Systematic Software Testing Techniques: Combinatorial Testing

Dr. Renée Bryce
Associate Professor
University of North Texas
Renee.Bryce@unt.edu

Presentation outline

- Introductions
- Motivation
- Background on combinatorial testing
  - Exercise – Create a combinatorial test suite on paper
- Algorithms for combinatorial testing
  - Exercise – Download and use the ACTS tool
- Prioritized Combinatorial Testing

Introductions

- Briefly share your experiences with Software Testing

Motivation

- Costs of software defects
  - Software defects cost ~$59 billion per year [1]
- One contributor to software defects
  - Many system components are tested individually, but often unexpected interactions between components cause failures.

[1] National Institute of Standards and Technology, The Economic Impacts of Inadequate Infrastructure for Software Testing, U.S. Department of Commerce, May 2002.

Combinatorial Test Example

Hardware	Operating System	Network Connection	Memory
PC	Windows XP	Dial-up	64MB
Laptop	Linux	DSL	128MB
PDA	FreeBSD	Cable	256MB

Four factors (components) have three levels (options) each

Sample test

Test No.	Hardware	Operating System	Network Connection	Memory
1	PC	Windows XP	Dial-up	64MB

Pairs covered

1. (PC, Windows XP)	4. (Windows XP, Dial-up)
2. (PC, Dial-up)	5. (Windows XP, 64MB)
3. (PC, 64MB)	6. (Dial-up, 64MB)

Combinatorial Test Example

Hardware	Operating System	Network Connection	Memory
PC	Windows XP	Dial-up	64MB
Laptop	Linux	DSL	128MB
PDA	FreeBSD	Cable	256MB

Four factors (components) have three levels (options) each

Test No.	Hardware	Operating System	Network Connection	Memory
1	PC	Windows XP	Dial-up	64MB
2	Laptop	FreeBSD	DSL	64MB
3	PDA	Linux	Cable	64MB
4	Laptop	Windows XP	Cable	128MB
5	Laptop	Linux	Dial-up	256MB
6	PC	Linux	DSL	128MB
7	PDA	Windows XP	DSL	256MB
8	PC	FreeBSD	Cable	256MB
9	PDA	FreeBSD	Dial-up	128MB

Exercise 1

1. List all of the 2way combinations (pairs) for this input:

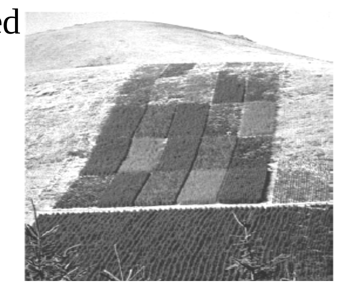
f_0	f_1	f_2	f_3
0	3	6	9
1	4	7	10
2	5	8	11

Hint: Example pairs are (0,3) (0,4) (0,5)…. (8,11)

2. Create a combinatorial test suite for the input above. No credit will be given for an exhaustive test suite.

Brief background

- Combinatorial testing has been used in several fields:
  - Agriculture
  - Combinatorial chemistry
  - Genomics
  - Software/hardware testing



- Study of Mozilla web browser found 70% of defects with 2-way coverage; ~90% with 3-way; and 95% with 4-way. [Kuhn *et. al.,* 2002]
- Combinatorial testing of 109 software-controlled medical devices recalled by US FDA uncovered 97% of flaws with 2-way coverage; and only 3 required higher than 2. [Kuhn *et. al.,* 2004]

Covering arrays

A *covering array*, $CA_{\lambda}(N;t,k,v)$, is an N x *k* array. In every N x *t* subarray, each *t*-tuple occurs at least λ times. In our application, *t* is the *strength* of the coverage of interactions, *k* is the number of components (factors), and *v* is the number of options for each component (levels). In all of our discussions, we treat only the case when λ = 1, (i.e. that every *t*-tuple must be covered at least once).

