Pairwise Testing in Real World

Jacek Czerwonka

Test Lead, Core OS Division, Microsoft Corp.

jacekcz@microsoft.com

24th Pacific Northwest Software Quality Conference, Portland 2006

What Is Pairwise Testing

Type: Simple, Spanned, Striped, Mirror, RAID-5
VolumeSize: 10, 100, 1000, 10000, 40000
FileSystem: FAT, FAT32, NTFS
ClusterSize: 512, 1024, 2048, 4096, 8192, 16384
Compression: On, Off

Type	VolumeSize	FileSystem	ClusterSize	Compression
RAID-5	10	FAT	1024	Off
RAID-5	40000	NTFS	16384	On
Spanned	100	FAT32	512	Off
Mirror	100	FAT	4096	On
Striped	40000	NTFS	2048	Off
Simple	1000	FAT32	4096	Off

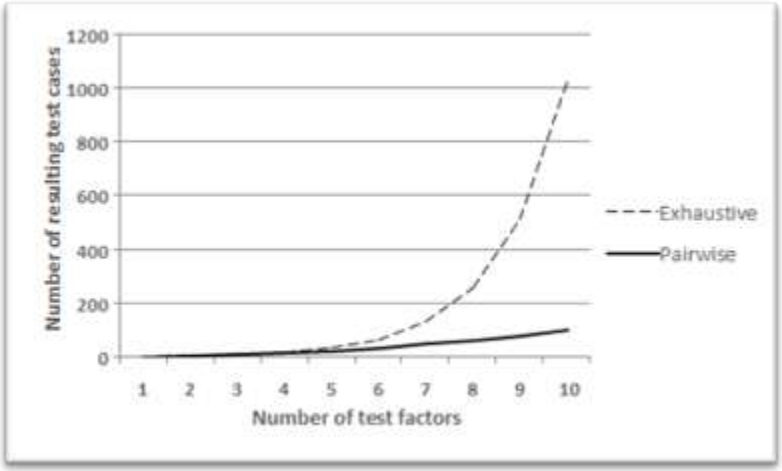
…

What Is Pairwise Testing

- *[…] a combinatorial testing method that, for each pair of input parameters to a system tests all possible discrete combinations of those parameters* [wikipedia.org]

- *[…] a technique that is based on the observation that most faults are caused by interactions of at most two factors* [pairwise.org]

- *[…] a reasonable cost-benefit compromise between often computationally infeasible higher-order combinatorial testing methods, and less exhaustive methods which fail to exercise all possible pairs of parameters* [wikipedia.org]



Focus of Research

- Effectiveness of pairwise:
  - *29 tests gave 90% block coverage for the UNIX sort command* [D. M. Cohen et al., 1997]
  - *block coverage obtained for [pairwise] was comparable with that achieved by exhaustively testing all factor combinations* [I. S. Dunietz et al., 1997]
  - *98% [problems] could have been detected by testing the device with all pairs of parameter settings* [D. R. Wallace and D. R. Kuhn, 2001]

- Pairwise generation efficiency:
  - Heuristics
  - Iterative algorithms
  - Instant (OA-, CA-based) algortihms

Practitioner's View

```
Type:        Simple, Spanned, Striped, Mirror, RAID-5
VolumeSize:  10, 100, 1000, 10000, 40000
FileSystem:  FAT, FAT32, NTFS
ClusterSize: 512, 1024, 2048, 4096, 8192, 16384
Compression: On, Off
```

- Great results:
  - Exhaustive: 900 test cases
  - Pairwise: 31 cases

- But:
  - I can't really apply FAT to a 40GB volume
  - {Size, FileSystem and ClusterSize} have to be tested more thoroughly
  - I *am able* to type *-10* in Cluster Size and Volume Size textboxes in AUT
  - I need to test some very common combinations
  - NTFS and Simple volumes are most common user's choices

