Using Logic-Based Reduction for Adversarial Component Recovery*

J. Todd McDonald, Eric D. Trias, Yong C. Kim,
and Michael R. Grimaila

Center for Cyberspace Research
Air Force Institute of Technology
WPAFB, OH

*The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government

Outline

- Protection Context
- Polymorphic Variation as Protection
- Hiding Properties of Interest
- Framework and Experimental Results

Protection Context

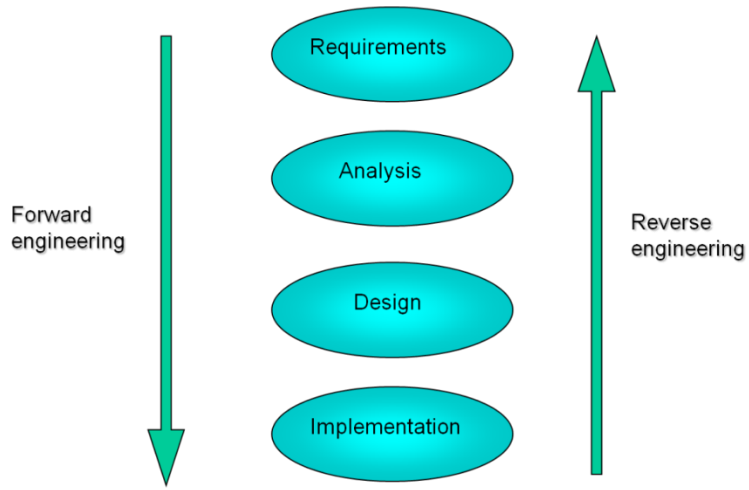
- Embedded Systems / “Hardware”
  - Increasingly represented as reprogrammable logic (i.e., software!)
  - We used to like hardware because it offered “hard” solutions for protection (physical anti-tamper, etc.)

- Our beginning point: what happens if hardware-based protections fail?
  - Hardware protection: I try to keep you from physically getting the netlist/machine code
  - Software protection: I give you a netlist/machine code listing and ask you questions pertaining to some protection property of interest

- Protection/exploitation both exist in the eye of the beholder

Protection Context

- Critical military / commercial systems vulnerable to malicious reverse engineering attacks
  - Financial loss
  - National security risk

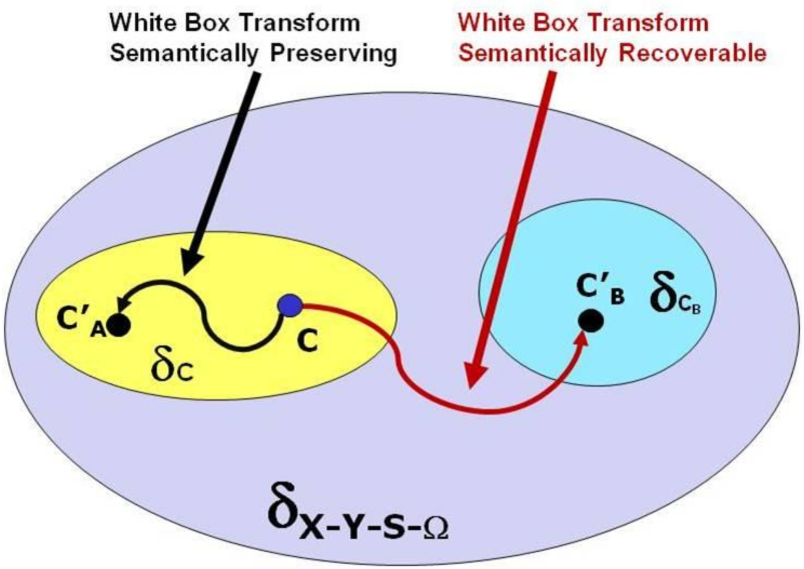


- Reverse Engineering and Digital Circuit Abstractions
  - Architectural (Behavioral)
  - Register Transfer Language (RTL)
  - Gate Level
  - Transistor Level
  - Layout

INCREASING DETAIL

Polymorphic Variation as Protection

- Experimental Approach:
  - Consider practical / real-world / theoretic circuit properties related to security
  - Use a variation process to create polymorphic circuit versions
    - *Polymorphic* = *many forms* of circuits with semantically equivalent or semantically recoverable functionality
  - Characterize algorithmic effects:
    - Empirically demonstrate properties
    - Prove as intractable
    - Prove as undecidable

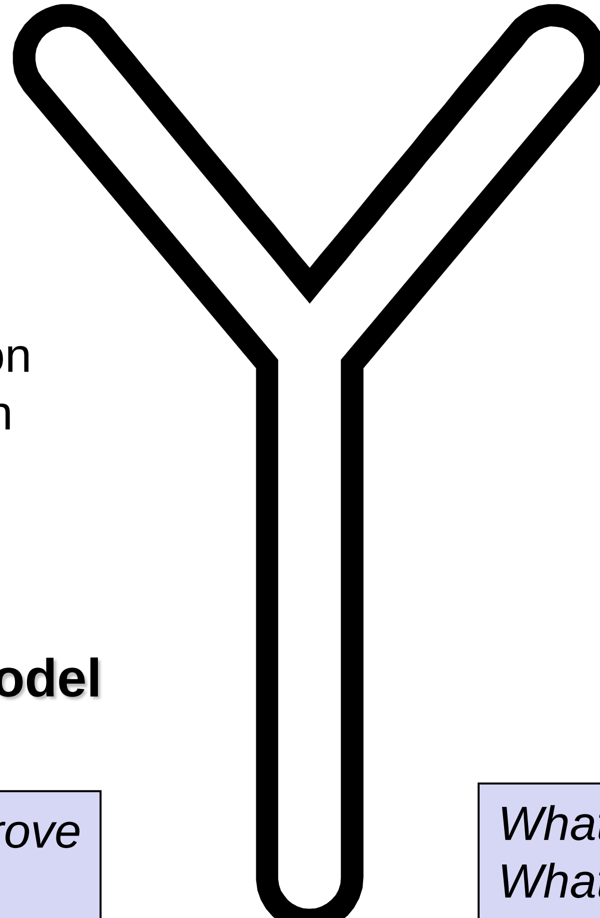


Semantic Changing

Black-Box Refinement
Semantic Transformation
Polymorphic Generation

Program Encryption
Random Program Model

What can I prove / not prove under RPM?



Semantic Preserving

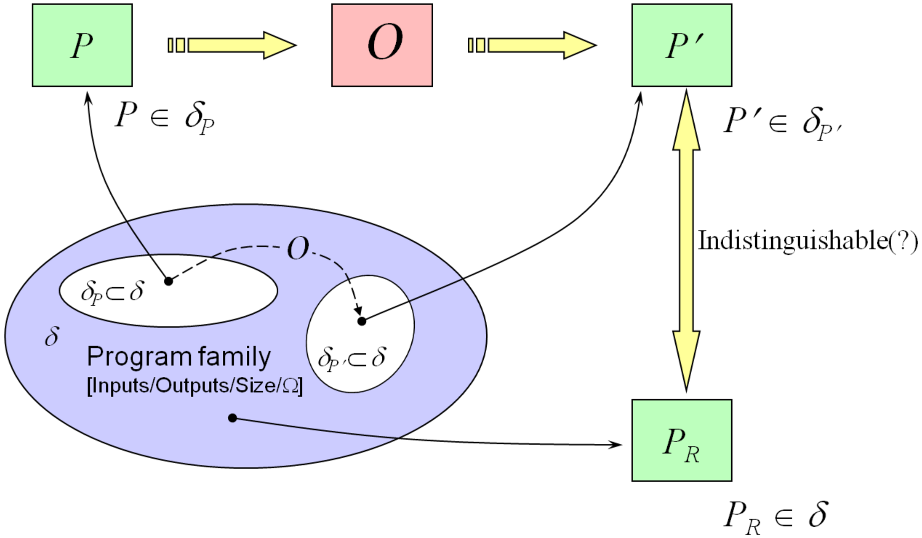
Polymorphic Generation

Obfuscation

What can I measure?
What can I characterize?
What are the limits if I am only allowed to retain functionality?

Defining Obfuscation

- Since we can't hide *all* information leakage….
  - Can we protect intent?
    - Tampering with code in order to get specific results
    - Manipulating input in order to get specific results
    - Correlating input/output with environmental context
  - Can we impede identical exploits on functionally equivalent versions?
  - Can we define and measure *any* useful definition of hiding short of absolute proof and not based *solely* on variant **size**?



Hierarchy of Obfuscating Transforms

Logical View

- Functional Hiding
- Control Hiding
- Component Hiding
- Signal Hiding
- Topology Hiding (Gate Replacement)

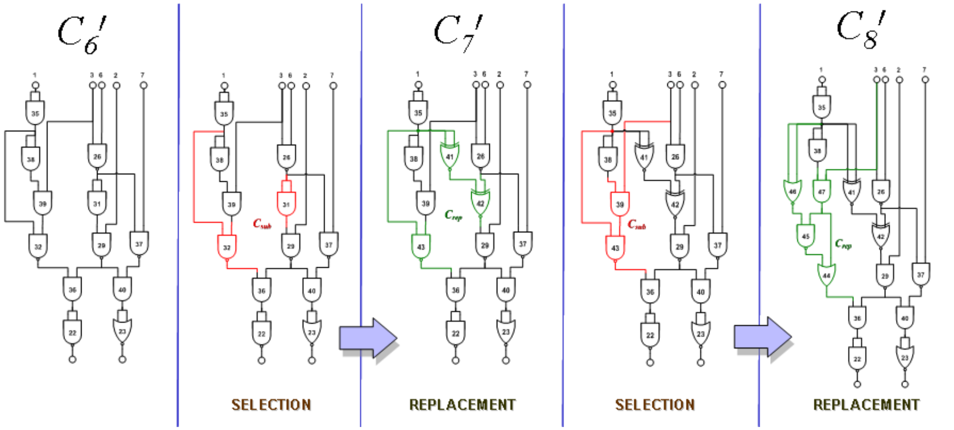
Physical Manifestation

- Side Channel Properties

Algorithm and Variant Characterization:

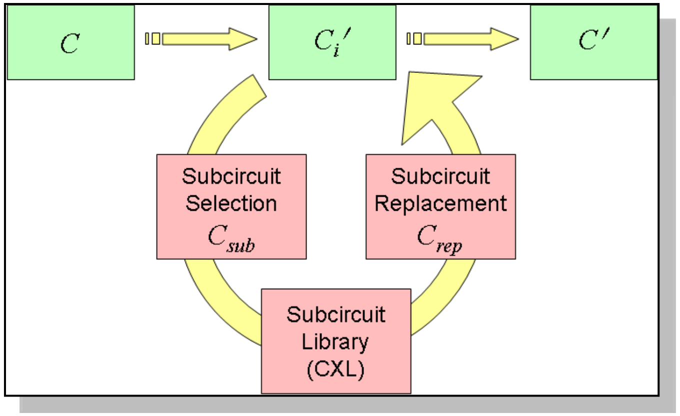
Selection:

