Sabancı Üniversitesi

Combinatorial Interaction Testing (CIT)

Cemal Yilmaz
Faculty of Engineering and Natural Sciences
Sabanci University
Istanbul, Turkey

Outline

Part I

State of the Art in CIT

Part II

Making CIT More Practical

Part III

Other Research Topics

PART I
State of the Art in CIT

A Motivating Example: MySQL

- MySQL characteristics
  - Database management system
  - Large user community – 6M+
  - Large code base – ≈1M
  - Geographically distributed developers
  - Continuous evolution – 200+ commits per month
- A highly configurable system
  - 100+ configuration options
  - Dozens of OS, compiler, and platform combinations

A Motivating Example: MySQL

- 100+ configuration options
- Assuming each option has two levels of settings
  - 2^{100+} configurations for testing

Which configurations should be tested?

- Assu o test
  - 2^{100+} secs. $\approx 10^{20+}$ centuries for exhaustive testing
  - Big Bang is estimated to be about 10^{7} centuries ago!
- Exhaustive testing is impossible!

Sabancı Üniversitesi

Configuration Space Model

- In its simplest form, the model includes
  - A set of configuration options $O=\{o_1, o_2, \ldots, o_n\}$
  - Their possible settings $V=\{V_1, V_2, \ldots, V_k\}$
- Implicitly defines the configuration space

```
[Options]
o1: {0, 1, 2}
o2: {0, 1, 2}
o3: {0, 1, 2}

[Constraints]
# none, for now!
```

Sabancı Üniversitesi

t-way Covering Arrays

- Given a model, a t-way covering array is a set of configurations, in which each possible combination of option settings for every combination of *t* options appears at least once
- *t:* coverage strength

o1	o2	o3
0	0	0
0	1	1
0	2	2
1	0	1
1	1	2
1	2	0
2	0	2
2	1	0
2	2	1

A 2-way covering array

Basic Justification

t-way covering arrays can efficiently exercise all system behaviors caused by the settings of *t* or fewer options

Example Covering Array Sizes

coverage strength (t)	# of binary options (n)	covering array size
3	12	15
3	26	22
3	68	31
3	256	46
4	12	24
4	26	51
4	67	67
4	256	160

Inter-Option Constraints

- Not all configurations may be valid in practice
- Inter-option constraints invalidate some option setting combinations
  - Hard constraints
  - Soft constraints [Bryce 06]

```
[Options]
o1, o2, o3: {0, 1, 2}
[Constraints]
o1=0 ➔ (o2≠0∧o3≠0)
```

o1	o2	o3
0	1	1
0	2	2
1	0	0
1	1	2
1	2	1
2	[illegible]	1
2	1	0
2	2	0
*	0	2
2	*	2

“Don’t Care”

A 2-way covering array

Example Uses of Inter-Option Constraints

- To avoid invalid configurations
  - Leads to better utilization of testing resources [Cohen 11]
- To avoid previously discovered failing sub-spaces
  - E.g., in adaptive testing scenarios [Dumlu and Yilmaz 11]
- To reduce actual cost of testing
  - E.g., in cost-aware covering arrays [Gulsen and Yilmaz 12]
- …

Seeds

- A set of *fully* or *partially* specified configurations
- The t-way combinations included in the seed are considered to be already covered
- A new set of configurations are generated only to cover the rest of the t-way combinations

A fully specified seed
A 2-way covering array

o1	o2	o3
0	0	0
0	1	1
0	2	2
1	0	1
1	1	2
1	2	0
2	0	2
2	1	0
2	2	1

Seeds

- A set of *fully* or *partially* specified configurations
- The t-way combinations included in the seed are considered to be already covered
- A new set of configurations are generated only to cover the rest of the t-way combinations

