Reexamining Black Box and White Box Sampling Strategies for Testing Large-Scale Configurable Systems

Adam Porter

Configuration Spaces Grow Fast

- Exhaustive testing is generally too expensive
  - Cfg spaces large, individual tests are expensive
- Nearest Neighbor Search (MySQL)
  - 110k cfgs, 3hrs/cfg, 50 cpus took > 6 mos
- Component-Based System Testing (Intercom)
  - Very restricted subset had ~ 3500 cfgs, ~ 3hrs/cfg, takes ~15 mos.

Example: Phone S/W Product Line

Possible Values	Product Line Options (factors)				
	Display	**Email Viewer**	**Camera**	**Video Camera**	**Video Ringtones**
	16 Million Colors	Graphical	2 Megapixels	Yes	Yes
	8 Million Colors	Text	1 Megapixel	No	No
	Black and White	None	None		

Example: Phone S/W Product Line

Possible Values	Product Line Options (factors)				
	Display	**Email Viewer**	**Camera**	**Video Camera**	**Video Ringtones**
	16 Million Colors	Graphical	2 Megapixels	Yes	Yes
	8 Million Colors	Text	1 Megapixel	No	No
	Black and White	None	None		

Example: Phone S/W Product Line

Possible Values	Product Line Options (factors)				
	Display	**Email Viewer**	**Camera**	**Video Camera**	**Video Ringtones**
	16 Million Colors	Graphical	2 Megapixels	Yes	Yes
	8 Million Colors	Text	1 Megapixel	No	No
	Black and White	None	None		

Example: Phone S/W Product Line

Possible Values	Product Line Options (factors)				
	Display	**Email Viewer**	**Camera**	**Video Camera**	**Video Ringtones**
	16 Million Colors	Graphical	2 Megapixels	Yes	Yes
	8 Million Colors	Text	1 Megapixel	No	No
	Black and White	None	None		

6

Example: Phone S/W Product Line

Possible Values	Product Line Options (factors)				
	Display	**Email Viewer**	**Camera**	**Video Camera**	**Video Ringtones**
	16 Million Colors	Graphical	2 Megapixels	Yes	Yes
	8 Million Colors	Text	1 Megapixel	No	No
	Black and White	None	None		



7

Example: Phone S/W Product Line

Possible Values	Product Line Options (factors)				
	Display	**Email Viewer**	**Camera**	**Video Camera**	**Video Ringtones**
	16 Million Colors	Graphical	2 Megapixels	Yes	Yes
	8 Million Colors	Text	1 Megapixel	No	No
	Black and White	None	None		



8

Example: Phone S/W Product Line

Possible Values	Product Line Options (factors)				
	Display	**Email Viewer**	**Camera**	**Video Camera**	**Video Ringtones**
	16 Million Colors	Graphical	2 Megapixels	Yes	Yes
	8 Million Colors	Text	1 Megapixel	No	No
	Black and White	None	None		



9

A Combinatorial Problem

Display	Email Viewer	Camera	Video Camera	Video Ringtones
16 Million Colors	Graphical	2 Megapixels	Yes	Yes
8 Million Colors	Text	1 Megapixel	No	No
Black and White	None	None		

A Combinatorial Problem

Display	Email Viewer	Camera	Video Camera	Video Ringtones
3				
16 Million Colors	Graphical	2 Megapixels	Yes	Yes
8 Million Colors	Text	1 Megapixel	No	No
Black and White	None	None		

A Combinatorial Problem

Display	Email Viewer	Camera	Video Camera	Video Ringtones
3 x 3				
16 Million Colors	Graphical	2 Megapixels	Yes	Yes
8 Million Colors	Text	1 Megapixel	No	No
Black and White	None	None		

A Combinatorial Problem

Display	Email Viewer	Camera	Video Camera	Video Ringtones
16 Million Colors	Graphical	2 Megapixels	Yes	Yes
8 Million Colors	Text	1 Megapixel	No	No
Black and White	None	None		

3 x 3 x 3

A Combinatorial Problem

Display	Email Viewer	Camera	Video Camera	Video Ringtones
16 Million Colors	Graphical	2 Megapixels	Yes	Yes
8 Million Colors	Text	1 Megapixel	No	No
Black and White	None	None		