1) Random
2) Deterministic
3) Mixture



Replacement

1) Random
2) Deterministic
3) Mixture



Framework and Experimental Results

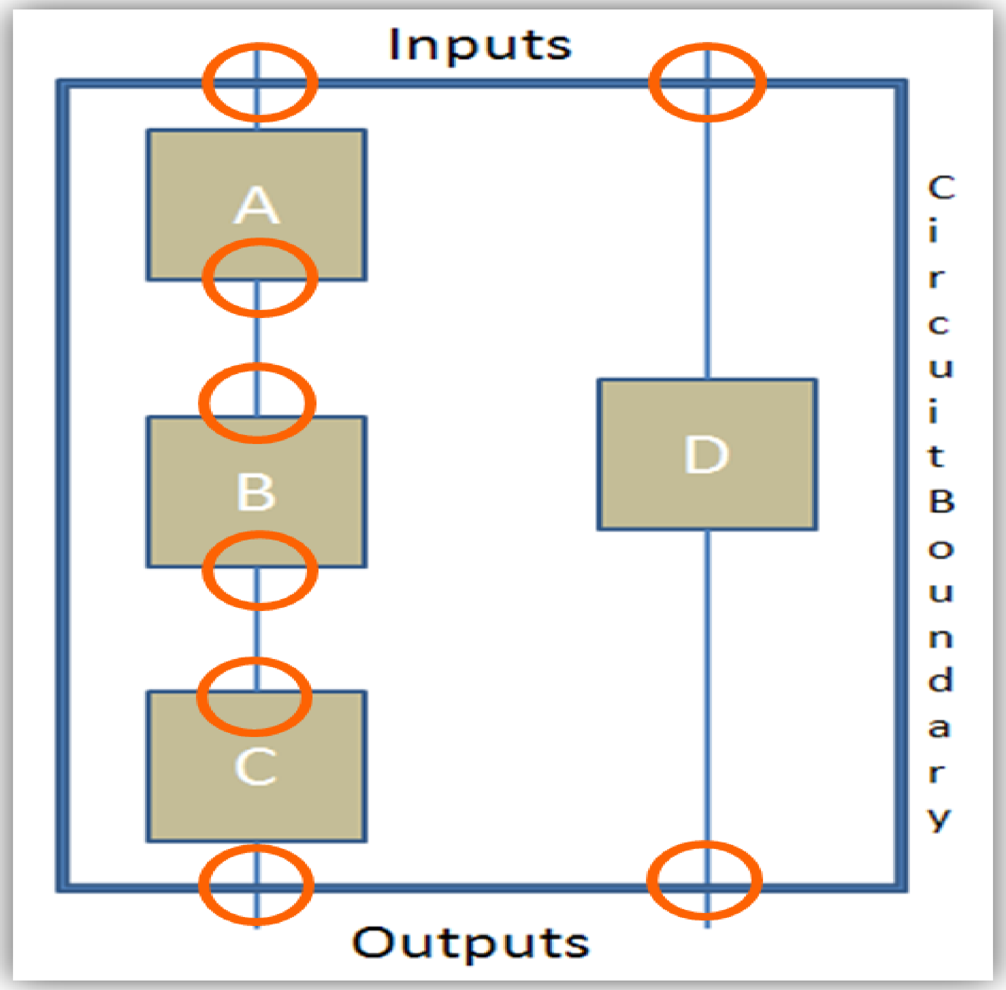
- When does (random/deterministic) iterative selection and replacement:
  1) Manifest hiding properties of interest?
  2) Cause an adversarial reverse engineering task to become intractable or undecidable?

- What role does logic reduction and adversarial reversal play in the outcome (ongoing)

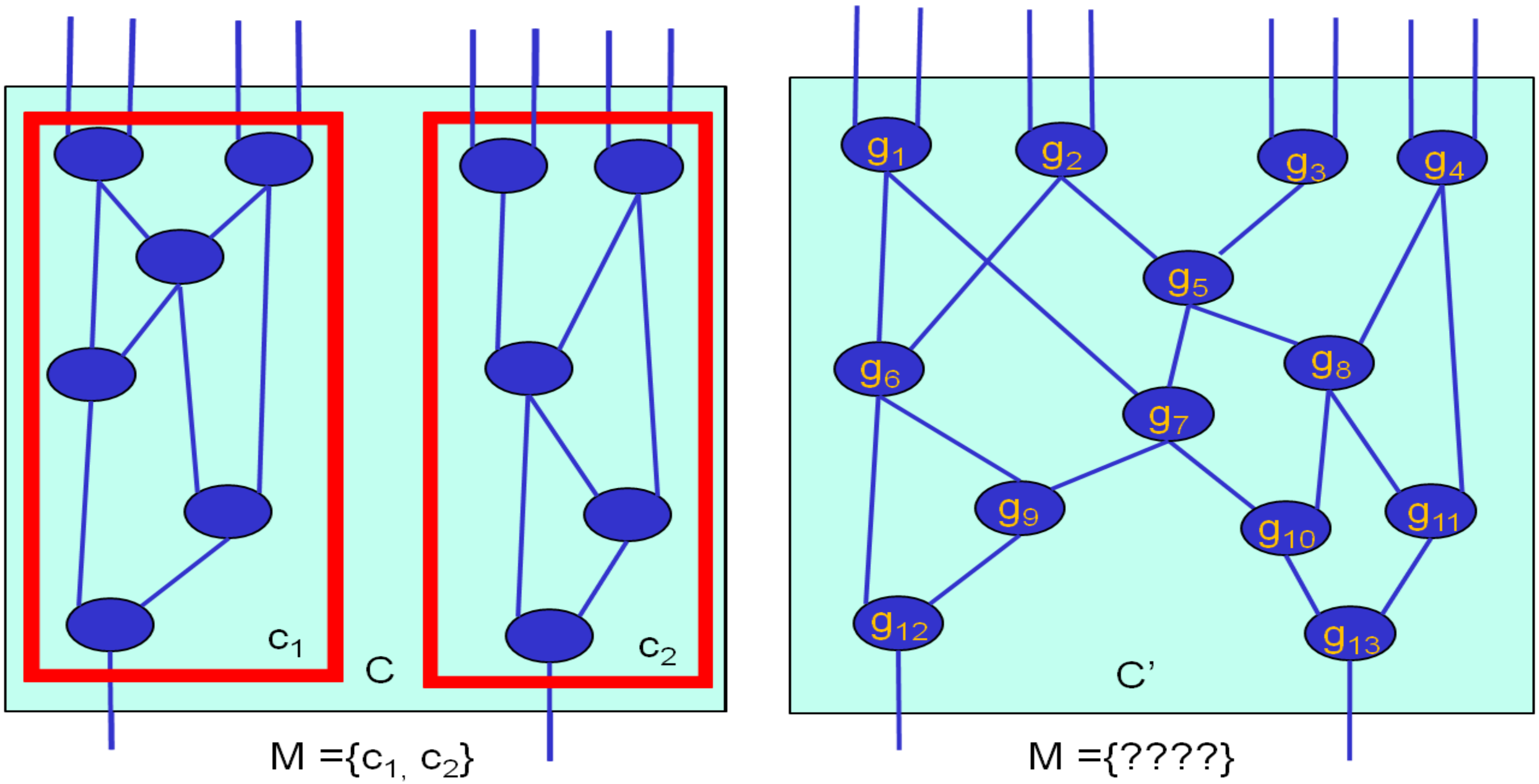
- Are there circuits which will fail despite the best variation we can produce? (yes)

Components

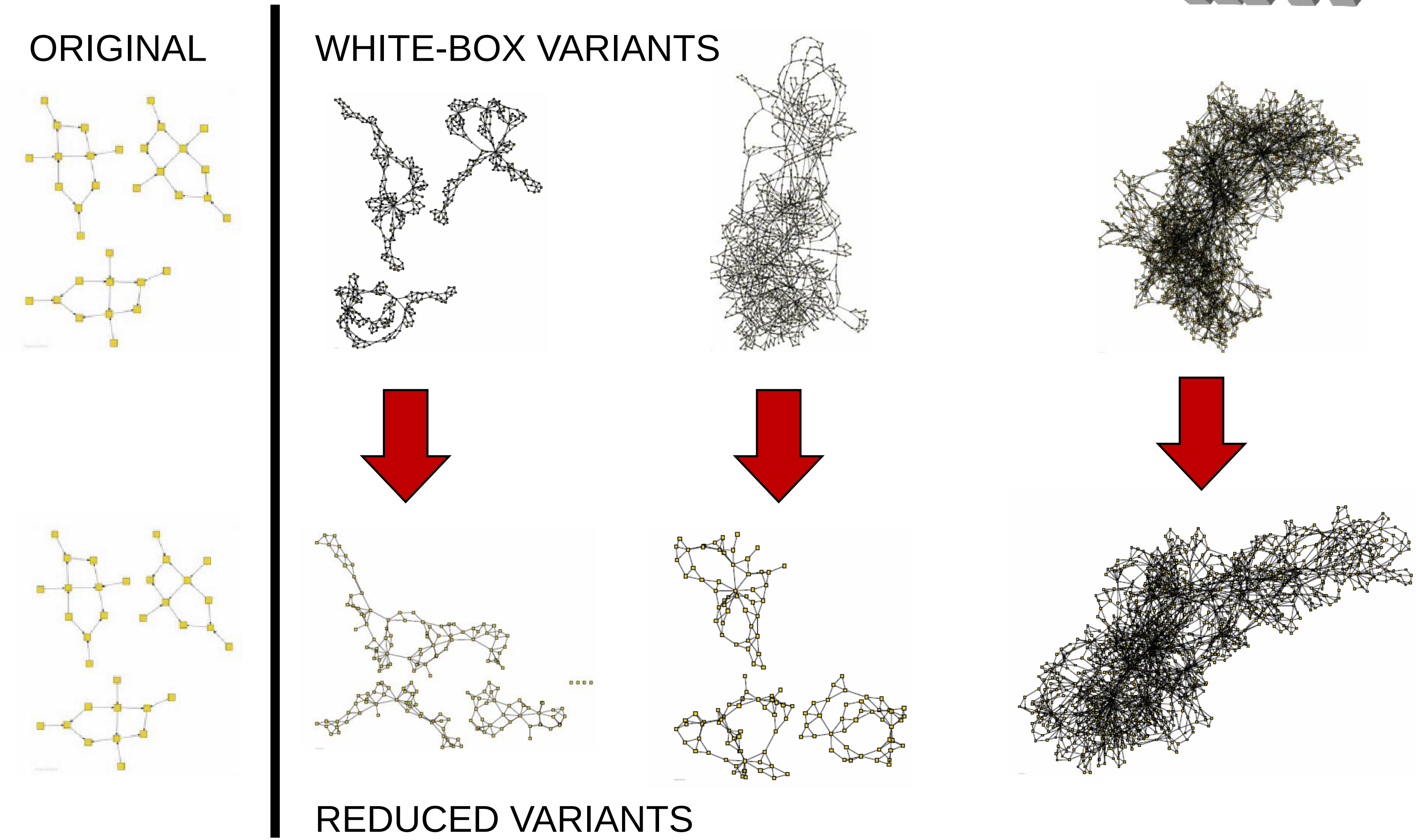
- Components are building block for virtually all real-world circuits
- Given:
  - circuit *C*
  - gate set *G*
  - input set *I*
  - integer $k > 1$, where k is the number of components
- Set *M* of components $\{c_1, \ldots, c_k\}$ partitions *G* and *I* into *k* disjoint sets of inputs and/or gates.
- Four base cases
  - Based on input/output boundary of component and the parent circuit



