3-26-2009

Removing Redundant Logic Pathways in Polymorphic Circuits

Hanseok Kim

Removing Redundant Logic Pathways in Polymorphic Circuits

THESIS

Hanseok Kim, Captain, ROKA

AFIT/GCS/ENG/09-03

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GCS/ENG/09-03

Removing Redundant Logic Pathways in Polymorphic Circuits

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Hanseok Kim, BSCE

Captain, ROKA

26 March 2009

AFIT/GCS/ENG/09-03

Removing Redundant Logic Pathways in Polymorphic Circuits

Hanseok Kim, BSCE

Captain, ROKA

Approved:

Lt Col J. Todd McDonald, Ph.D. (Chairman) — 13 MAR 09 — date

Dr. Yong C. Kim (Member) — 13 MAR09 — date

Lt Col Stuart H. Kurkowski, Ph.D. (Member) — 13 Mar 09 — date

AFIT/GCS/ENG/09-03

Abstract

Evaluating the quality of software and circuit obfuscators is a research goal of great interest. However, there exists little research about evaluation of obfuscation effectiveness through analyzing and investigating redundancies found in the obfuscated variants.

In this research, we consider programs represented as structural combinational circuits and then analyze obfuscated variants of those circuits through a tool that produces functionally equivalent variants based on subcircuit selection and replacement. We then consider how Boolean logic and reduction affects the size and levelization of circuit variants, giving us a concrete metric by which to consider obfuscation effectiveness. To accomplish these goals, we create an experimental environment based on a set of predefined circuits, a set of predefined algorithms which produce variants of those circuits, and a collection of logic reduction techniques and tools. We build logic reduction techniques using predefined patterns and predefined functions expressed as truth tables.

As a contribution, we characterize and evaluate the effectiveness of obfuscating algorithms based on these reduction techniques. We show, for the circuits we observe, optimization on size is affected by ordering of the specific reduction patterns and functions. We also show, for the circuits we observe, reduction is affected by the specific obfuscating algorithm used to produce the variant. Based on these results, we provide a promising measurement of interest to compare both circuit variants and obfuscating algorithms.

Acknowledgements

I would first like to express my sincere appreciation to my thesis advisor, Lt Col. Todd McDonald, for his guidance and support throughout the course of this thesis effort. I would also like to thank Dr. Yong C. Kim for his advice and assistance about circuit domain. I could not have completed my work without their support and concerns.

I would like to thank my academic advisor, Lt Col. Stuart Kurkowski, for showing me the right path to finish the Computer Science Master's Degree program at the Air Force Institute of Technology. I would also like to thank IMSO director, Mrs. Annette Robb, for enthusiastically providing me with various assistance needed as international military student.

Most importantly, I would like to thank my loving mother for all her support and for encouraging me to finish my education successfully.

Hanseok Kim

Table of Contents

List of Figures

Figure Page

Figure Page

List of Tables

List of Abbreviations

Abbreviation		Page
CORGI	Circuit Obfuscation via Randomization of Graphs Iteratively	2
VBB	Virtual Black Box	5
PPT	Probabilistic Polynomial time Turning machine	5
ISCAS	International Symposium of Circuits and Systems	8
IEEE	Institute of Electrical and Electronics Engineers	8

Removing Redundant Logic Pathways in Polymorphic Circuits

I. Introduction

1.1 Background

Software obfuscation remains a research area of great interest. If possible, proving or demonstrating that we can protect software from adversarial actions holds great interest to both military and civilian communities. In general, obfuscation seeks to find suitable variants of an original program or circuit that accomplishes the same function as the original. The suitability of the variant is determined by whether or not the variant demonstrates some security property of interest [8]. In most cases, we link the security property to driving up the cost of reverse engineering or demonstrating that recovery of certain program information is equivalent to the work of solving a known hard problem [6, 7, 13].

If we consider simple straight-line programs represented as combinational circuits, we can focus more precisely on what actually defines obfuscation and consider measurements of those attributes. In this research, we experiment with and characterize a polymorphic circuit generator that creates functionally equivalent circuits by using a mixture of random and deterministic subcircuit selection/replacement algorithms applied sequentially. Using this approach, such a generator, when given an original circuit P, produces a circuit P' with the same function by using a sequence of iterative selections and replacements on circuit P. Each individual selection and replacement represents an obfuscation on a small scale: the engine replaces each subcircuit selected with a functionally equivalent replacement subcircuit.

