block

Choosing Combinations of Values

Step 4 – Choosing Combinations of Values

- Once characteristics and partitions are defined, the next step is to choose test values
- We use criteria – to choose effective subsets
- The most obvious criterion is to choose all combinations …

All Combinations (ACoC) : All combinations of blocks from all characteristics must be used.

- Number of tests is the product of the number of blocks in each characteristic :

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

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

- Example of “Too many” !

- 34 switches = 2^{34} = 1.7 x 1010 possible inputs = 1.7 x 1010 tests
????

National Institute of Standards and Technology

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 **3 switch** settings interacting?

  If only 3-way interactions, need only **33** tests
  For 4-way interactions, need only **85** tests

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 block

> Each Choice (EC) : One value from each block for each characteristic must be used in at least one test case.

- Number of tests is the number of blocks in the largest characteristic

$$\mathrm{Max}_{i=1}^{Q}(B_i)$$

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

ISP Criteria – Pair-Wise

- **Each choice yields few tests – cheap but perhaps ineffective**
- **Another approach asks values to be combined with other values**

Pair-Wise (PW) : A value from each block for each characteristic must be combined with a value from every block for each other characteristic.

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

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

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**

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
  - Extended by t-wise: test all the combinations of t values

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: 3^{3} = 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; 1020 200 tests; …

ISP Criteria –T-Wise

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

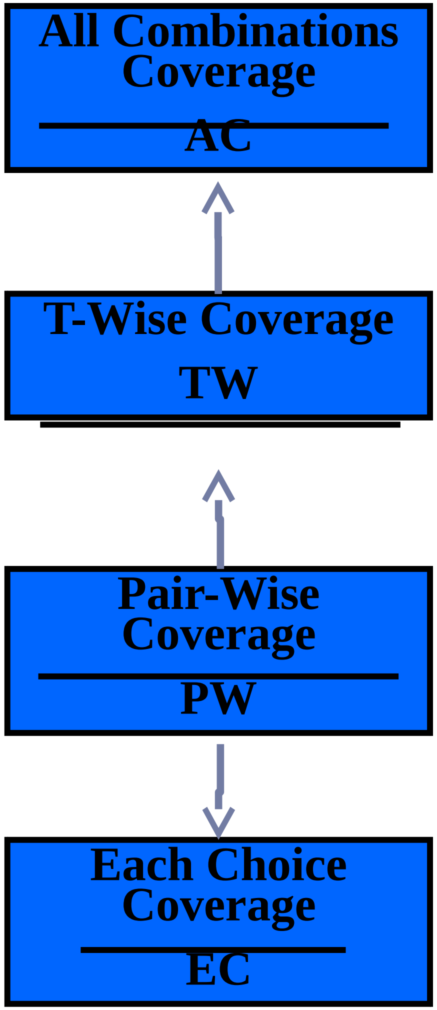
t-Wise (TW) : A value from each block for each group of t characteristics must be combined.

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

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

- 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
exhaustive test cases
9 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

Generation techniques

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 colum at the time
    - IPO
    - IPOS – still room to improve
- Test case based
  - Add one test at the time
    - AETG