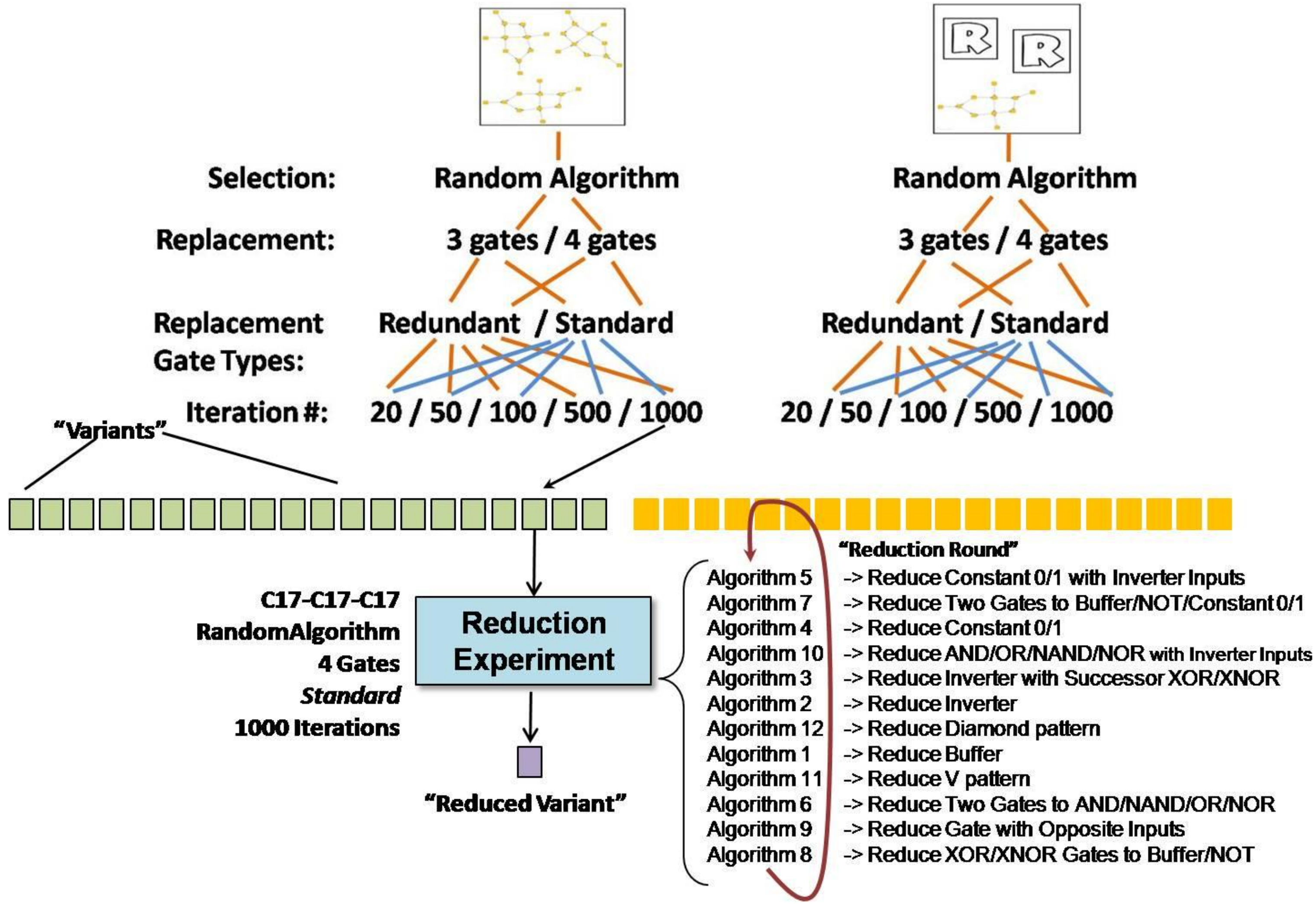
Component Recovery



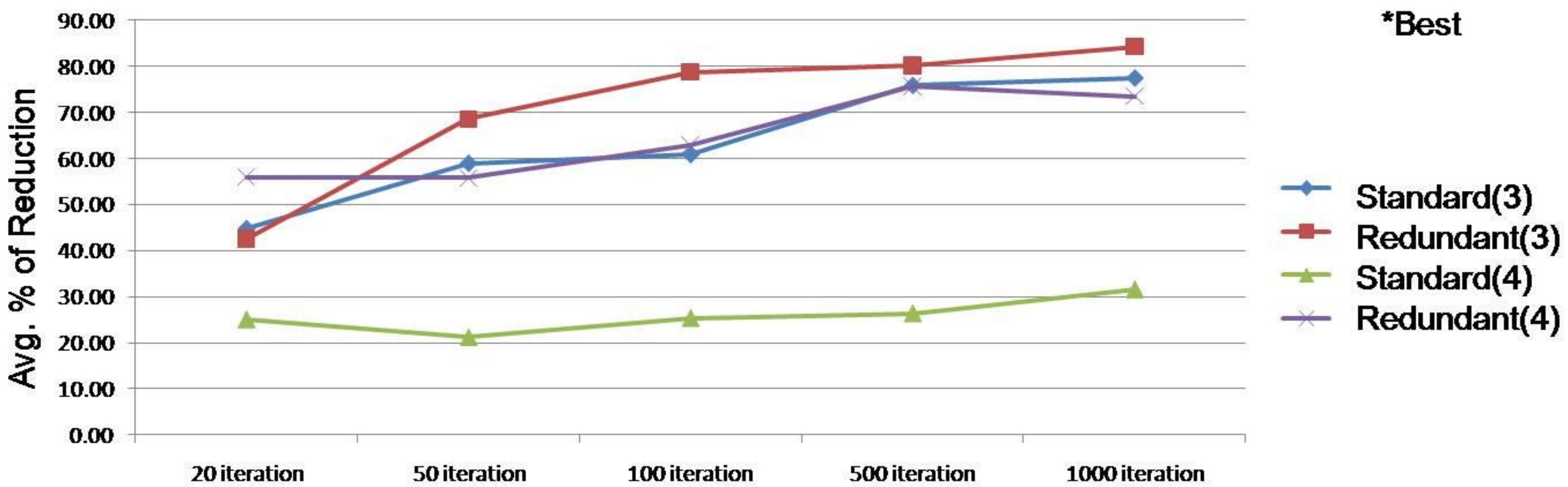
Independent Components and Induced Redundancy



Observing Independent Component Hiding

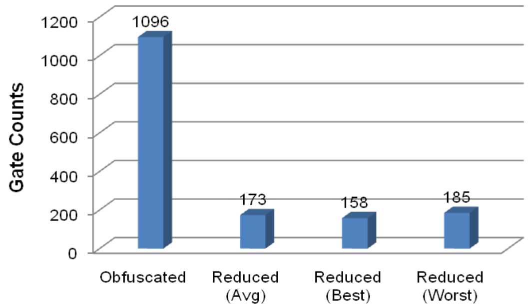


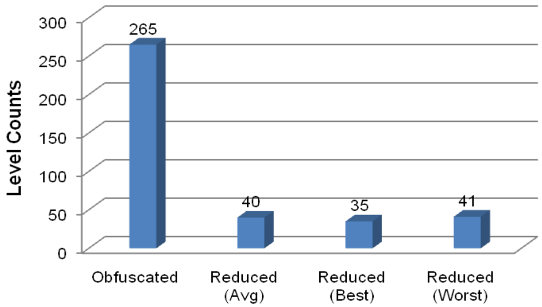
		Experiment Type (gate replacement size)			
		Standard(3)	Redundant(3)	Standard(4)*	Redundant(4)
Iterations	20	44.74 %	42.50	**25.00**	55.93
	50	58.90	68.49	**21.14**	55.74
	100	60.80	78.74	**25.33**	62.88
	500	75.80	80.18	**26.35**	75.62
	1000	77.41	84.22	**31.56**	73.46



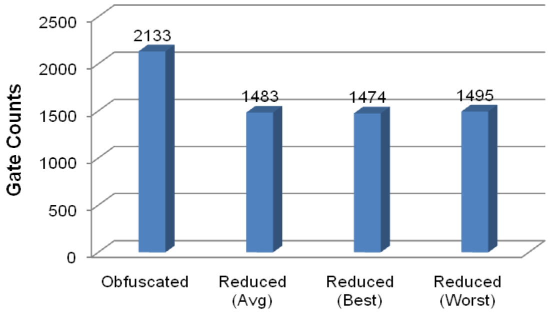
*Best

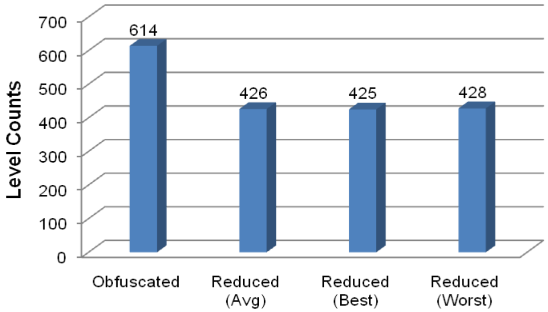
	Variant (Obfuscated)	Reduced (Avg)	Reduced (Best)	Reduced (Worst)
Gates	1096	173 (84.22%)	158 (85.58%)	185 (83.12%)
Levels	265	40 (84.91%)	35 (86.79%)	41 (84.53%)





	Obfuscated	Reduced (Avg)	Reduced (Best)	Reduced (Worst)
Gates	2133	1483 (30.47%)	1474 (30.90%)	1495 (29.91%)
Levels	614	426 (30.62%)	425 (30.78%)	428 (30.29%)





Case Study

	c432-c499			c432-c880			ISCAS Merge			Buffer-100			Buffer-500		
Variant Algorithm	O	S	C	O	S	C	O	S	C	O	S	C	O	S	C
Pattern Based Reduction Size/Levels	-	85%	21-29%	-	63%	22%	-	16-18%	9%	-	90%	28%	-	89%	26%
	-	89%	24-36%	-	72%	24%	-	70%	23%	-	93%	29%	-	92%	28%
Independent Components (pattern-based reduction)	2	2	1	2	2	1	8	1	1	100	59	15	500	253	109
Logic Cells (Quartus II)	133	155	165	173	184	185	1600	1685	nn	0	0	0	xx	xx	xx
Independent Components (as realized by Quartus II)	2	2	2	2	2	2	nn	nn	nn	100	100	100	xx	xx	xx

O – original circuit
S – Simple C - Complex

nn– not tested
xx – too big based on I/O

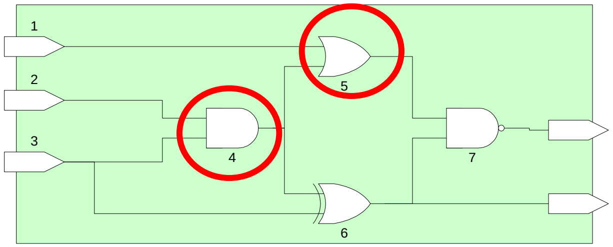
Conclusions

Questions

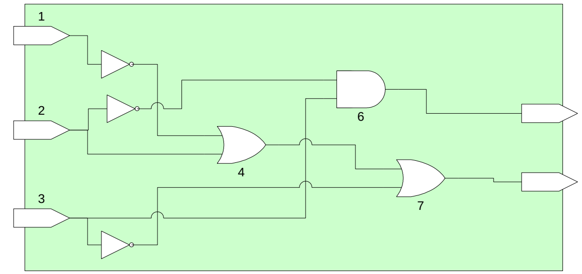
?

