Pairwise Testing
(aka “All Pairs” Testing)

Combinatorial Explosion

- If there are N different **attributes**, with value set sizes S_1, S_2, …, S_N, then the number of combinations is the product:

$$S_1 \, S_2 \, \ldots \, S_N$$

- Assuming $S_i \geq 2$ for each i, we have a number of combinations exponential in N.

Example: Web media serving

- 5 different browsers {IE, Firefox, Netscape, Opera, Safari} (but some have multiple versions
- 2 different media players {MediaPlayer, Real}
- 3 different webservers {IIS, Apache, Weblogic}
- 3 different operating systems {Widows, Apple, Linux}
- At least 5 x 2 x 3 x 3 = 80 combinations (not considering versions)

Pairwise Testing

- Pairwise testing is a **heuristic** for achieving good test coverage *without* trying all combinations:

- Select a set of combinations such that:
  - For any **pair** of attributes,
    there is a combination that includes that pair.

Example

- Combinations = {1, 2} x {3, 4}
- Every combination is needed

1	3
1	4
2	3
2	4

Example

- Combinations = {1, 2} x {3, 4} x {5, 6}
- 4 of 8 possible suffice

1	3	5
1	4	6
2	3	6
2	4	5

Does this alternate work?

- Combinations = {1, 2} x {3, 4} x {5, 6}

1	3	5
1	4	5
2	3	6
2	4	6

Is this heuristic effective?

- **Examples from: http://www.pairwise.org/results.asp**

> [...] a set of 29 pair-wise AETG tests gave 90% block coverage for the UNIX sort command. We also compared pair-wise testing with random input testing and found that pair-wise testing gave better coverage.
>
> [D. M. Cohen et al., 1997]

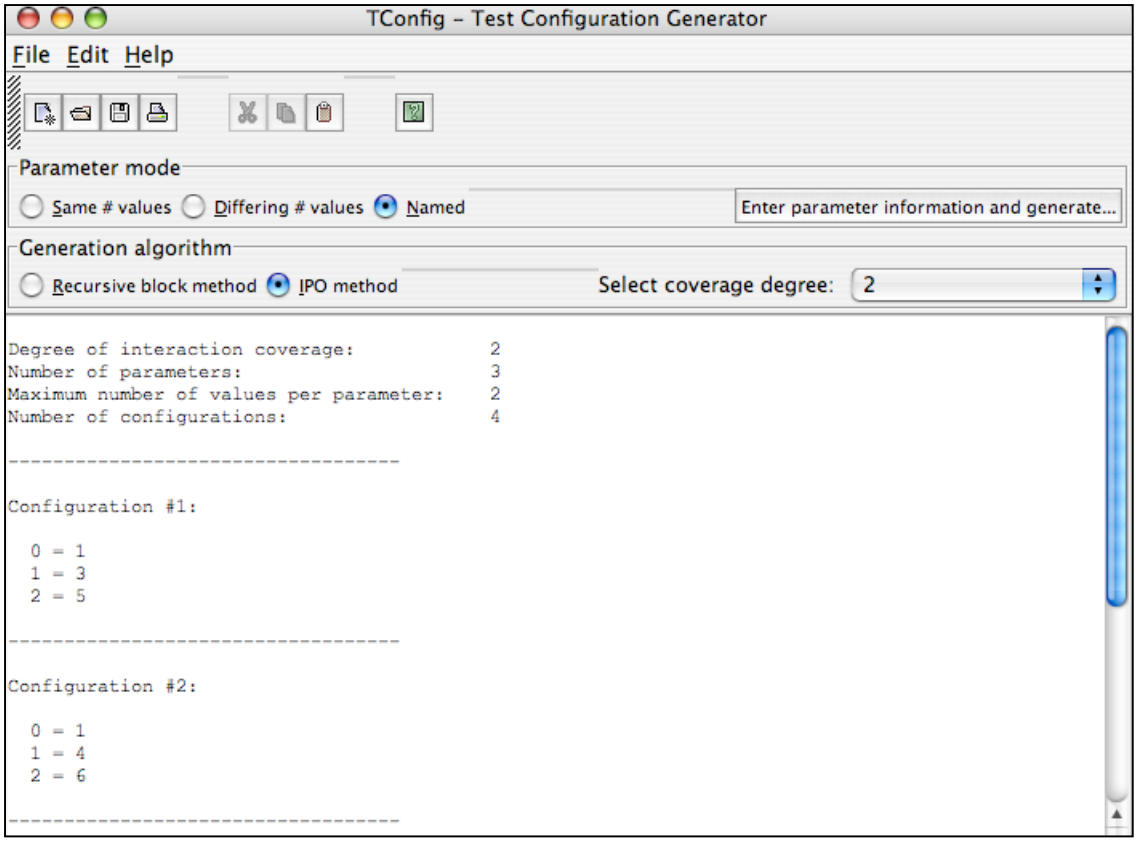
> Our initial trial of this was on a subset Nortel's internal e-mail system where we able cover 97% of branches with less than 100 valid and invalid testcases, as opposed to 27 trillion exhaustive testcases.
>
> [K. Burr and W. Young, 1998]