IPO: In-Parameter-Order

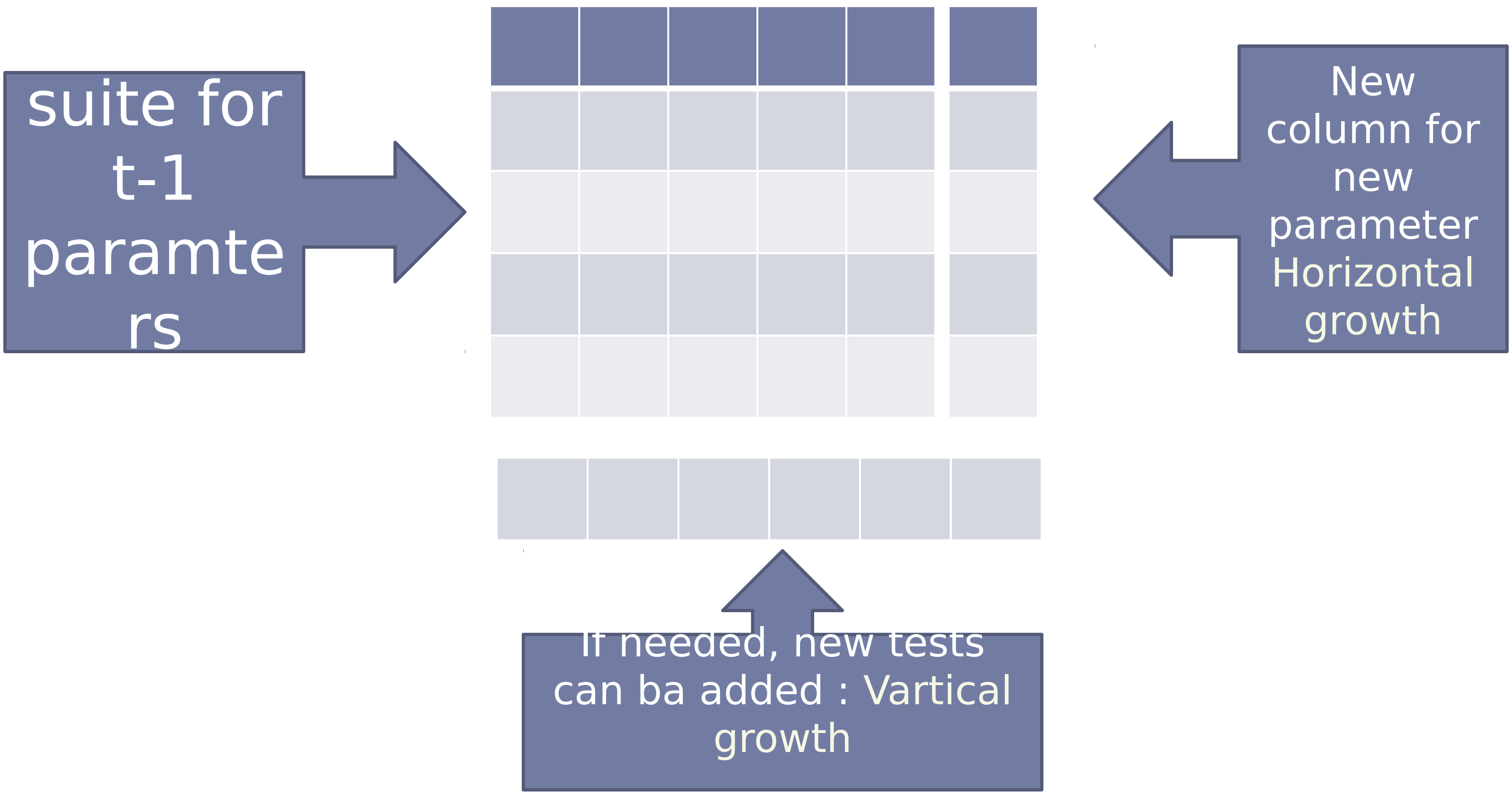
- Originally presented in:
- Yu Lei, K. C. Tai, "*In-Parameter-Order: A Test Generation Strategy for Pairwise Testing*," High-Assurance Systems Engineering, IEEE International Symposium on, p. 254, Third IEEE International High-Assurance Systems Engineering Symposium, 1998
- Several extensions
- NIST

  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
  1. A t-way test set is first constructed for the first t parameters, simply considering their combinations
  2. Then, the test set is extended to generate a t-way test set for the first t + 1 parameters
  3. 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



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 }

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;
```

Example

- Consider a system with the following parameters and values:
  - parameter A has values WIN and LIN
  - parameter B has values 1NT 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**

Horizontal Growth

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)**

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

missing :
AMD, IPV4
INT, IPV6

Vertical Growth

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