3 x 3 x 3 x 2 x 2

A Combinatorial Problem

Display	Email Viewer	Camera	Video Camera	Video Ringtones
16 Million Colors	Graphical	2 Megapixels	Yes	Yes
8 Million Colors	Text	1 Megapixel	No	No
Black and White	None	None		

- This example has $3^3x2^2=108$ cfgs
- A subset of MySQL we tested in previous work had ~72,000,000 cfgs
  - At 3hrs per test, would take ~24,650 cpu years to test

Combinatorial Interaction Testing (CIT)

- Can't test entire cfg space
- CIT systematically samples cfg space
- Covering arrays
  - Select a subset of cfgs such that all t-way combs of values occur at least once
  - t is called the testing *strength*

A 2-way Covering Array

	Display	Email Viewer	Camera	Video	V. Ringtones
1	16 Million Colors	None	1 Megapixel	Yes	Yes
2	8 Million Colors	Text	None	No	No
3	16 Million Colors	Text	2 Megapixels	No	Yes
4	Black and White	None	None	No	Yes
5	8 Million Colors	None	2 Megapixels	Yes	No
6	16 Million Colors	Graphical	None	Yes	No
7	Black and White	Text	1 Megapixel	Yes	No
8	8 Million Colors	Graphical	1 Megapixel	No	Yes
9	Black and White	Graphical	2 Megapixels	Yes	Yes

Sampling the Space

	Display	Email Viewer	Camera	Video	V. Ringtones
1	16 Million Colors	None	1 Megapixel	Yes	Yes
2	8 Million Colors	Text	None	No	No
3	16 Million Colors	Text	2 Megapixels	No	Yes
4	Black and White	None	None	No	Yes
5	8 Million Colors	None	2 Megapixels	Yes	No
6	16 Million Colors	Graphical	None	Yes	No
7	Black and White	Text	1 Megapixel	Yes	No
8	8 Million Colors	Graphical	1 Megapixel	No	Yes
9	Black and White	Graphical	2 Megapixels	Yes	Yes

Sampling For Testing

	Display	Email Viewer	Camera	Video	V. Ringtones
1	16 Million Colors	None	1 Megapixel	Yes	Yes
2	8 Million Colors	Text	None	No	No
3	16 Million Colors	Text	2 Megapixels	No	Yes
4	Black and White	None	None	No	Yes
5	8 Million Colors	None	2 Megapixels	Yes	No
6	16 Million Colors	Graphical	None	Yes	No
7	Black and White	Text	1 Megapixel	Yes	No
8	8 Million Colors	Graphical	1 Megapixel	No	Yes
9	Black and White	Graphical	2 Megapixels	Yes	Yes

Sampling For Testing

	Display	Email Viewer	Camera	Video	V. Ringtones
1	16 Million Colors	None	1 Megapixel	Yes	Yes
2	8 Million Colors	Text	None	No	No
3	16 Million Colors	Text	2 Megapixels	No	Yes
4	Black and White	None	None	No	Yes
5	8 Million Colors	None	2 Megapixels	Yes	No
6	16 Million Colors	Graphical	None	Yes	No
7	Black and White	Text	1 Megapixel	Yes	No
8	8 Million Colors	Graphical	1 Megapixel	No	Yes
9	Black and White	Graphical	2 Megapixels	Yes	Yes

CIT Summary

- CIT approaches used to limit test suite size
  - e.g. covering arrays
- Effort concentrated on these configurations
  - May still be too many in practice
  - MySQL case study – ~1k cfgs/release , 2-4hrs/cfg, using 50 client cpus, takes > 40-80 hrs to test
- Other configurations receive little attention
  - Problems ultimately found in the field

Another Look at CIT

- Most research focused on black box approaches
  - They define a criteria & select cfgs that fully cover it
- In absence of other info, must assume that interactions could be *complete*
  - Any combo of options meeting coverage criteria might interact
  - Therefore all must be tested