Hiding Properties of Interest

General Intuition and Hardness of Obfuscation



The ONLY true "Virtual Black Box"



X1	X2	X3	4	5	Y6	Y7
			AND(3,2)	OR(4,1)	XOR(4,3)	NAND(5,6)
0	0	0	0	0	0	1
0	0	1	0	0	1	1
0	1	0	0	0	0	1
0	1	1	1	1	0	1
1	0	0	0	1	0	1
1	0	1	0	1	1	0
1	1	0	0	1	0	1
1	1	1	1	1	0	1

"The How"

X1	X2	X3	Y6	Y7
			XOR(4,3)	NAND(5,6)
0	0	0	0	1
0	0	1	1	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	1	0
1	1	0	0	1
1	1	1	0	1

Semantic Behavior

Framework and Experimental Results

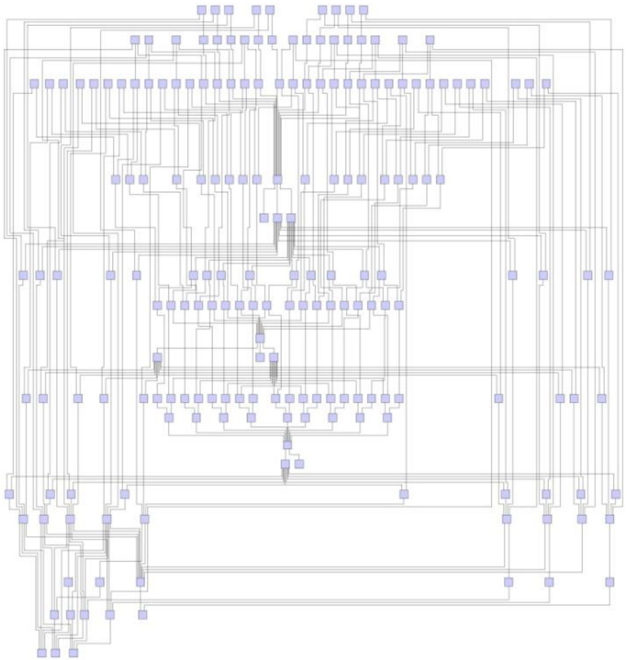
- Is perfect or near topology recovery useful (therefore, is topology *hiding* useful)?
  - In some cases, yes
  - Foundation for other properties (signal / component hiding)
  - For certain attacks, it is all that is required

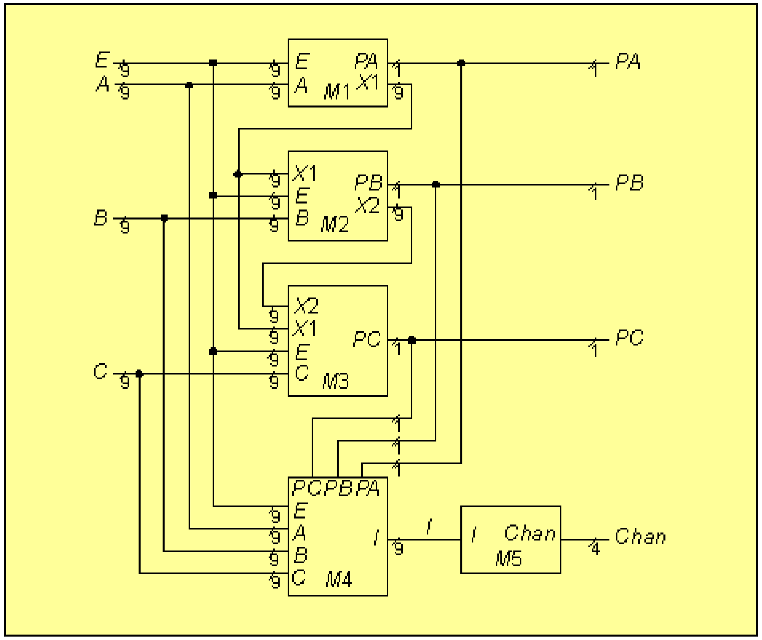
- Accomplishing topology hiding
  - Change basis type (normalizing distributions, removing all original)
  - Guarantee every gate is replaced at least once
  - Multiple / overlapping replacement = diffusion

Topology:
Gate fan-in
Gate fan-out
Gate type

Experiment 1: Measuring “Replacement” Basis Change

c432

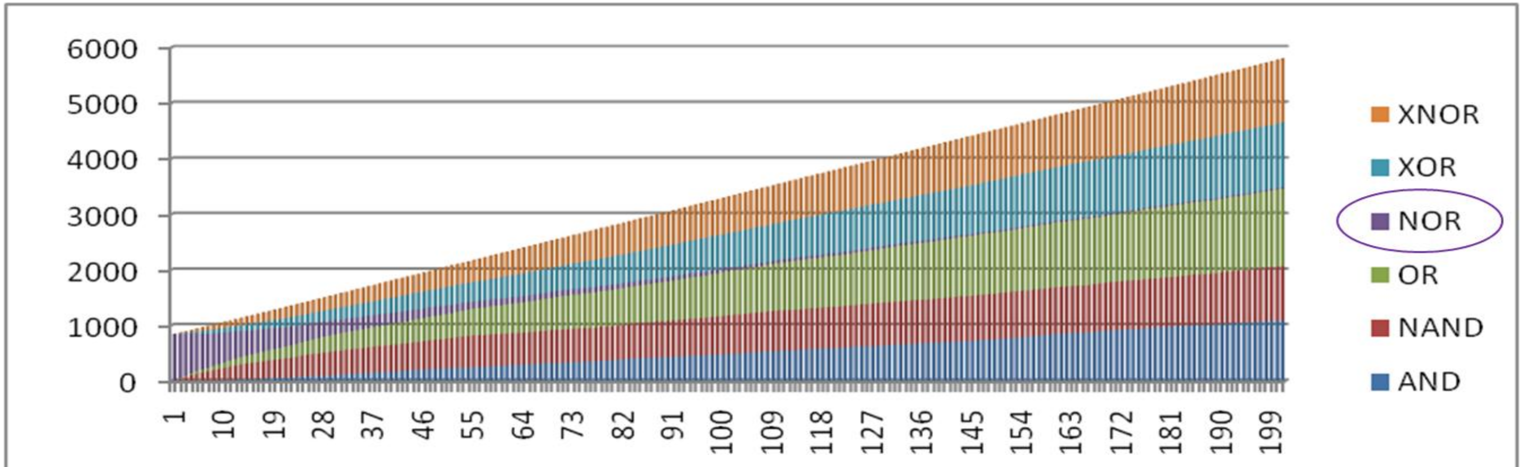


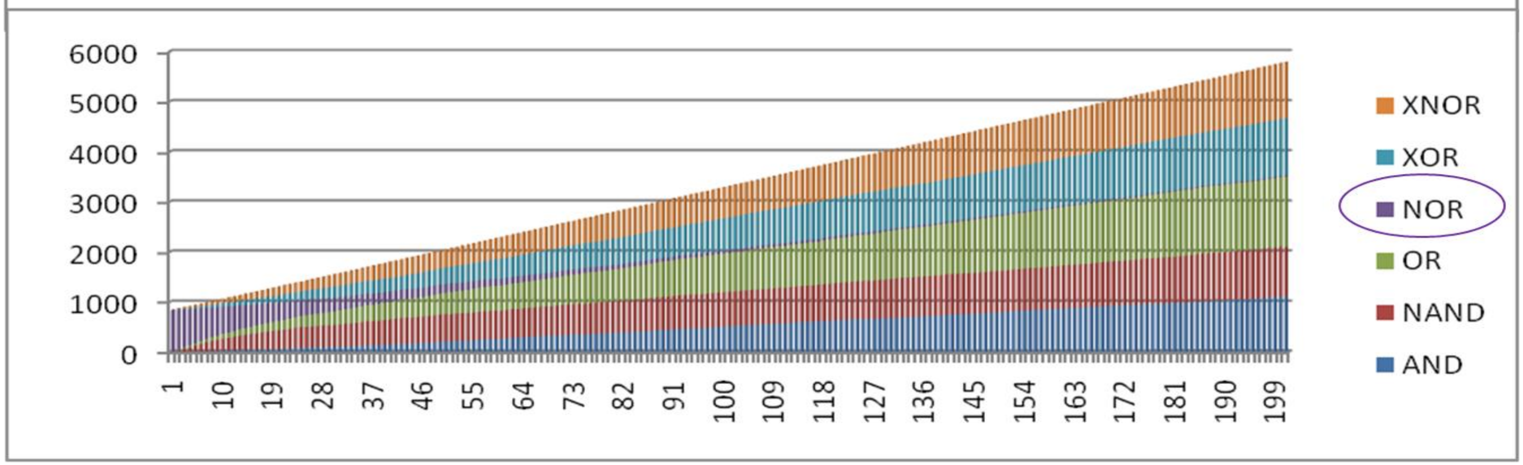


c432	**120 gates (4 ANDs + 79 NANDs + 19 NORs + 18 XORs + 40 inverters)**
Decomposed	**230 gates (60 ANDs + 151 NANDs + 19 NORs + 40 inverters)**
Decomposed NOR	**843 gates (843 NORs)**

Experiment 1a: Measuring “Replacement” Basis Change

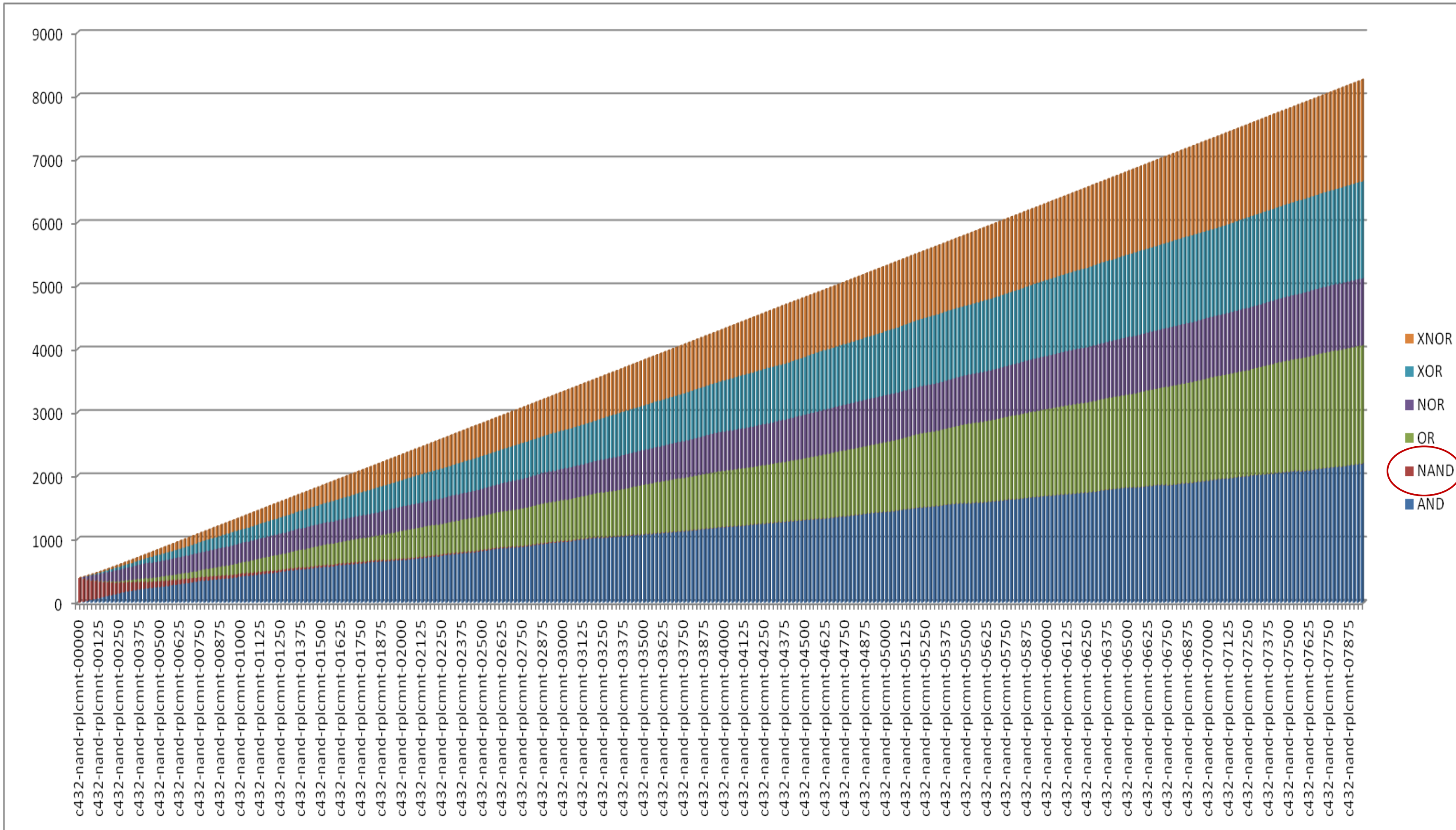
Ω = {NOR} → Ω = {AND, NAND, OR, XOR, NXOR}