- Behind the scenes this combinatorial object is constructed to represent interaction test suites
- No efficient exact method is known
- Mathematicians and Computer Scientists have offered solutions from different view points. Their solutions have been measured by *time to generate test suites* and *sizes of test suites.*

Perceived benefits of greedy algorithms

	Mathematical	Greedy	Search
Size of test suites	**Accurate on special cases; but not as general as needed**	**Reasonably accurate**	**Most accurate (if given enough time)**
Time to generate tests	**Yes**	**Yes**	**Often time consuming (for good results)**
Seeding/ Constraints	**Difficult to accommodate seeds/constraints**	**Yes**	**Yes**

History of One-test-at-a-time Greedy Algorithms

AETG

- Pros
  - First tool to generate test suites (based on covering arrays)
- Cons
  - Produces different test suites to the same inputs
  - Slow

TCG

- Pros
  - Deterministic
  - Particularly good for mixed-level inputs
  - Faster
- Cons
  - Overly large test suites for fixed-level inputs

DDA

- Pros
  - Deterministic
  - Competitive results
  - Logarithmic guarantee on the size of test suites

Sample sizes of test suites

Parameters	DDA	AETG	TCG
$5^1 3^8 2^2$	21	19	20
$7^1 6^1 5^1 4^5 3^8 2^3$	43	45	45
$5^1 4^4 3^{11} 2^5$	27	30	30
$6^1 5^1 4^6 3^8 2^3$	34	34	33
$4^{15} 3^{17} 2^{29}$	35	41	*35*
$4^1 3^{39} 2^{35}$	27	28	*27*
3^{13}	18	15	*20*
2^{100}	15	10	*16*
4^{40}	43	*42*	*46*
4^{100}	51	*51*	*55*

[1] R. Bryce, C.J. Colbourn. A Density-Based Greedy Algorithm for Higher Strength Covering Arrays, *Journal of Software Testing, Verification, and Reliability*, (March 2009), 19(1):37-53.

[2] R. Bryce, C.J. Colbourn. The Density Algorithm for Pairwise Interaction Testing, *Journal of Software Testing, Verification and Reliability*, (August 2007), 17(3): 159-182.
**(Citeseer impact ranking of STVR: .36)*

Framework of One-row-at-a-time Greedy Methods

- Defines commonalities that all “one-row-at-a-time” greedy algorithms have in common
- A process provides statistical feedback on the impact of different decisions that can be made in the framework
- Experiments explore several thousand instantiations of the framework and provide a requisite of knowledge

Framework for greedy methods

Framework

- **Layer one: test suite repetitions**
- **Layer two: multiple candidates**
- **Layer three: factor ordering***
- **Layer four: level selection***

Pairs left to cover:
48
42
36

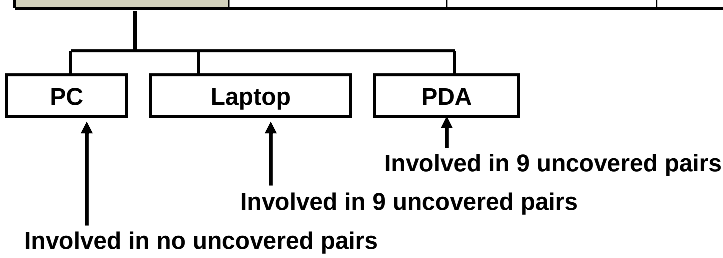
Test No.	Hardware	Operating System	Network Connection	Memory
1	PC	Windows XP	Dial-up	64MB
2	PC	Linux	DSL	128MB
3	PC	FreeBSD	Cable	256MB
4	?	?	?	?

Framework for greedy methods

Framework

- Layer one: test suite repetitions
- Layer two: multiple candidates
- **Layer three: factor ordering***
- **Layer four: level selection***

Factor 1	?	?	?

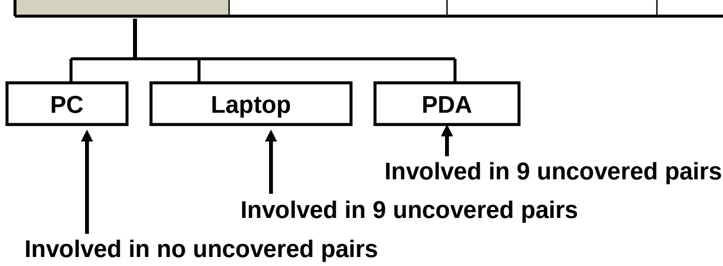


Framework for greedy methods

Framework

- Layer one: test suite repetitions
- Layer two: multiple candidates
- **Layer three: factor ordering***
- **Layer four: level selection***

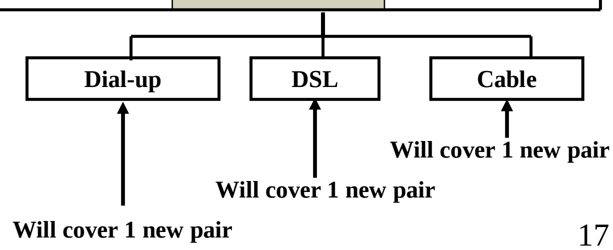
Laptop	?	?	?