A 2-way covering array

A partially specified seed

o1	o2	o3
0	0	0
0	1	1
0	2	2
1	0	1
1	1	2
1	2	0
2	0	2
2	1	0
2	2	1

Example Uses of Seeds

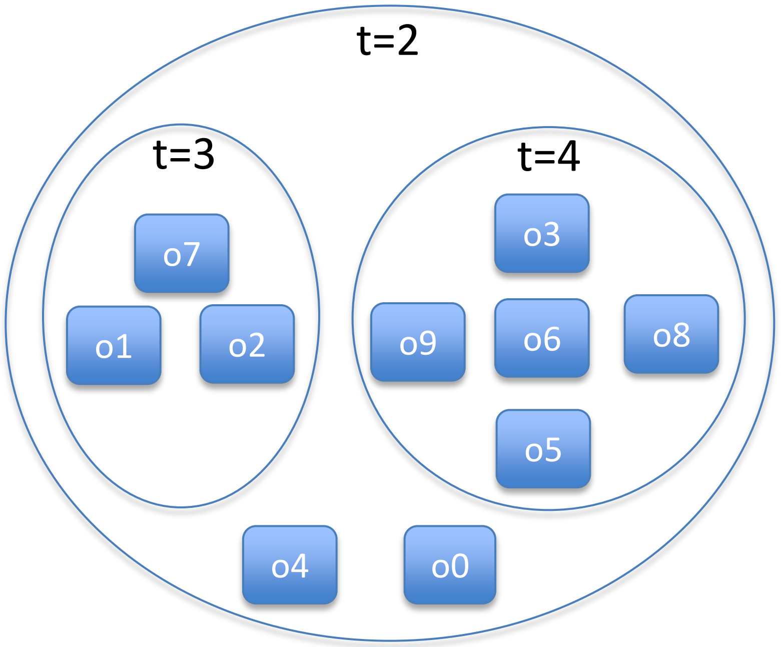
- To guarantee the inclusion of certain configs.
  - E.g., to test widely-used configurations
- To better utilize testing resources
  - E.g., some configs. may have already been tested [Dumlu and Yilmaz 11]
- For incremental covering array construction
  - E.g., using a low strength covering array as a seed to compute a higher strength covering array [Fouche 07]
- …

Sabancı Üniversitesi

Variable Strength Covering Arrays

- The order of interactions among configuration options may vary
- Variable strength covering arrays allow to vary strength *t* across different groups of options

  [Cohen 03]



Example Uses of Variable Strength Covering Arrays

- To more thoroughly test
  - interactions among closely interrelated options
  - critical interactions
  - frequently faced interactions
  - recently modified interactions
- …

Computing Covering Arrays

- Random search-based approaches
- Greedy approaches
  - [Sherwood 94, Cohen 97, Tung 00, Bryce 05, ...]
- Metaheuristic search-based approaches
  - Tabu search [Nurmela 2004]
  - Simulated annealing [Cohen 2003]
  - Genetic algorithms [Ghazi 2003]
  - Colony algorithms [Shiba 2004]
- Mathematical approaches
  - [Sherwood 94, Williams 00, Kobayashi 02, ...]

Applications to Software Testing

- Input parameter testing
  - [Mandal 85, …]
- Configuration testing
  - [Williams 01, Yilmaz 04, …]
- GUI testing
  - [Memon 05, …]
- Testing of software product lines
  - [Cohen 06, Oster 10, Perrouin 10, …]
- …

PART II
Making CIT More Practical

Revisiting the Basic Justification

t-way covering arrays can efficiently exercise all system behaviors caused by the settings of *t* or fewer options

However, we hypothesize that in practice many such behaviors are not actually tested due to *masking effects*

Sabancı Üniversitesi

Masking Effects

Emine Dumlu, M.S., 2011

- Prevent test cases from testing all the required option setting combinations present in configurations, which the test cases are normally expected to test
- Perturb the program executions in ways that prevent some option-related behaviors from being tested

E. Dumlu, C. Yilmaz, M. Cohen, and A. Porter, "Feedback driven adaptive combinatorial testing" *International Symposium on Software Testing and Analysis (ISSTA)*, pp. 243-253, July 2011.