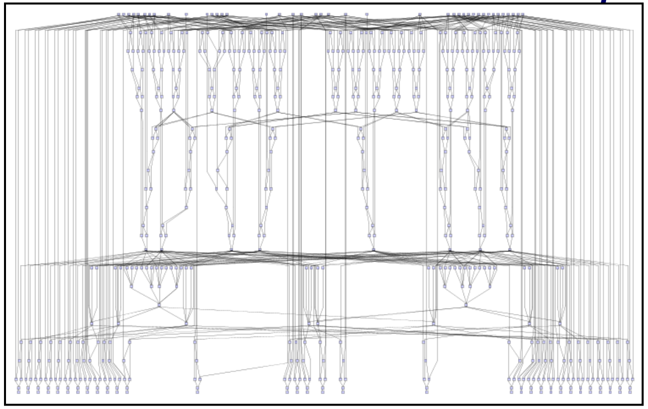


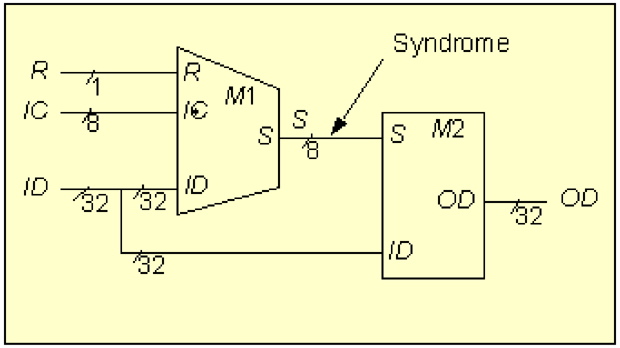
Experiment 1b: Measuring "Replacement" Basis Change

Ω = {NAND} → Ω = {AND, NOR, OR, XOR, NXOR}



Experiment 2: Measuring "Replacement" Uniform Basis Distribution





ISCAS-85 c1355

Iterative Random Selection Algorithm:

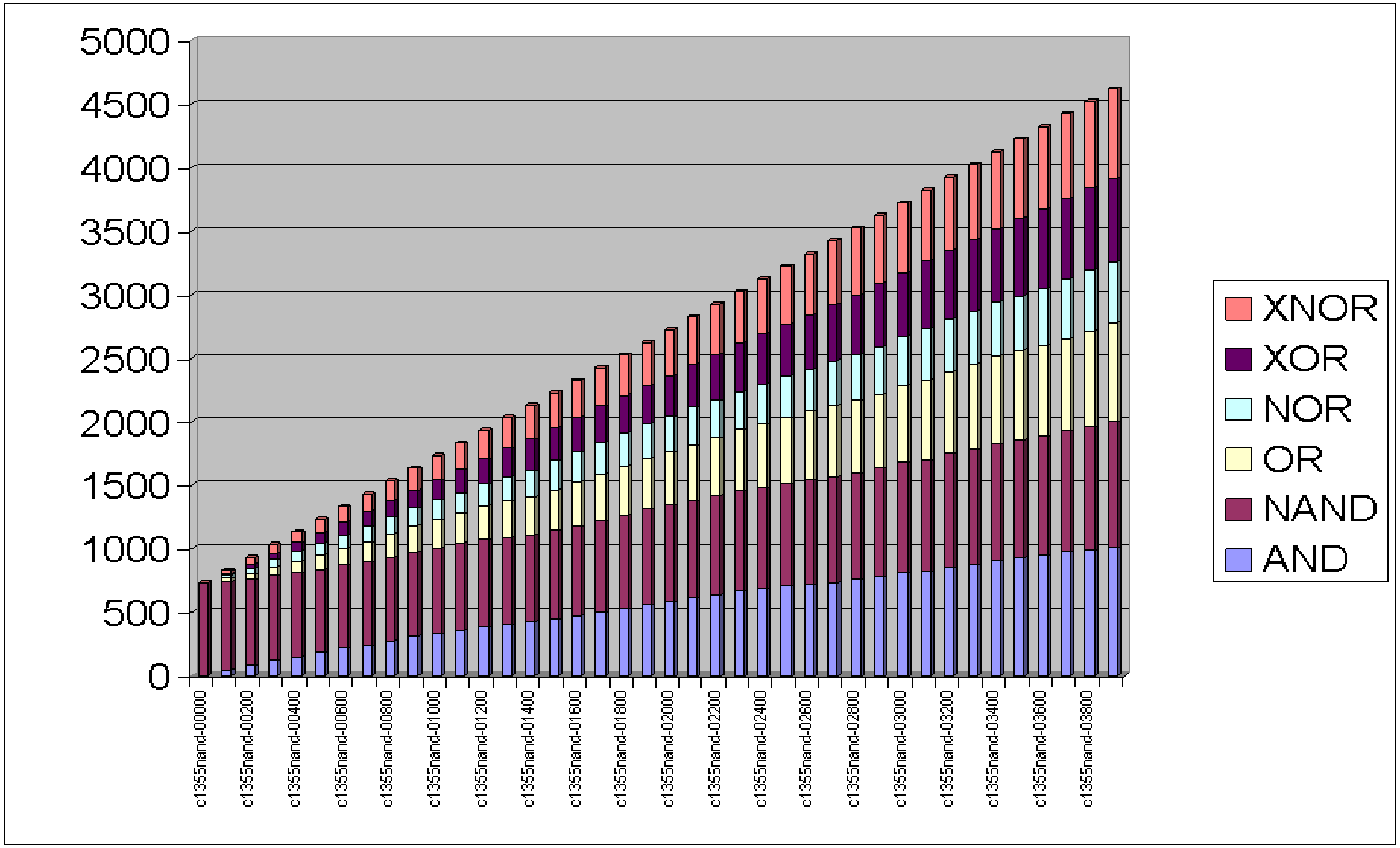
Selection Strategy:
- 5% 1) Single Gate: Random
- 75% 2) Two Gate: Random
- 5% 3) Two Gate: Largest Level
- 5% 4) Two Gate: Output Level
- 5% 5) Two Gate: Random Level
- 5% 6) Two Gate: Fixed Level

Replacement Strategy:
Random 6-GATE Basis

C1355	**506 gates (56 ANDs + 416 NANDs + 2 ORs + 32 buffers + 40 inverters)**
Decomposed	**550 gates (96 ANDs + 416 NANDs + 6 ORs + 32 buffers + 40 inverters)**
Decomposed NAND	**730 gates (730 NANDs)**

Experiment 2: Measuring "Replacement" Uniform Basis Distribution

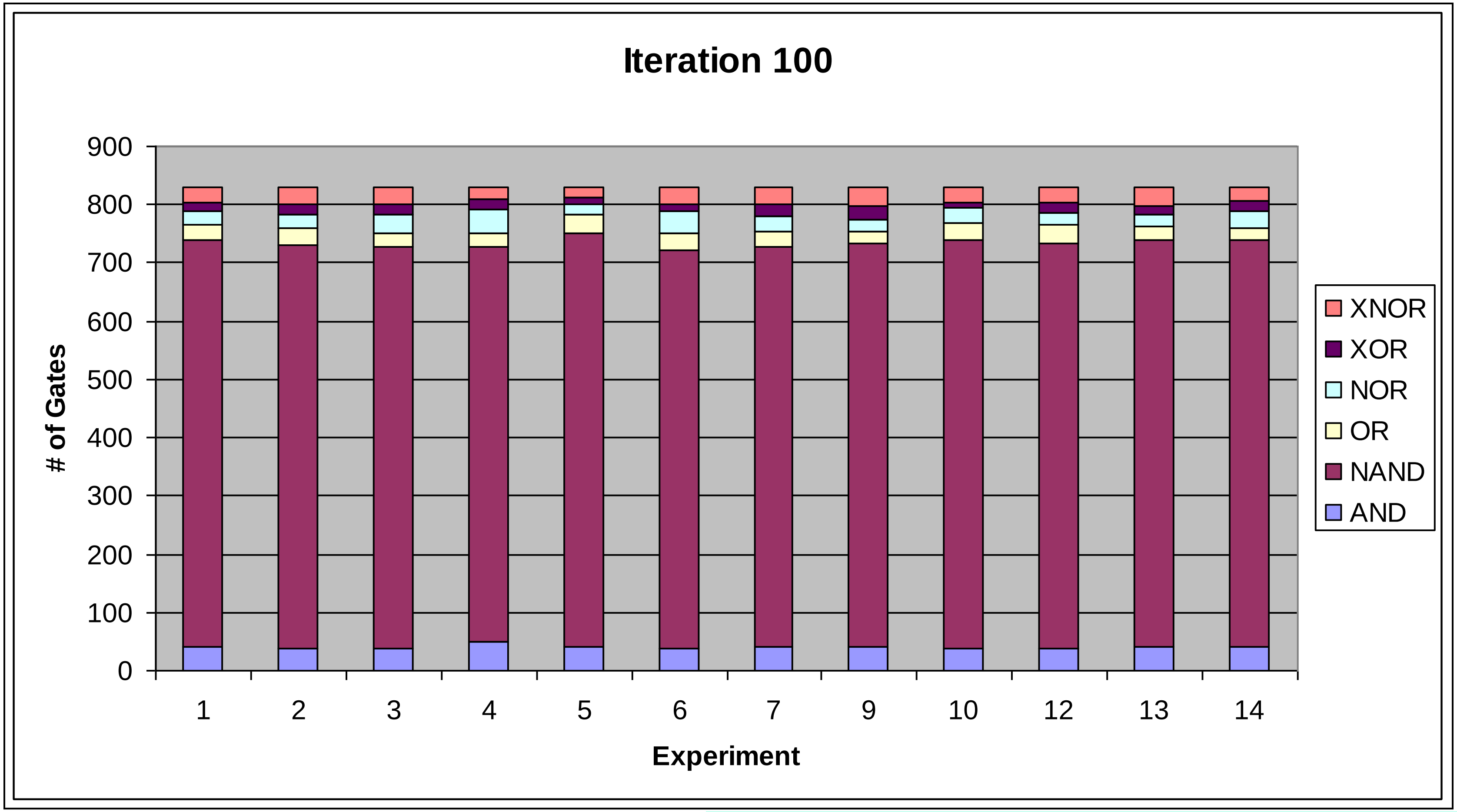
Ω = {NAND} → Ω = {AND, NAND, OR, NOR, XOR, NXOR}