Framework for greedy methods

Framework
- Layer one: test suite repetitions
- Layer two: multiple candidates
- **Layer three: factor ordering***
- **Layer four: level selection***

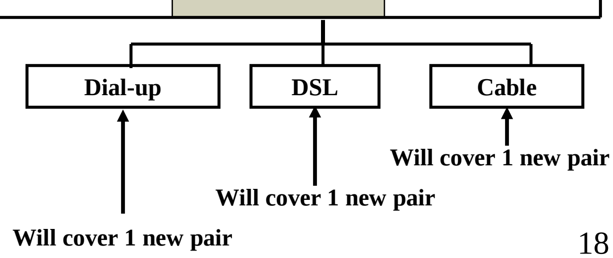
Laptop	?	Factor 3	?



Framework for greedy methods

Framework
- Layer one: test suite repetitions
- Layer two: multiple candidates
- **Layer three: factor ordering***
- **Layer four: level selection***

Laptop	?	DSL	?

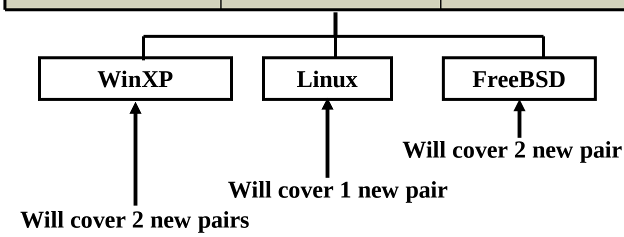


Framework for greedy methods

Framework

- Layer one: test suite repetitions
- Layer two: multiple candidates
- **Layer three: factor ordering***
- **Layer four: level selection***

Laptop	Factor 2	DSL	?

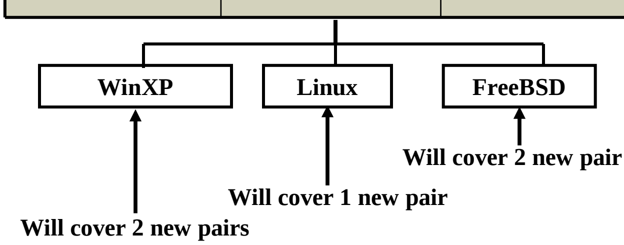


Framework for greedy methods

Framework

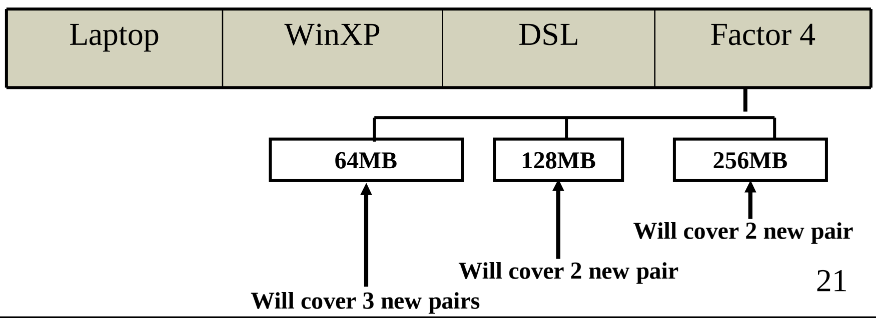
- Layer one: test suite repetitions
- Layer two: multiple candidates
- **Layer three: factor ordering***
- **Layer four: level selection***

Laptop	WinXP	DSL	?



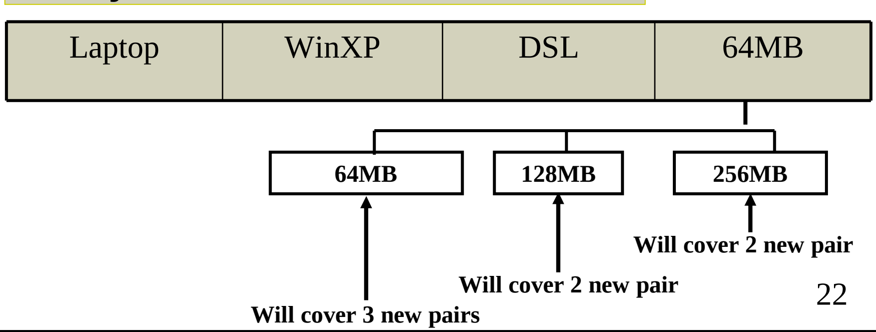
Framework for greedy methods

Framework
- Layer one: test suite repetitions
- Layer two: multiple candidates
- **Layer three: factor ordering***
- **Layer four: level selection***