Sabancı Üniversitesi

Masking Effects in Action - An Illustration -

o1	o2	o3	o4	t1
1	1	1	1	**F**
1	1	0	0	**F**
1	0	1	0	**F**
1	0	0	1	**F**
0	1	1	0	**P**
0	1	0	1	**P**
0	0	1	1	**P**
0	0	0	0	**P**

A 3-way covering array

- t1 failed when o1=1
- The highlighted 3-way combinations might not have been actually tested
- Since these combinations appear nowhere else in the covering array, they might not have been tested at all
- Fooling developers into thinking that they have tested certain combinations, when they in fact have not

Causes of Masking Effects

- System failures
- Unaccounted test case-specific inter-option constraints
- Unaccounted system-wide inter-option constraints
- Unaccounted control dependencies
- …

Test Case-Specific Constraints

- Traditional covering arrays assume that all test cases can run in all the selected configurations
- Our observations, however, say otherwise
- Test cases are likely to have some assumptions about the underlying configurations
- In a study conducted on MySQL and Apache, we observed that 337 of 738 MySQL test cases and 378 of 3789 Apache test cases had some test-case specific inter-option constraints

Test Case-Specific Constraints

- Why do they exist? Test cases of configurable systems often depend on some features that need to be explicitly configured into the system, such as
  - DAV feature of Apache
  - non-ANSI SQL feature of MySQL

- How do they affect testing? When a test case-specific constraint is not satisfied by a configuration, the test case refuses to run in the configuration
  - *i.e., skips* the configuration

Test Case-Aware CIT

- Traditional covering arrays do not account for test case-specific constraints
- Thus, they suffer from masking effects
  - If a test case skips a configuration, none of the option setting combinations appearing in the configuration will be tested by that test case
- To overcome this issue, we introduced a novel combinatorial object, called a *test case-aware covering array.*

Cemal Yilmaz, "Test-case aware combinatorial interaction testing" *IEEE Transactions on Software Engineering,* 39(5): 684-706, May 2013.

Example Scenario

- Configuration space model contains 4 binary options
  - *o1, o2, o3, o4 : {0, 1}*
- The system under test is to be tested with 3 test cases
  - *t1, t2, and t3*
- There is no system-wide constraints
- However, *t1* and *t2* have test case-specific constraints
  - *t1* skips when *o1=1*
  - *t2* skips when *o1=0*
- Whereas, *t3* has no test case-specific constraints
- For each test case, all valid 3-way option setting combinations are required to be tested

Sabancı Üniversitesi

A Traditional 3-way Covering Array

Traditional 3-way Covering Array

o1	o2	o3	o4	t1	t2	t3
1	1	1	1			
1	1	0	0			
1	0	1	0			
1	0	0	1			
0	1	1	0			
0	1	0	1			
0	0	1	1			
0	0	0	0			

11% of all valid 3-way option setting combination-test case pairs were not tested at all!

Configuration Model
[options] o1,o2,o3,o4: {0,1}
[sys-wide constraints]
none

Sabancı Üniversitesi

A 3-way Test Case-Aware Covering Array

Traditional 3-way Covering Array

o1	o2	o3	o4
1	1	1	1
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	1
0	0	0	0

Test Case-Aware 3-way Covering Array

o1	o2	o3	o4	tests	o1	o2	o3	o4	tests
0	1	1	1	{t1}	1	1	1	1	{t2, t3}
0	1	0	0	{t1}	1	1	0	0	{t2, t3}
0	0	1	0	{t1}	1	0	1	0	{t2, t3}
0	0	0	1	{t1}	1	0	0	1	{t2, t3}
0	1	1	0	{t1, t3}	1	1	1	0	{t2}
0	1	0	1	{t1, t3}	1	1	0	1	{t2}
0	0	1	1	{t1, t3}	1	0	1	1	{t2}
0	0	0	0	{t1, t3}	1	0	0	0	{t2}

Configuration Model
[options] o1,o2,o3,o4: {0,1}
[sys-wide constraints] none

Configuration Model
[options] o1,o2,o3,o4 : {0,1}
[sys-wide constraints] none
[tests] {t1, t2, t3}
[test constraints]
t1: o1 ≠ 1
t2: o1 ≠ 0