"Single 4000 Iteration Experiment"

Experiment 2: Measuring "Replacement" Uniform Basis Distribution

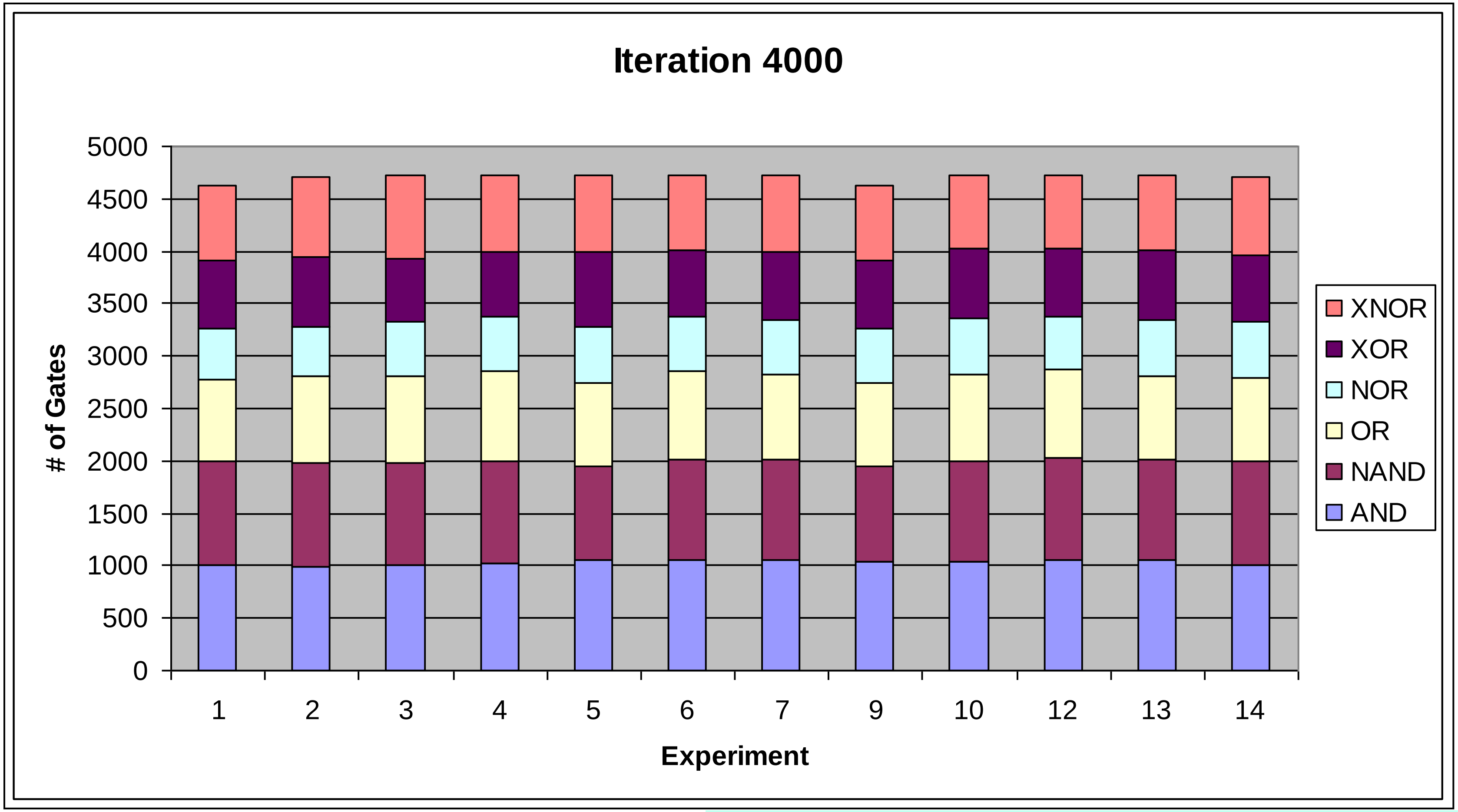
Ω = {NAND} → Ω = {AND, NAND, OR, NOR, XOR, NXOR}



"Multiple 4000 Iteration Experiments"

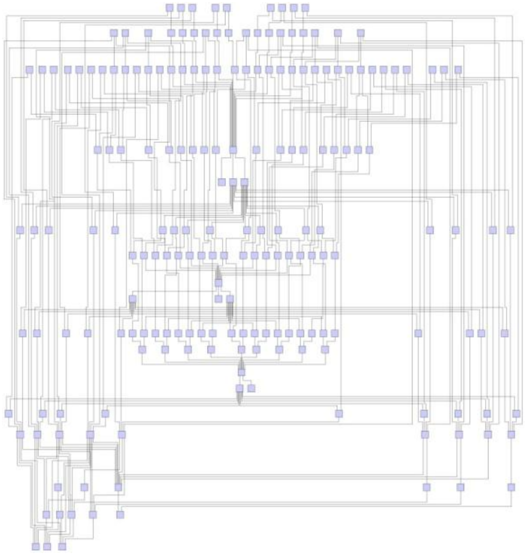
Experiment 2: Measuring "Replacement" Uniform Basis Distribution

Ω = {NAND} → Ω = {AND, NAND, OR, NOR, XOR, NXOR}



"Multiple 4000 Iteration Experiments"

Experiment 3: Measuring "Replacement" Smart Random Selection



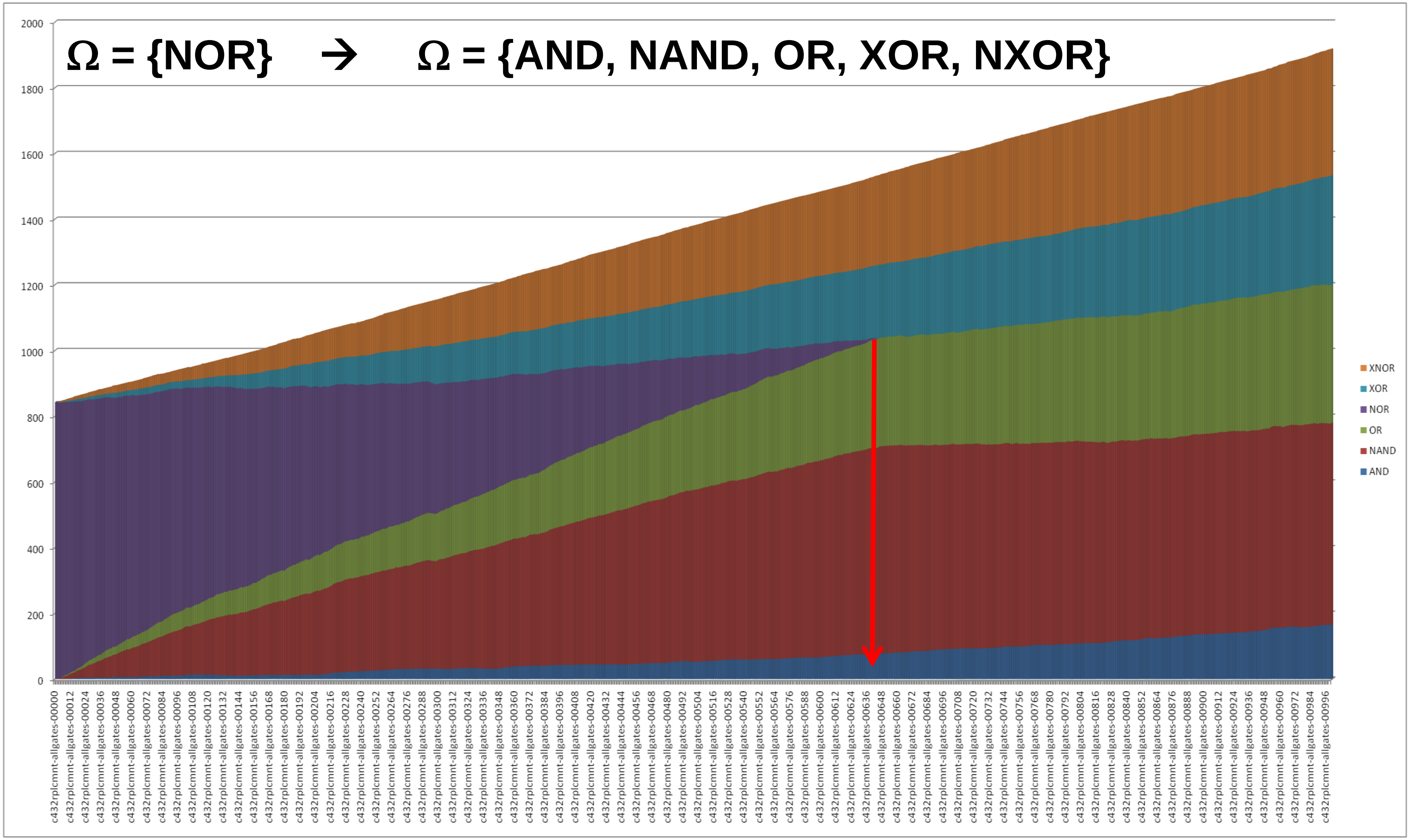
ISCAS-85 c432

Iterative Smart Random 2-Gate Selection Algorithm:

Selection Strategy:
Smart Two Gate Random

Replacement Strategy:
Random Equivalent

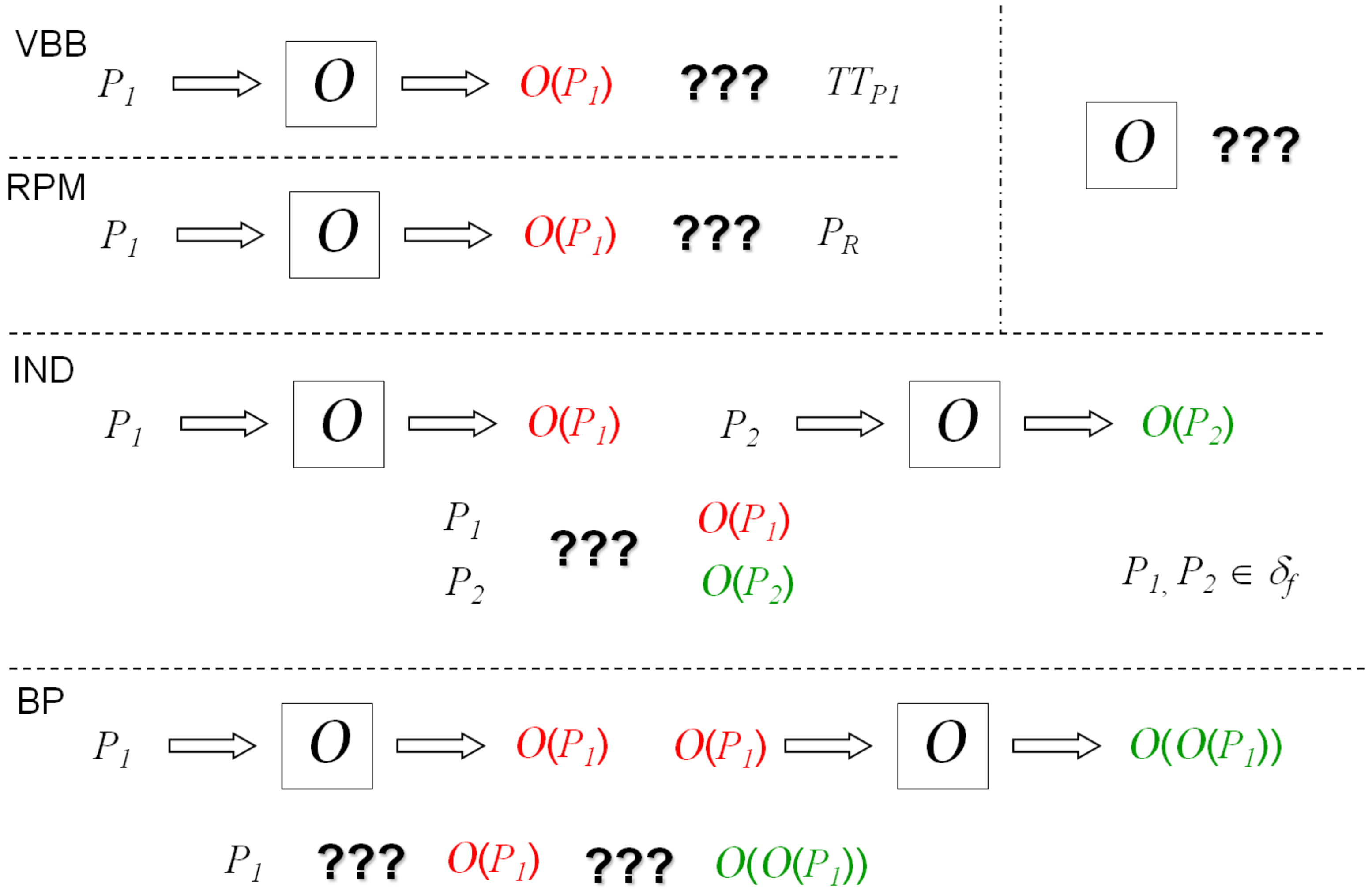
Experiment 3: Measuring “Replacement” Smart Random Selection



Things We've Learned Along the Way

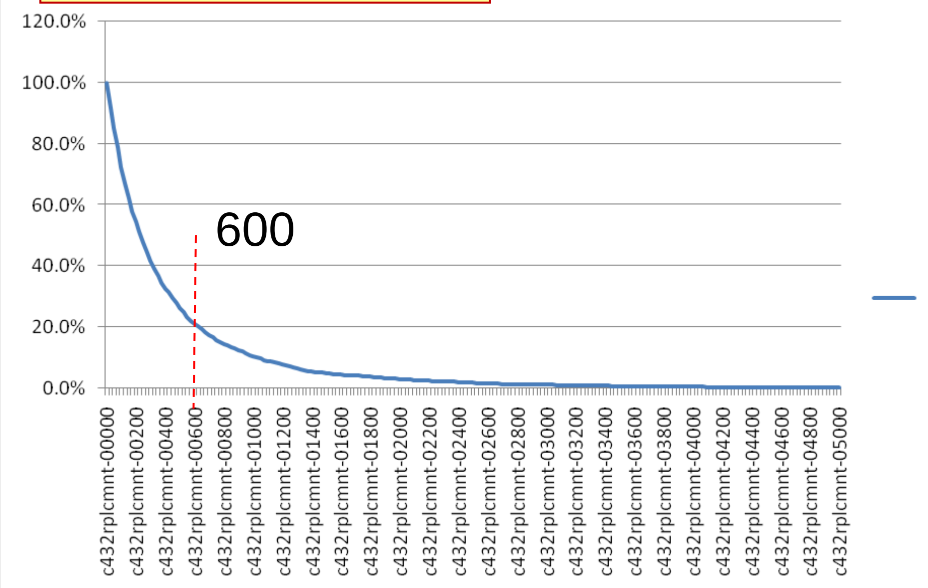
- What algorithmic factors influence hiding properties the most?
  - Iteration number
  - Selection size
  - Replacement circuit generation (redundant vs. non-redundant)
- Ongoing work in:
  - Increasing selection size
  - Determinist generation
  - Integrated logic reduction
  - Formal models: term rewriting systems, abstract interpretation, graph partitioning