Framework for greedy methods

Framework
- Layer one: test suite repetitions
- Layer two: multiple candidates
- **Layer three: factor ordering***
- **Layer four: level selection***



Framework for greedy methods

Framework

- Layer one: test suite repetitions
- **Layer two: multiple candidates***
- Layer three: factor ordering
- Layer four: level selection

3 candidate rows				Number of newly covered pairs
PDA	Linux	Cable	128MB	6
PDA	Win XP	Cable	64MB	5
PDA	Linux	Cable	128MB	5

Framework for greedy methods

Framework

- **Layer one: test suite repetitions***
- Layer two: multiple candidates
- Layer three: factor ordering
- Layer four: level selection

Test suite A

Test suite B

Test suite C

Select the smallest test suite generated.

Framework experiment - ANOVA results for several inputs

Input:	$10^1 9^1 8^1 7^1 6^1 5^1 4^1 3^1 2^1 1^1$	$8^2 7^2 6^2 5^2$	$6^6 5^5 3^4$	3^4	6^4	3^{40}
Layer 1 (Reps)	-	1.0645	1.157	5.698	4.797	10.466
Layer 2 (Candidates)	-	1.0656	-	-	-	-
Layer 3 (Factor Ordering)	73.176	70.8598	69.431	10.723	2.472	28.836
Layer 3 Tie-break 1	-	-	-	2.16	-	1.288
Layer 3 Tie-break 2	-	-	-	-	-	-
Layer 4 Tie-break 1	4.526	1.0975	4.148	33.646	21.267	3.433
Layer 4 Tie-break 2	-	-	-	-	-	-
Interaction of Layer 1 and Layer 2	-	1.4686	-	-	1.956	-
Interaction of Layer 2 and Layer 3	1.827	2.2132	1.061	1.45	-	-
Interaction of Layer 3 and Layer 3 Tie-break 1	-	-	-	3.435	1.29	2.866
Interaction of Layer 3 and Layer 4 Tie-break 1	3.535	2.1676	6.285	3.157	18.161	3.604
Lack Of Fit	13.649	17.7303	14.41	35.786	44.196	44.774
**Values that contribute <1% are not reported*						

[1] R.Bryce, C.J. Colbourn, M.B. Cohen. *A Framework of Greedy Methods for Constructing Interaction Tests.* The 27th International Conference on Software Engineering (ICSE), St. Louis, Missouri. (May 2005), pp. 146-155. *(13% acceptance rate)*

**Citeseer impact ranking of ICSE: 2.05*

ACTS – Free Download

- ACTS is a free tool to generate combinatorial test suites
- Download at: http://csrc.nist.gov/groups/SNS/acts/download/
- login ID is 'fireeye',
- password 'acts71362'
- Create a test suite for this input with 2way coverage and then generate a 2nd test suite with 3way coverage:

f_0	f_1	f_2	f_3
0	3	6	9
1	4	7	10
2	5	8	11

Prioritized combinatorial testing

- What if parts of a system are more important to test earlier?
- What if a tester learns during testing and wants to regenerate a test suite with new priorities?
- What if a tester has time to run pair-wise coverage and time to run some three-way tests?

A variation of the covering array

An ℓ-*biased covering array* is a covering array *CA*(*N*; 2*, k, v*) in which the first rows form tests whose total benefit is as large as possible. That is, no *CA*(*N*; 2*, k, v*) has rows that provide larger total benefit.

Input

Factor	v_0	v_1	v_2	v_3
f_0	0 (.2)	1 (.1)	2 (.1)	3 (.1)
f_1	4 (.2)	5 (.3)	6 (.3)	-
f_2	7 (.1)	8 (.9)	-	-

3 factors with varying numbers of associated levels (options) and weights assigned to each level

Algorithm Walkthrough

Input

Factor	v_0	v_1	v_2	v_3
f_0	0 (.2)	1 (.1)	2 (.1)	3 (.1)
f_1	4 (.2)	5 (.3)	6 (.3)	-
f_2	7 (.1)	8 (.9)	-	-

3 factors with varying numbers of associated levels (options) and weights assigned to each level

Larger weight means higher priority should be given to testing earlier!

Step 1 – Calculate Factor Interaction Weights.

Factor Interaction Weight	f_0	f_1	f_2	Total Weight
f_0	-	.4	.5	.9
f_1	.4	-	.8	1.2
f_2	.5	.8	-	1.3

Factors will be assigned values in order of "highest priority"

Algorithm Walkthrough

- ☑ Input has been processed
- ☑ Factor interaction weights have been calculated (to determine order to assign level values to factors).