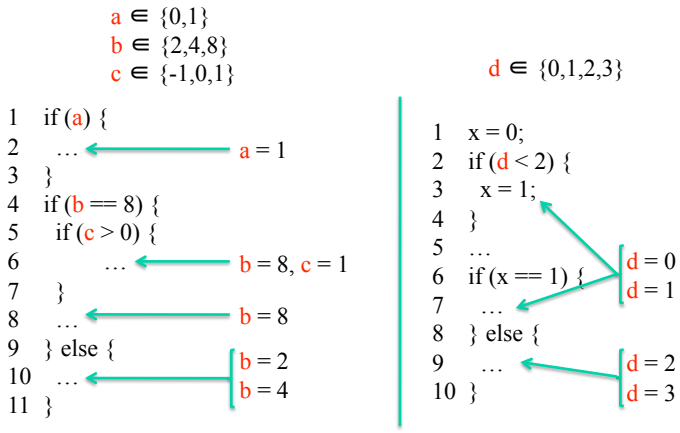
Are All These Interactions Possible?

- Conjecture: in practice, interactions are sparse
  - Few options/settings interact
- If true,
  - Black box approaches, such as CIT, may test more cfgs than strictly necessary, while not testing other important cfgs
  - It might be possible to limit test suite size further, without compromising effectiveness

What is an Interaction, Really?

- A *minimal* partial configuration that g*uarantees coverage* of a code element
  - e.g., {a=1 ∧ b=0} is an interaction *iff* some entity E is guaranteed to be covered whenever {a=1 ∧ b=0}, but E is not guaranteed to be covered when {a=1} ∨ {b=0} ∨ {*true*}

Finding Interactions in Code



Experimental Approach

- Given a system, configuration space & test suite
- Define a *coverage metric*
- Compute *maximum coverage* under all cfgs
- Analyze resulting data
  - Determine which option setting(s) *interact*
    - i.e., which option setting(s) needed to achieve max coverage
  - Compute *minimal set* of cfgs that achieves max coverage
    - Compare to other approaches, such as covering arrays

Subject Systems

Name	vsftpd	ngIRCd	grep
Version	2.0.7	0.12.0	2.4.2
SLOC	10,482	13,601	8,456
Exec. LOC	4412	4387	3302
#Test cases	64	142	113
#Cfg options	30	13	18
Boolean	20	5	14
Integer	10	8	4
Full config. space	$1.4x10^{11}$	$4.2x10^{7}$	$6.7x10^{7}$

Coverage Criteria

- We compute all paths and then project down to
  - Line
  - Block
  - Edge
  - Condition

Computing Maximum Coverage

- Used symbolic evaluation technology to compute **all** program paths reachable under **any** cfg
- Symbolic evaluator called Otter
  - Similar to Klee [Cadar et al. 2008]

Notional Symbolic Evaluation Process

- Mark program vars corresponding to cfg options as symbolic
  - Requires manual intervention
  - Non-symbolic variables are concrete (set by test cases)
- Set initial path condition = {*true*}
- Run code in Otter
  - Interprets concrete statements normally
  - Tracks all references/updates involving symbolic values

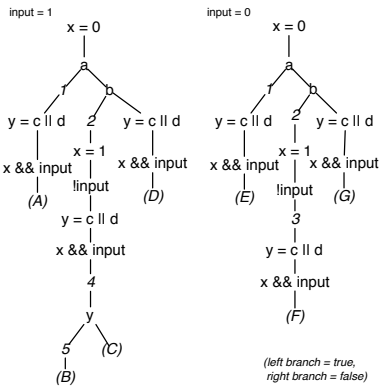
Notional Symbolic Evaluation Process

- On reaching a conditional stmt containing symbolic vars, updates path conditions & forks execution, if necessary
  - Duplicate existing path condition
  - For true branch's path condition, add conditional
  - For false branch's, add negation of conditional
  - For each path condition, invoke SAT solver (SMT)
    - if not satisfiable, exit, else, continue execution
- When all executions terminate, the set of path conditions describes **all** entities (e.g., lines, blocks, etc.) reachable under **any** configuration