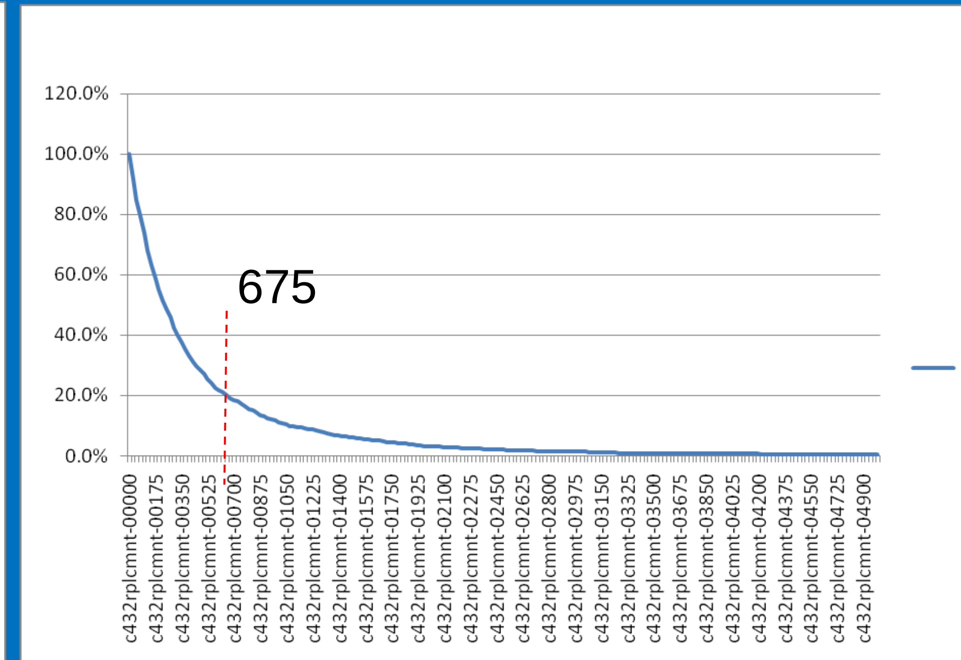
Obfuscation Comparison Models

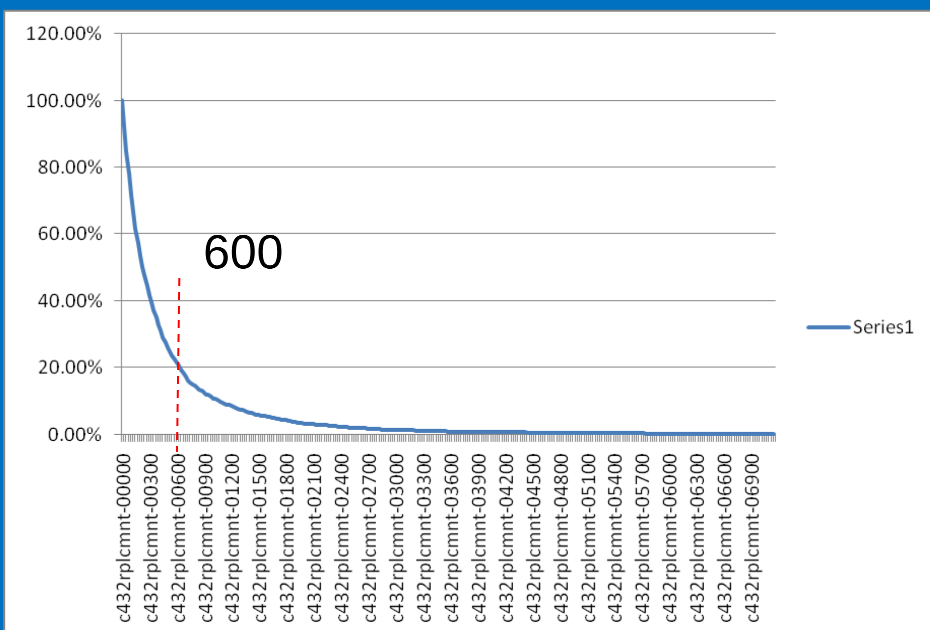


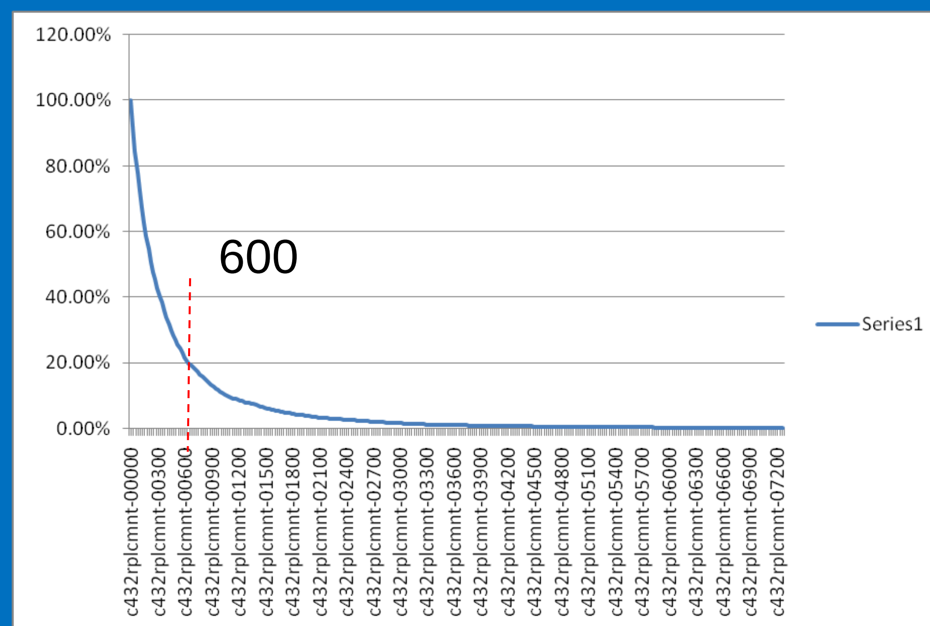
Experiment 1a: Measuring

% of ORIGINAL GATES



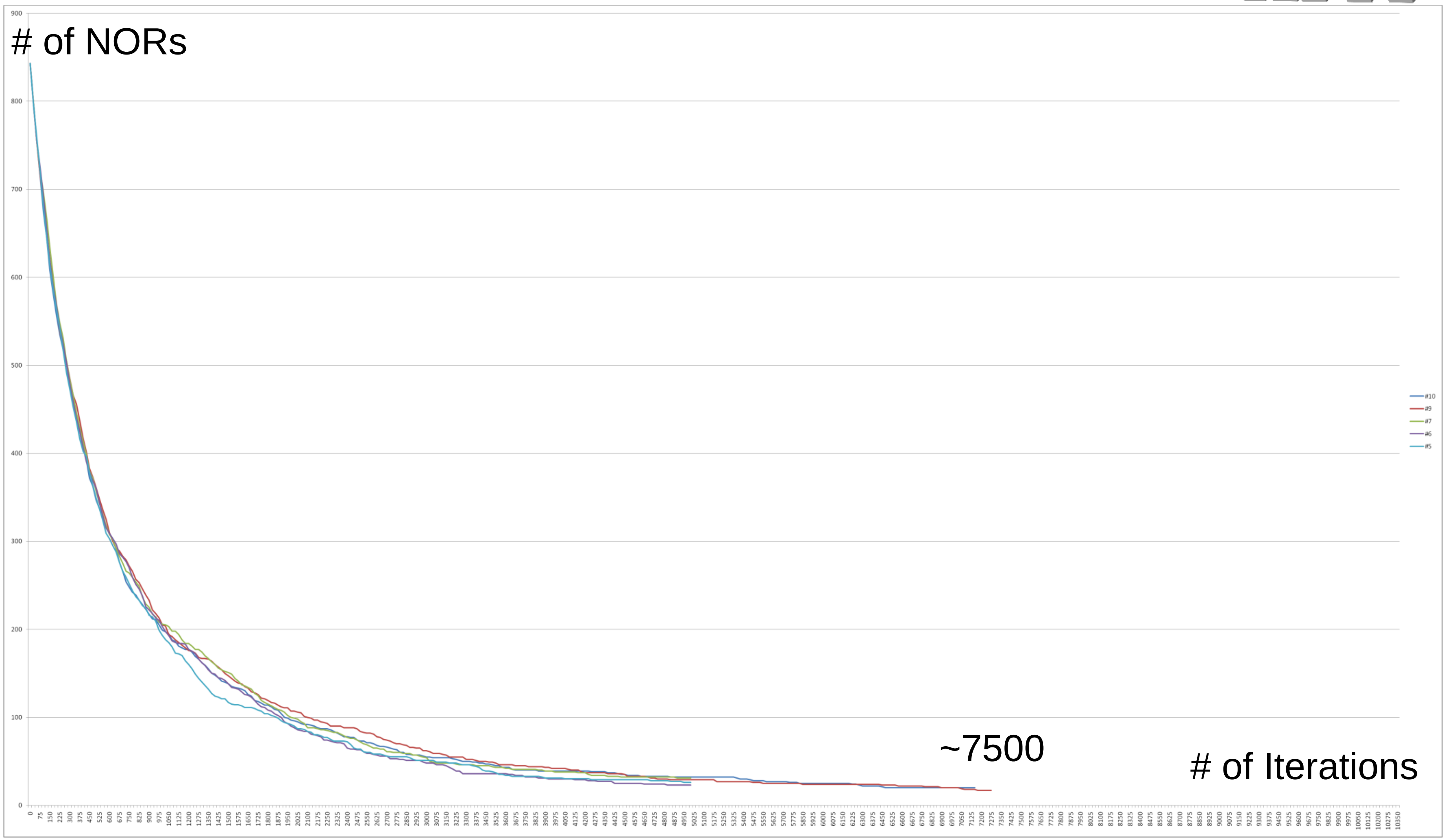






Experiment 1a: Measuring “Replacement”

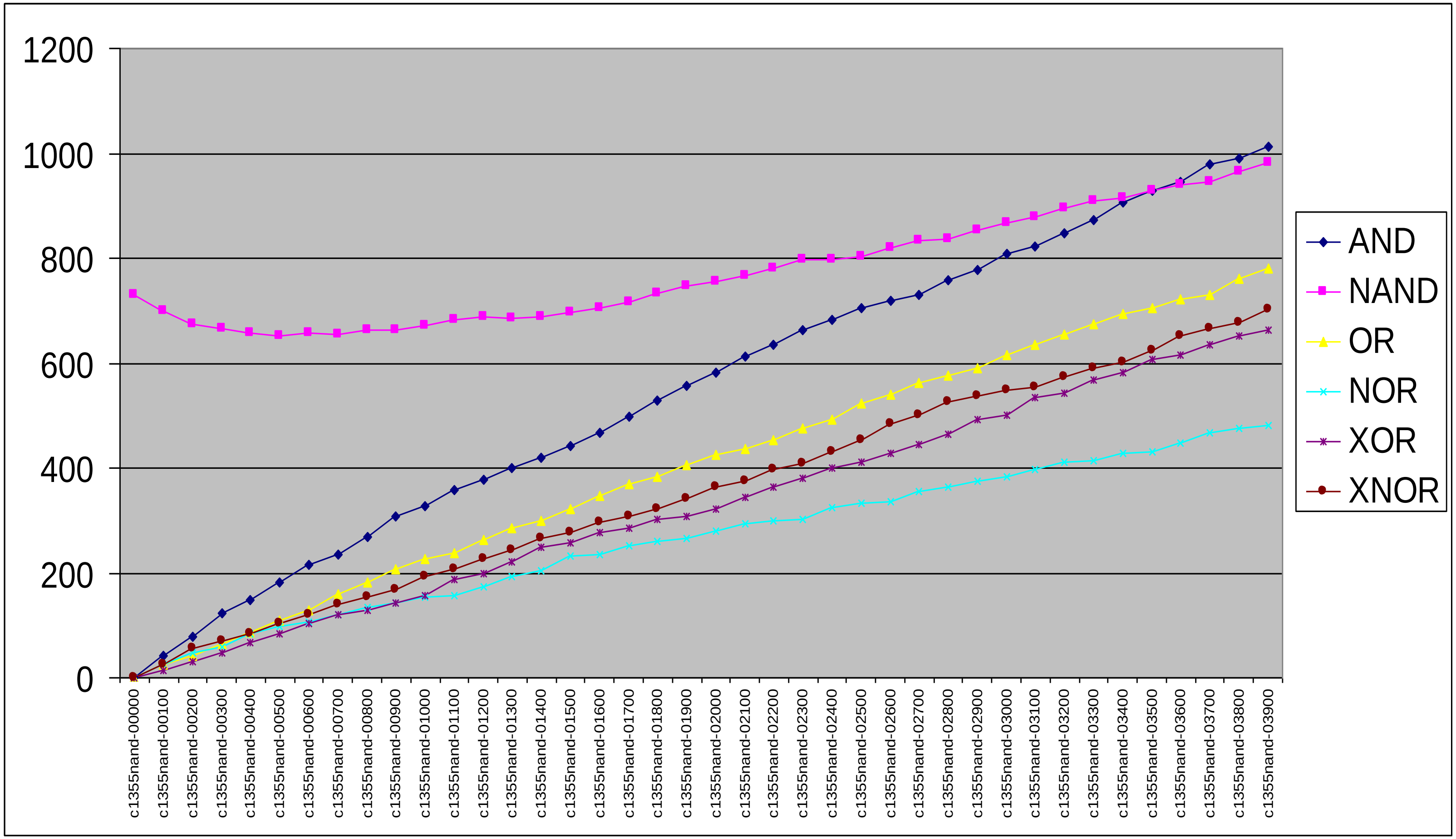
Ω = {NOR} → Ω = {AND, NAND, OR, XOR, NXOR}



ISCAS-85 c1355

Experiment 2: Measuring “Replacement”

Ω = {NAND} → Ω = {AND, NAND, OR, NOR, XOR, NXOR}

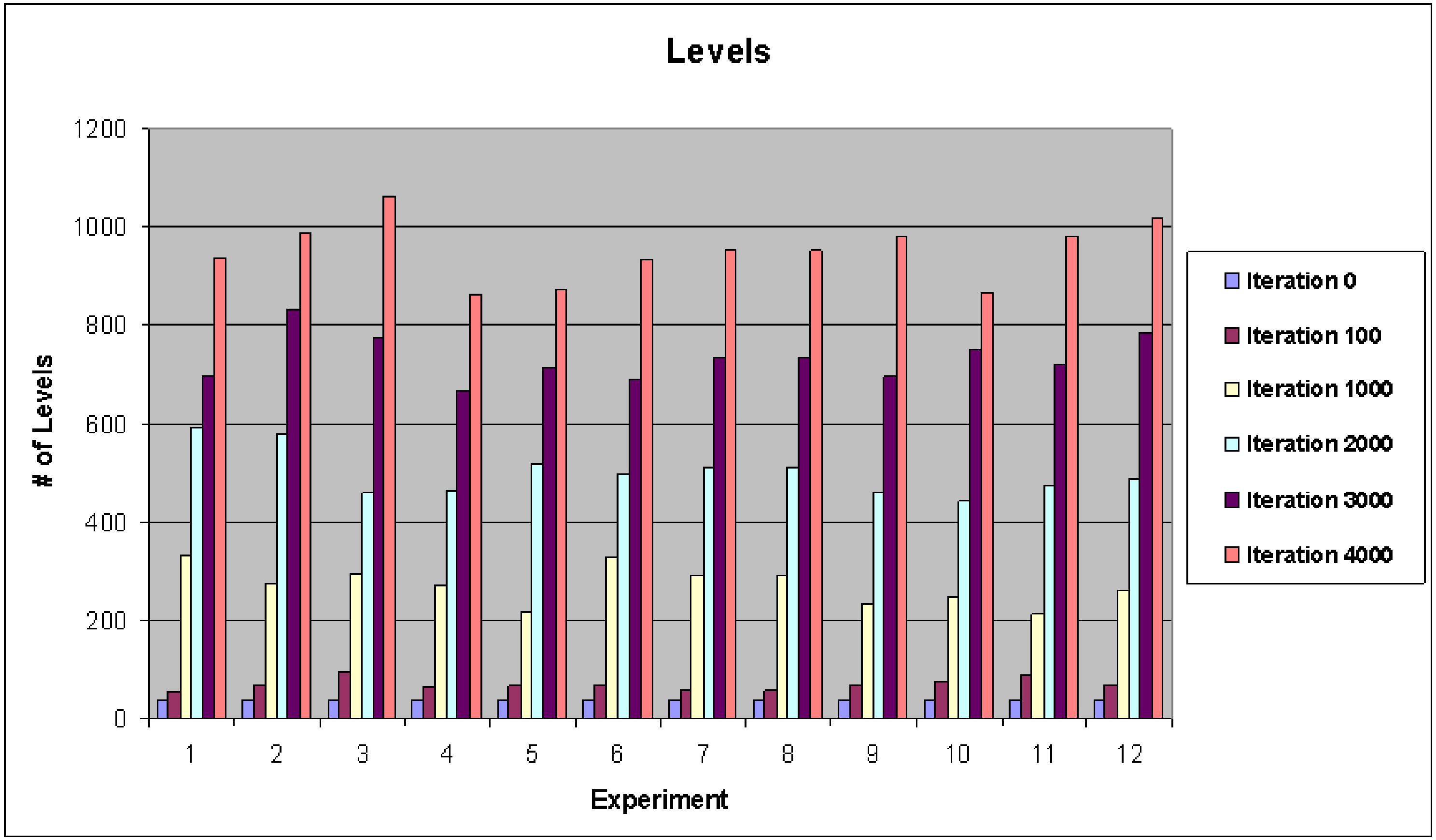


“Single 4000 Iteration Experiment”

Experiment 2: Measuring "Replacement"

Develop America's Airmen Today ... for Tomorrow

Ω = {NAND} → Ω = {AND, NAND, OR, NOR, XOR, NXOR}



"Multiple 4000 Iteration Experiments"