Our research focuses on how to characterize and measure circuit obfuscation that uses such selection/replacement algorithms. Specifically, we analyze the nature of the structural and logic changes produced by subcircuit selection/replacement. Based on the logic and component properties of the selection and replacement operations themselves, we characterize specific trends that occur within a circuit that undergoes transformation. A question of great interest is ”how much” of the transformations introduced via subcircuit selection and replacement algorithms can be “undone” via optimization or logic minimization. The answer to this question possibly provides a measurable basis to characterize the degree of “actual” obfuscation that is taking place. By looking at the effect of subcircuit transformations themselves, we formulate an approach for answering this question and organize our research accordingly.

1.2 Research Objectives

Based on these goals, we describe the main objectives of our research as follows:

1. Produce and analyze obfuscated variants of combinational circuits created using iterative subcircuit selection and replacement.
2. Devise methods to remove identifiable artifacts of circuit variants based on logic and component analysis.
3. Characterize circuit obfuscation effectiveness using reduction methods and determine principles of successful deobfuscation.

To accomplish these objectives, we utilize a polymorphic circuit generator (CORGI, Circuit Obfuscation via Randomization of Graphs Iteratively) developed by McDonald *et al.* [14–16, 18, 21] known as *CORGI*. The whitebox obfuscation aspect of the *CORGI* framework specifically uses subcircuit selection and replacement as its transformation basis.

To accomplish our first objective, we generate the obfuscated circuit variants using specific generation options defined by *CORGI*. We then use the circuit transformation history (or journal) produced during a *CORGI* experiment to identify obfuscation patterns that occur. Using these patterns, we create a strategy for reduction based on the logic attributes associated with each pattern.

For the second objective, we define patterns that have specific logic that is conducive to reduction or minimization. Classic examples of such reduction may be seen in whitebox circuit structures such as buffers, double inverters, constant 0/1 gates, and so forth. Using this approach, we devise ten circuit reduction algorithms which represent the pattern-based approach of logic minimization. We also create two reduction algorithms that use functional identification (based on the truth table) to find logic structures suitable for minimized replacement. These algorithms try to find whitebox structures whose functions are simple AND, OR, $NAND$, and NOR logic patterns.

For the last objective, we characterize obfuscation by using these circuit reduction techniques under various experimental conditions. We consider the effect of using these patterns in a combined fashion based on either predetermined order or random sequences and present conclusions based on our observations. We analyze our results to see how size correlates with various approaches to producing the circuit variant (the obfuscating algorithm). Ultimately, we provide results which give definable measures for the effectiveness of obfuscation and give a basis for comparing obfuscating algorithms themselves.

1.3 Organization

We organize the remainder of the thesis as follows. Chapter II provides an overview of reverse engineering and obfuscation and explains the *CORGI* circuit generation engine. Chapter III defines our methodology including analysis of circuit obfuscation log files and creation of circuit reduction algorithms. Chapter IV presents the results of experiments using circuit reduction techniques and our analysis. Chapter V gives our conclusions along with our contributions and discussion of future work for our reduction techniques.

II. Literature Review

This chapter presents a background review of the literature pertaining to the research subjects related to this research work. Section 2.1 describes the concept of reverse engineering. Section 2.2 summarizes theoretical obfuscation with the description of virtual black box obfuscation, indistinguishability obfuscation, and best-possible obfuscation. Section 2.3 shows that the overview of *CORGI*, Circuit Obfuscation via Randomization of Graphs Iteratively [18].

2.1 Reverse Engineering

Aronson, R.B. [1] defines reverse engineering as

> “the process of systematic evaluation of a product for the purpose of replication.”

Chikofsky and Cross [5] describes reverse engineering as

> “the process of analyzing a subject system to identify the system’s components and their interrelationships and create representations of the system in another form or at a higher level of abstraction.”

As we can see from these definitions, reverse engineering can be used to reproduce commercial or military systems by analyzing them. Since it makes possible to reduce cost and time, there are many researches and attempts to reverse engineering for their advantages. Reverse engineering is used in many situations such as when the original manufacturer of a product no longer produces a product, when there is inadequate documentation of the original design, and when the original manufacturer no longer exists, but a customer needs the products [2]. In contrast to these beneficial motivations, reverse engineering can be exploited with negative objectives such as copying other nations’ technologies, devices or information that have been obtained by the industrial spy. Obfuscation is defense against the malicious exploitations of reverse engineering.