Example Execution

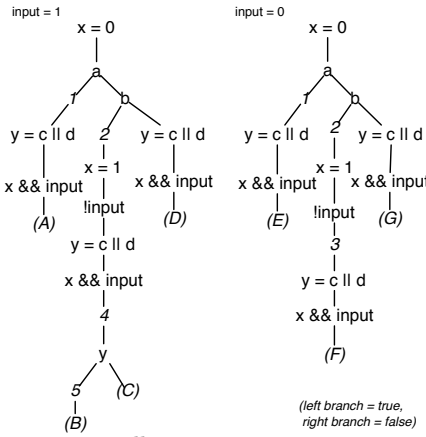
```
1  int a=α, b=β,
2      c=γ, d=δ; /* symbolic */
3  int input=...; /* concrete */
4  int x = 0;
5  if (a)
6    /* 1 */
7  else if (b) {
8    /* 2 */
9    x = 1;
10   if (!input) {
11     /* 3 */
12   }
13 }
14 int y = c || d;
15 if (x && input) {
16   /* 4 */
17   if (y)
18     /* 5 */
19 }
```

Example program



Full execution trees

Example Execution



Full execution trees

	Path condition and config.	Cov
input = 1	(A) α a = 1	1
	(B) $\neg\alpha \wedge \beta \wedge (\gamma \vee \delta)$ a = d = 0, b = c = 1	2, 4, 5
	(C) $\neg\alpha \wedge \beta \wedge \neg(\gamma \vee \delta)$ a = c = d = 0, b = 1	2, 4
	(D) $\neg\alpha \wedge \neg\beta$ a = b = 0	–
input = 0	(E) α a = 1	1
	(F) $\neg\alpha \wedge \beta$ a = 0, b = 1	3
	(G) $\neg\alpha \wedge \neg\beta$ a = b = 0	–

Path conditions and configurations

Maximum Coverage

Name	**vsftpd**	**ngIRCd**	**grep**
# Paths	30304	53205	625181
Line Cov. (%)	62	73	75
# Opts. used / total	22/30	13/13	17/18

Practical Limitations

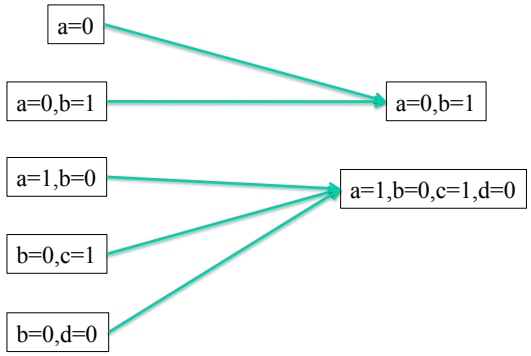
- Interpreter maturity – Otter currently doesn't handle
  - multithreading, dereferencing symbolic pointer values, floating-point arithmetic, or inline assembly
- Whole program analysis
  - Users must supply mock library implementations
  - Possible source of errors
  - Actual libraries could, in theory, modify configuration options
- SAT solver complexity
  - Specific invocations might take a long time to finish

Compute Minimal Covering Set

- We know all lines coverable & which paths are covered in each configuration
- Therefore, would like to find the smallest set of cfgs achieving max coverage (minimal covering set)
  - Problem is NP-hard
  - Use greedy algorithm to approximate
- Can be smaller than the number of path conditions
  - A single cfg can satisfy multiple path conditions

Minimal Covering Sets

- Multiple interactions can be exercised with a single configuration



Compute Minimal Covering Set (cont.)

- For each path condition, use SAT solver to derive one partial cfg that satisfies it
- Merge coverage info for compatible partial cfgs
  - Two cfgs compatible when they have the same values for all cfg options appearing in a given path condition
- Greedy algorithm:

L = {coverable lines}, $MCS = \varnothing$
while L is not empty
  C = partial cfg covering the largest number of lines in L
  $MCS = MCS \cup C$
  $L = L$ – covered(C)