PICT

- Focused on:
  - Usability
  - Speed
  - Simplicity of engine

C:\WINDOWS\system32\cmd.exe

```
$ more fs.txt
VolType:  Simple, Spanned, Striped, Mirror, RAID-5
VolSize:  10, 100, 1000, 10000, 40000
FileSys:  FAT, FAT32, NTFS
Cluster:  512, 1024, 2048, 4096, 8192, 16384
Compress: On, Off

$ pict fs.txt
VolType VolSize FileSys Cluster Compress
RAID-5  40000   FAT     1024    Off
Simple  100     NTFS    16384   On
Mirror  1000    FAT32   8192    On
Spanned 40000   NTFS    2048    On
Spanned 100     FAT32   512     Off
Mirror  10      NTFS    4096    Off
Striped 1000    FAT     16384   Off
Simple  1000    FAT     4096    On
Spanned 10      FAT     16384   On
RAID-5  10000   FAT32   2048    On
Striped 10      FAT32   1024    On
Spanned 10000   NTFS    1024    Off
Simple  40000   FAT32   16384   Off
Spanned 40000   FAT32   4096    Off
Simple  10000   FAT     2048    Off
Striped 1000    NTFS    512     On
RAID-5  10      NTFS    8192    Off
```

- Generation algo:
  - Generate combinations to pick from
  - Combine in a reasonably efficient way (greedy heuristic)

Excluding Unwanted Combinations

- **Problem**:
  - Can’t format 40GB disk with FAT
  - Can’t remove test cases – loss of good pairs

- **Solution**:
  - Extract combinations matching constraints
  - Remove them from a set to be combined

```
Type:        Simple, Spanned, Striped, Mirror, RAID-5
VolumeSize:  10, 100, 1000, 10000, 40000
FileSystem:  FAT, FAT32, NTFS
ClusterSize: 512, 1024, 2048, 4096, 8192, 16384
Compression: On, Off

if [VolumeSize] > 2048 then [FileSystem] <> "FAT";
```

Including Must-Have Combinations

- **Problem 1:**
  - I really need to test Simple NTFS volume with no Compression
  - Add extra test cases post-generation

- **Solution**:
  - Seeding files

```
Type:        Simple, Spanned, Striped, Mirror, RAID-5
VolumeSize:  10, 100, 1000, 10000, 40000
FileSystem:  FAT, FAT32, NTFS
ClusterSize: 512, 1024, 2048, 4096, 8192, 16384
Compression: On, Off
```

Type	FileSystem	ClusterSize	Compression
Simple	NTFS		Off
Mirror	FAT32	512	

Including Must-Have Combinations

- **Problem 2**:
  - I generated tests, started setting up, found out HyperThreading doesn't matter
  - Remove HT, re-generate the tests -> changed tests invalidate prior setup
  - Ignore tests with HT or change to Simple -> constraints? wasted effort?
- **Solution:**
  - Remove HT
  - Provide old test cases as seeding file
  - Re-generate

```
Type:          Simple,
VolumeSize:    10, 10(
FileSystem:    FAT, F/
ClusterSize:   512, 1(
Compression:   On, Of-

Platform:      x86, x64, ia64
CpuCount:      1, 2, 4
CpuType:       SingleCore, DualCore, HyperThreaded
```

Type	VolumeSize	FileSystem	ClusterSize	Compress	Platform	CpuCount	CpuType
Simple	1000	FAT	1024	Off	x86	4	SingleCore
RAID-5	40000	FAT32	2048	On	ia64	1	DualCore
Spanned	100	NTFS	16384	On	x64	2	HyperThreaded
Striped	10000	FAT32	512	Off	x64	1	SingleCore
Striped	10	NTFS	4096	Off	ia64	4	HyperThreaded
Mirror	10000	FAT	16384	On	x86	4	DualCore
...							