Test Case-Aware Covering Arrays

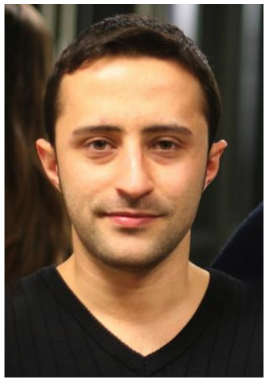
Test Case-Aware 3-way Covering Array

o1	o2	o3	o4	tests	o1	o2	o3	o4	tests
0	1	1	1	{t1}	1	1	1	1	{t2, t3}
0	1	0	0	{t1}	1	1	0	0	{t2, t3}
0	0	1	0	{t1}	1	0	1	0	{t2, t3}
0	0	0	1	{t1}	1	0	0	1	{t2, t3}
0	1	1	0	{t1, t3}	1	1	1	0	{t2}
0	1	0	1	{t1, t3}	1	1	0	1	{t2}
0	0	1	1	{t1, t3}	1	0	1	1	{t2}
0	0	0	0	{t1, t3}	1	0	0	0	{t2}

- No configuration violates the system-wide constraints
- No test case is scheduled to be executed in a configuration that violates its test case-specific constraint
- For each test case, every valid t-way combination of option settings appears at least once in the set of configurations, in which the test case is scheduled to be executed

Sabancı Üniversitesi

Computing Test Case-Aware Covering Arrays


Ugur Koc, M.S., 2014

- Identified a trade-off between the number of configurations and the number of test runs required by test case-aware covering arrays
- Introduced a number of algorithms to
  – "minimize" the number of test cases
  – "minimize" the number of configurations

1. Cemal Yilmaz, "Test-case aware combinatorial interaction testing" *IEEE Transactions on Software Engineering,* 39(5): 684-706, May 2013.
2. U. Koc and C. Yilmaz, "Using Simulated Annealing for Computing Test Case-Aware Covering Arrays", *submitted, 2014.*

Sabancı Universitesi

Feedback Driven Adaptive Combinatorial Interaction Testing

- Traditional CIT is static
- However, testing is dynamic and inherently unpredictable
- Demonstrated that traditional covering arrays significantly suffer from masking effects caused by system failures during testing
- Developed a feedback driven adaptive CIT approach
- Defined a novel coverage criterion, called *tested t-way interaction coverage*, that gives every test case a fair chance to test all required option setting combinations
- At each iteration
  1. Detect likely masking effects
  2. Isolate their causes
  3. Schedule the set of *t*-way combinations being masked for testing in the subsequent iteration
- Terminate when a full coverage under the coverage criterion is obtained

C. Yilmaz, E. Dumlu, M. Cohen, A. Porter, "Reducing Masking Effects in Combinatorial Interaction Testing: A Feedback Driven Adaptive Approach" *IEEE Transactions on Software Engineering,* 40(1): 43-66, Jan 2014.

…

PART III
Other Research Topics

…

Sabancı Üniversitesi

Combining Hardware and Software Instrumentation

Burcu Ozcelik, M.S., 2012

Pushing substantial parts of the data collection task onto hardware, so that internal program execution data can be collected at low costs for various types of data-driven program analyses

1. B. Ozcelik and C. Yilmaz, “Seer: A Lightweight Online Failure Prediction Approach”, *under revision, IEEE Transactions on Software Engineering, 2013.*
1. C. Yilmaz et. al, “Combining Hardware and Software Instrumentation to Classify Program Executions,” *International Symposium on Foundations of Software Engineering (FSE '10)*, pp. 67-76, 2010.

Concluding Remarks

- CIT is an efficient and effective way of revealing option-related failures
- There is still room for improvement, especially for making CIT more practical
  - Developing novel combinatorial objects
  - Developing adaptive CIT processes
  - Applying it to different domains

C. Yilmaz, S. Fouche, M. Cohen, A. Porter, G. Demiroz, and U. Koc, "Moving Forward with Combinatorial Interaction Testing",
IEEE Computer Magazine, 47(2): 37-45, Feb 2014.

Questions??