Example Execution

	Path condition and config.	Cov
input = 1	(A) α $\mathsf{a} = 1$	1
	(B) $\neg\alpha \wedge \beta \wedge (\gamma \vee \delta)$ $\mathsf{a} = \mathsf{d} = 0, \mathsf{b} = \mathsf{c} = 1$	2, 4, 5
	(C) $\neg\alpha \wedge \beta \wedge \neg(\gamma \vee \delta)$ $\mathsf{a} = \mathsf{c} = \mathsf{d} = 0, \mathsf{b} = 1$	2, 4
	(D) $\neg\alpha \wedge \neg\beta$ $\mathsf{a} = \mathsf{b} = 0$	–
input = 0	(E) α $\mathsf{a} = 1$	1
	(F) $\neg\alpha \wedge \beta$ $\mathsf{a} = 0, \mathsf{b} = 1$	3
	(G) $\neg\alpha \wedge \neg\beta$ $\mathsf{a} = \mathsf{b} = 0$	–

Path	Config	Covered so far
(B)	a=d=0, b=c=1	{2,4,5}
(A),(E)	a=1	{1,2,4,5}
(C),(F)	a=c=d=0, b=1	{1,2,3,4,5}

Minimal Covering Sets

	vsftpd		ngIRCd		grep	
	Line	Condition	Line	Condition	Line	Condition
1	2,521	1,132	3,148	1,881	2,218	1,810
2	18	71	30	27	171	231
3	8	14	6	23	34	45
4	1	9	6	5	20	25
5	1	2	1	4	5	11
6		1	1	1	5	8
7		1	1	1	3	7
8		1		1	2	6
9		1		1	2	5
10						1

Compare to Covering Arrays

- For each system generated 1, t-way covering array for each value of t, $1 \leq t \leq 3$
- Ran each cfg in each covering array with all options concretely set

Comparison with Covering Arrays

	vsftpd		**ngIRCd**		**grep**	
	# Cfgs	Cov. (%)	# Cfgs	Cov. (%)	# Cfgs	Cov. (%)
1-way	3	2.6	7	56.2	3	62.6
2-way	12	40.1	28	62.0	13	64.1
3-way	41	44.8	131	62.2	42	64.3
Min.	9	49.2	18	62.2	10	64.7

Some Observations

- Covering arrays required more cfgs than strictly necessary to achieve maximum coverage
- Many covering array cfgs were redundant. adding no unique line coverage
- Hard to predict what strength *t* to use
  - For ngIRCd a 3-way covering array was enough, but for the other programs it was not.

Refining the Configuration Spaces

- Each program exhibited masking effects
  - Option settings that completely dictate behavior
- Grep: abbreviated behavior
  - showhelp
- Vsftpd: checks for violation of implicit constraints
  - check & exit when in 1-process mode & ssl enabled
- ngIRCd: performance boundaries
  - Many clients & short resp. timeouts cause dropped connections

Refined Configuration Space

- Added constraints that limit/avoid masking effects
- Reran analysis

	vsftpd		ngIRCd		grep	
	#Cfgs	Cov. (%)	#Cfgs	Cov. (%)	#Cfgs	Cov. (%)
1-way	3	43.5	7	61.5	3	63.2
2-way	12	48.9	28	61.6	12	64.0
3-way	40	48.9	119	61.6	37	64.2
Min.	6	48.9	8	61.6	10	64.2

Updated Observations

- Differences are smaller, but basic observations are unchanged
  - Covering arrays larger than necessary for coverage
  - Many covering array configurations are redundant
  - Hard to predict what strength *t* to use

Understanding Option Interactions

- Guaranteed coverage, *Cov(p)*, is the set of lines always covered by test suite when p holds
- Lines always covered:
  - $S_0 = Cov(true)$
- Lines covered by a 1-way covering array
  - $S_1 = \cup_{i=opt} \cup_{j=vals(opt)} Cov(x_{ij})$
- Easily extensible to any any value of *t*

Approximating Cov(p)

- $Cov(p) = \cup_T Cov_T(p)$, where $Cov_T(p)$ is the coverage guaranteed when running test case *T*
- Let p_i be a path condition from *T*'s sym. exec.
- Let $L(p_i)$ be the lines occuring in p_i
- $Compat(p) = \{p_i \mid \text{SAT}(p_i \wedge p)\}$
- $Cov_T(p) = \cap_{pj \in Compat(p)} L(p)$