Mixed-strength Generation

- **Problem**:
  - Combinations of **VolumeSize**, **FileSystem** and **ClusterSize** need to be tested more thoroughly
  - Could set order of combinations on entire model to 3 (150 test cases)

- **Solution**:
  - Allow groups of factors to have different order than entire domain
  - Other factors "combined in" with lower order
  - Result: 90 cases

```
Type:                Simple, Spanned, Striped, Mirror, RAID-5
VolumeSize @ 3:      10, 100, 1000, 10000, 40000
FileSystem @ 3:      FAT, FAT32, NTFS
ClusterSize @ 3:     512, 1024, 2048, 4096, 8192, 16384
Compression:         On, Off
```

t = 3

t = 2

Model Hierarchy

- **Problem:**
  - I'd like to better control variability of some parameters (to decrease cost)
  - 32 tests, 20 unique combinations of **Platform**, **CpuCount**, **CpuType**
- **Solution:**
  - Combine some params first
  - Use resulting combinations in the outer domain
  - 55 tests, 9 unique combinations of **Platform**, **CpuCount**, **CpuType**

```
Type:        Simple, Spanned, Striped, Mirror, RAID-5
VolumeSize:  10, 100, 1000, 10000, 40000
FileSystem:  FAT, FAT32, NTFS
ClusterSize: 512, 1024, 2048, 4096, 8192, 16384
Compression: On, Off
Platform:    x86, x64, ia64
CpuCount:    1, 2, 4
CpuType:     SingleCore, DualCore, HyperThreaded

{ Platform, CpuCount, CpuType } @ 2
```

```
Type
VolumeSize
FileSystem
ClusterSize
Compression
[CompoundParam]
   Platform
   CpuCount
   CpuType
```

Negative Testing

- **Problem**:
  - I can type -10 in VolumeSize and ClusterSize
  - I could add -10 to both factors:
    - Blocks other pairs from being executed
    - Input masking

- **Solution**:
  - Special treatment for values outside of allowed range
    - All pairs of in-range values still covered
    - All pairs of out-of-range with in-range values covered
    - Two out-of-range values never tested together

```
Type:        Simple, Spanned, Striped, Mirror, RAID-5
VolumeSize:  *-10, 10, 100, 1000, 10000, 40000
FileSystem:  FAT, FAT32, NTFS
ClusterSize: *-10, 512, 1024, 2048, 4096, 8192, 16384
Compression: On, Off
```

Weights

- **Problem**:
  - NTFS and Simple volumes are used the most
  - If nothing else matters, we should pick preferred values

- **Solution**:
  - Weights change probabilities of values being chosen...
  - ...all other criteria being equal

```
Type:        Simple (5), Spanned, Striped, Mirror, RAID-5
VolumeSize:  10, 100, 1000, 10000, 40000
FileSystem:  FAT, FAT32, NTFS (5)
ClusterSize: 512, 1024, 2048, 4096, 8192, 16384
Compression: On, Off
```

Expected Results

- **Problem**:
  - I want to map expected results onto the domain model
  - Custom processing scripts -> costly

- **Solution**:
  - Special parameter type
  - Use familiar syntax

```
# sum( int[] array, int start, int count )

Array:  *Null, Empty, Valid
Start:  *TooLow, InRange, *TooHigh
Count:  *TooFew, Some, All, *TooMany

$Result: Pass, OutOfBounds, InvalidPointer

if [Array] = "Null" then [$Result] = "InvalidPointer";

if [Start] in {"TooLow", "TooHigh"} or
   [Count] in {"TooFew", "TooMany"} then [$Result] = "OutOfBounds";
```

Efficiency Comparison

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

[1] Y. Lei and K. C. Tai In-parameter-order: a test generation strategy for pairwise testing, p. 8.
[2] K. C. Tai and Y. Lei A Test Generation Strategy for Pairwise Testing, p. 2.
[3] A. W. Williams Determination of Test Configurations for Pair-wise Interaction Coverage, p. 15.
[4] A. Hartman and L. Raskin Problems and Algorithms for Covering Arrays, p. 11.
[5] Supplied by Bob Jenkins.
[6] Supplied by George Sherwood.
[7] C. J. Colbourn, M. B. Cohen, R. C. Turban A Deterministic Density Algorithm for Pairwise Interaction Coverage, p. 6.

Summary

- Pure pairwise has practical limitations
- Tools need to enable real-world scenarios
- PICT:
  - Handles some practical scenarios
  - Industrial strength tool
  - Fast and efficient enough in generating tests
  - Get from: www.pairwise.org/tools.asp

- jacekcz@microsoft.com

Thank You!