Checking for all-pairs coverage

- It is easy to check whether a proposed set of combinations covers all pairwise combinations.
- Generating such a set, while keeping the number of combinations to a minimum, is another matter.
- General area is “combinatorial design”.

TConfig tool: Alan Williams

(no longer available?)



jenny tool

(http://burtleburtle.net/bob/math/jenny.html)

Suppose we want to cover all triples of features. There are (12 choose 3) = 220 ways to choose dimensions, and 23 = 8 ways to choose features within those dimensions, for a total of 1760 possible triples of features. That's a quite a bit fewer than the 4096 testcase that exhaustive testing would give. But jenny does almost two orders of magnitude better than both:

```
PROMPT>jenny -n3 2 2 2 2 2 2 2 2 2 2 2 2 | wc
    20     240     820

PROMPT>jenny -n3 2 2 2 2 2 2 2 2 2 2 2 2 | sort
 1a 2a 3a 4a 5a 6b 7b 8a 9a 10b 11a 12a
 1a 2a 3a 4a 5a 6b 7b 8b 9b 10a 11b 12b
 1a 2a 3a 4a 5b 6b 7a 8b 9b 10b 11b 12b
 1a 2a 3b 4b 5b 6a 7a 8a 9a 10b 11b 12b
 1a 2a 3b 4b 5b 6b 7b 8b 9b 10a 11a 12a
 1a 2b 3a 4b 5a 6a 7b 8a 9b 10a 11a 12b
 1a 2b 3a 4b 5b 6b 7a 8a 9a 10a 11b 12a
 1a 2b 3b 4a 5a 6a 7a 8b 9a 10a 11a 12a
 1a 2b 3b 4a 5a 6b 7b 8b 9a 10a 11b 12b
 1a 2b 3b 4a 5b 6a 7b 8a 9b 10b 11a 12a
 1a 2b 3b 4b 5a 6a 7b 8b 9b 10b 11b 12a
 1b 2a 3a 4a 5b 6a 7b 8b 9a 10a 11b 12b
 1b 2a 3a 4b 5b 6a 7a 8a 9b 10b 11a 12a
 1b 2a 3b 4a 5a 6a 7b 8b 9b 10b 11a 12b
 1b 2a 3b 4a 5a 6b 7a 8a 9b 10a 11b 12a
 1b 2a 3b 4b 5a 6b 7a 8b 9a 10a 11a 12b
 1b 2b 3a 4a 5a 6a 7a 8a 9a 10b 11b 12b
 1b 2b 3a 4b 5b 6b 7b 8b 9a 10b 11a 12a
 1b 2b 3b 4a 5b 6b 7a 8b 9b 10a 11a 12b
 1b 2b 3b 4b 5a 6b 7b 8a 9b 10b 11b 12b
```

Using Tool Output

- After getting test array, map back onto test suite:
  - Replace levels by actual values
  - Replace "wild card" levels by any arbitrary values
- For our example:

1	1	1	1	1	Explorer	Windows	DB2	WebSphere
2	1	2	2	2	Explorer	MacOS	Oracle	Apache
3	1	3	3	-	Explorer	Linux	MySQL	(either)
4	2	1	2	-	Safari	Windows	Oracle	(either)
5	2	2	3	1	Safari	MacOS	MySQL	WebSphere
6	2	3	1	2	Safari	Linux	DB2	Apache
7	3	1	3	2	Firefox	Windows	MySQL	Apache
8	3	2	1	-	Firefox	MacOS	DB2	(either)
9	3	3	2	1	Firefox	Linux	Oracle	WebSphere
10	4	1	1	1	Chrome	Windows	DB2	WebSphere
11	4	2	2	2	Chrome	MacOS	Oracle	Apache
12	4	3	3	-	Chrome	Linux	MySQL	(either)

Tool Efficiency Benchmarks

The number of test cases produced by different tools for the same model:

Parameter Sizes	AETG [1]	IPO [2]	TConfig [3]	CTS [4]	Jenny [5]	TestCover [6]	DDA [7]	AllPairs [McDowell] [5]	PICT	EXACT [8]
3^4	9	9	9	9	11	9	?	9	9	9
3^{13}	15	17	15	15	18	15	18	17	18	15
$4^{15}\ 3^{17}\ 2^{29}$	41	34	40	39	38	29	35	34	37	?
$4^1\ 3^{39}\ 2^{35}$	28	26	30	29	28	21	27	26	27	21
2^{100}	10	15	14	10	16	10	15	14	15	10
10^{20}	180	212	231	210	193	181	201	197	210	?

Concept extends to N-tuple-wise

More than 70% of bugs were detected with two or fewer conditions (75% for browser and 70% for server) and approximately 90% of the bugs reported were detected with three or fewer conditions (95% for browser and 89% for server). [...] It is interesting that a small number of conditions (n<=6) are sufficient to detect all reported errors for the browser and server software.

[R. Kuhn and M. J. Reilly, 2002]

Further Refinement:
All test combinations are not necessarily equal *strength*

1. When testing a software system, certain components may be closely interrelated. This may be determined by static analysis.

2. Operational profiles give us information that certain areas of the system are used more often than others.

3. In modifying a system only certain regions are changed therefore we want to test more strongly in this area.

4. Failures in certain parts of a system are more costlier than in others.

source: http://santos.cis.ksu.edu/STRESS/2006/Presentations/Myra-Cohen-Matt-Dwyer/stress06-part4.ppt

Constraints among attribute values

Constrained Set of Product Instances
"*Basic requires Motion Sensors and Keypad*"
"*Basic excludes Cullet Detection*"

Intrusion Detection A	Intrusion Detection B	Security Package	Door Locks
Camera Surveillance	None	Advanced	Fingerprint Scanner
Motion Sensors	Cullet Detection	Advanced	Fingerprint Scanner
Camera Surveillance	Cullet Detection	Advanced	Keypad
Motion Sensors	None	Basic	Keypad
~~Camera Surveillance~~	~~Cullet Detection~~	~~Basic~~	~~Fingerprint Scanner~~

source: http://santos.cis.ksu.edu/STRESS/2006/Presentations/Myra-Cohen-Matt-Dwyer/stress06-part4.ppt

Some References

Cohen et al., The Combinatorial Design Approach to Automatic Test Generation, *IEEE Software*, v. 13, no. 5, September 1996, pp. 83-88.

Y.-W. Tung and W. S. Aldiwan, Automating test case generation for the new generation mission software system," in *Proc. IEEE Aerospace Conf.*, 2000, pp. 431-437.

A. W. Williams, Determination of test configurations for pair-wise interaction coverage, in *Thirteenth Intl. Conf. Testing Communication Systems*, 2000, pp. 57-74.

Bryce et al., A Framework of Greedy Methods for Constructing Interaction Test Suites, *International Conference on Software Engineering*, May 2005, pp. 146-155.