Example Execution

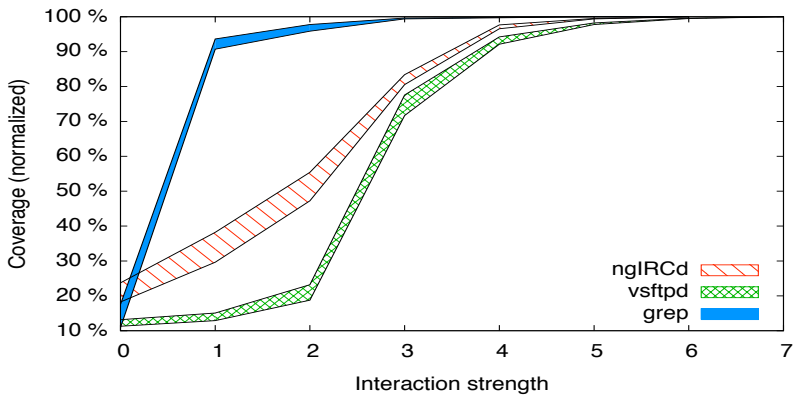
	Path condition and config.	Cov
input = 1	(A) α a = 1	1
	(B) $\neg\alpha \wedge \beta \wedge (\gamma \vee \delta)$ a = d = 0, b = c = 1	2, 4, 5
	(C) $\neg\alpha \wedge \beta \wedge \neg(\gamma \vee \delta)$ a = c = d = 0, b = 1	2, 4
	(D) $\neg\alpha \wedge \neg\beta$ a = b = 0	–
input = 0	(E) α a = 1	1
	(F) $\neg\alpha \wedge \beta$ a = 0, b = 1	3
	(G) $\neg\alpha \wedge \neg\beta$ a = b = 0	–

p	Compat(p) (input = 1)	Compat(p) (input = 0)	Cov(p)
α	(A)	(E)	{1}
¬α	(B),(C),(D)	(F),(G)	∅
¬α ∧ β	(B),(C)	(F)	{2,3,4}
¬α ∧ β ∧ γ	(B)	(F)	{2,3,4,5}

Interactions by Strength

	vsftpd		ngIRCd		grep	
Strength	Line	Condition	Line	Condition	Line	Condition
1	7	9	11	17	13	23
2	4	4	19	33	27	45
3	3	4	33	37	36	49
4	16	32	117	131	7	16
5	5	14	144	174	5	9
6	6	9	111	111		2
7	2	2				

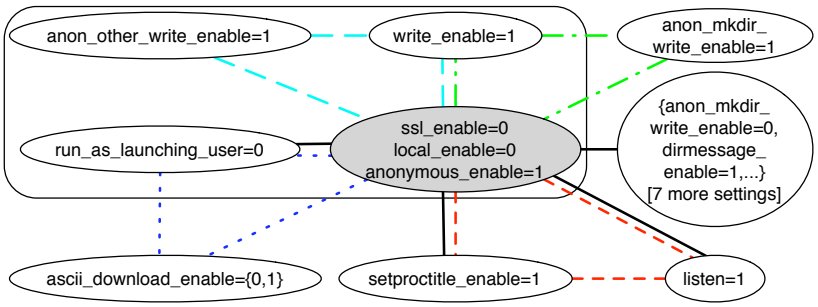
Coverage by Interaction Strength



Observations

- 1-way covering arrays guarantee a significant amount of coverage
- Most lines are guaranteed to be covered by 2- or 3-way covering arrays
- Higher strength covering arrays are still required to guarantee full coverage
  - All 3 subject programs have lines that cannot be covered unless 3 or more options take specific settings

Interaction Graphs



Some Observations

- Many options are missing from the graph because they guarantee no coverage
- Several options display weak interactions—some settings guarantee coverage, but others don't
- A small number display strong interactions
- Those options that do interact appear to form disjoint clusters

Conclusions

- Minimal covering sets were very small
- Covering arrays had very high line coverage, but
  - Missed some cfgs needed for maximum coverage
  - Included many redundant cfgs
- Options interactions were sparse and clustered
- Results provide support our conjecture that many cfg options simply do not interact with each other