Input

Factor	v_0	v_1	v_2	v_3
f_0	0 (.2)	1 (.1)	2 (.1)	3 (.1)
f_1	4 (.2)	5 (.3)	6 (.3)	-
f_2	7 (.1)	8 (.9)	-	-

Step 2 – Calculate Factor-Level Interaction Weights to select the level that covers the most uncovered *weighted density*.

For a factor, i, and a level, ℓ, that number of levels for a factor is called $v_{\max}$ **, and the factor interaction weight is called** $w_{\max}$

Factor-Level Interaction Weight = $\sum_{j=1}^{v_{\max}} \left(\frac{w_{f_{i_j}} * w_{\lambda}}{w_{\max}} \right)$

$$f2_{v_0} = (.05/1.3) + (.08/1.3) = .1$$

$$f2_{v_1} = (.45/1.3) + (.72/1.3) = .9$$

Remember, the factor interaction weight was 1.3 for factor 2.

Algorithm Walkthrough

For a factor, i, and a level, ℓ, that number of levels for a factor is called $v_{\max}$ **,**
and the factor interaction weight is called $w_{\max}$

Factor-Level Interaction Weight = $\sum_{j=1}^{v_{\max}} \left(\frac{w_{f_{ij}} * w_{\lambda}}{w_{\max}} \right)$

- Step 2 (continued) - Calculate Factor-Level Interaction Weights to select the level that covers the most uncovered *weighted density*.

$$\underbrace{\begin{aligned} f2_{v_0} &= (.05/1.3)+(.08/1.3)=.1 \\ f2_{v_1} &= (.45/1.3)+(.72/1.3)=.9 \end{aligned}}$$

Weight between two levels divided by max factor-interaction weight

$$\underbrace{\begin{aligned} f1_{v_0} &= (.1/1.3)+(.2/.9)=.2569 \\ f1_{v_1} &= (.15/1.3)+(.3*.9)=.38538 \\ f1_{v2} &= (.15/1.3)+(.3*.9)=.38538 \end{aligned}}$$

Weight between two levels divided by max factor-interaction weight + Weight of level multiplied by weight of level (of fixed factor)

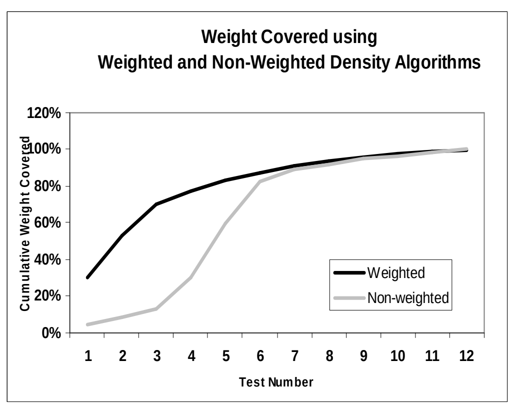
$$\underbrace{\begin{aligned} f0_{v_0} &= (.2*.3)+(.2*.9)=.24 \\ f0_{v_1} &= (.1*.3)+(.1*.9)=.12 \\ f0_{v2} &= (.1*.3)+(.1*.9)=.12 \\ f0_{v3} &= (.1*.3)+(.1*.9)=.12 \end{aligned}}$$

Weights of levels between each multiplied

Sample Results

Input

Factor	v_0	v_1	v_2	v_3
f_0	0 (.2)	1 (.1)	2 (.1)	3 (.1)
f_1	4 (.2)	5 (.3)	6 (.3)	-
f_2	7 (.1)	8 (.9)	-	-



Output

Row Number	f_0	f_1	f_2
1	0	5	8
2	1	6	8
3	2	4	8
4	3	5	8
5	0	6	7
6	0	4	7
7	1	5	7
8	2	5	7
9	3	6	7
10	2	6	7
11	1	4	7
12	3	4	7

[1] R. Bryce, C.J. Colbourn. Prioritized Interaction Testing for Pairwise Coverage with Seeding and Avoids, *Information and Software Technology Journal* (IST, Elsevier), (October 2006), 48(10):960-970.

** Citeseer impact ranking of ICST: .19*

Ongoing and future work

- Ultimate goal: to develop systematic testing methodologies that are widely used to improve software quality
- Reaching this goal:
  - Testing methodology driven by practical concerns of testers
  - Algorithms (for testing tools)
  - Empirical studies