2.2 Obfuscation

Meriam-Webster Dictionary [17] presents obfuscate as

> "to make obscure" or "to be evasive, unclear, or confusing"

Wiktionary Dictionary [19] describes obfuscation as

> "The option to alter computer code, preserving its behavior but concealing
> its structure and intent"

In other words, obfuscation provides an ability to hide system structure and intent in order to make other people confused. Barak et al. [3] introduced the first formalized publication of program (or circuit) obfuscation. They claim that an obfuscator must have the following property:

1. **functionality property**, For every program P, $O(P)$ describes a program that compute the same function as P.
2. **polynomial slowdown property**, The size and run time of $O(P)$ are at most polynomial larger than that of P. That is, there is a polynomial p such that for every program P, $|\mathcal{O}(P)| \leq p(|P|)$.
3. **virtual black-box (VBB) property**, For any probabilistic polynomial time turning machine (PPT) A, there is a PPT S and a negligible function α such that for all programs P,

$$|\Pr[A(\mathcal{O}(P)) = 1] - \Pr[S^P(1^{|P|}) = 1]| \leq \alpha(|P|) \tag{2.1}$$

We say that the obfuscator $\mathcal{O}$ is efficient if it runs in polynomial time.

Unfortunately, Barak, et al. [3] proved that it is impossible to construct a universal obfuscator under such requirements. In terms of results, there exists a set of programs that cannot be obfuscated under the VBB definition. As an alternative to VBB obfuscation, Barak et al. propose the idea of indistinguishability obfuscation.

2.2.1 Indistinguishability Obfuscation. We define an indistinguishability obfuscator in the same way as a circuit obfuscator, without replacing the "virtual black box" property with the following:

- **indistinguishability property**, For any PPT A, there is a negligible function α such that for any two circuits C_1, C_2 which compute the same function and are of the same size k,

$$|\Pr[A(\mathcal{O}(C_1))] - \Pr[A(\mathcal{O}(C_2))]| \leq \alpha(k) \tag{2.2}$$

As we can know in VBB obfuscation, the VBB property compares an obfuscated circuit to a simulator that has only black box access to the original circuit. However, the indistinguishability property compares the obfuscations of two different circuits, C_1 and C_2. Since this property weakens the VBB definition, it demonstrates that obfuscation (at least in the information theoretic sense) is not achievable but may be achievable under other definitions that are of interest.

2.2.2 Best-Possible Obfuscation. In another study, Goldwasser and Rothblum [9] clarify the notion of best-possible obfuscation that guarantees that when an obfuscated program $O(P)$ leaks some information, any other program P which computes the same functionality also leaks the same information. The best-possible obfuscation is defined in the same way as circuit obfuscator, without substituting the "virtual black box" property by the following:

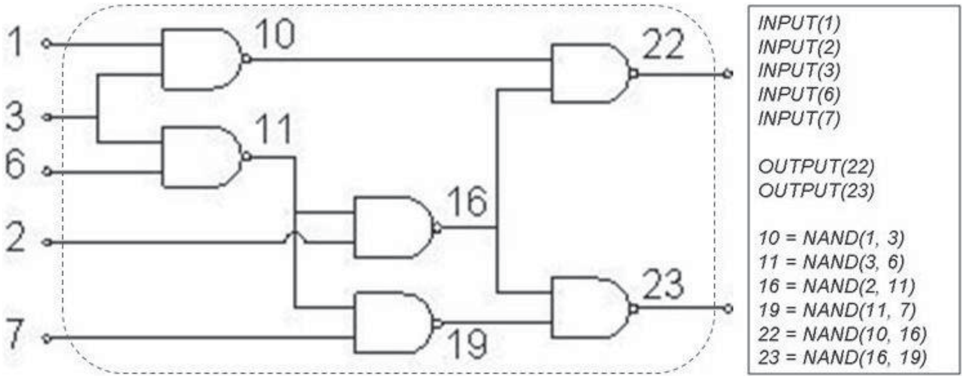
- **Computational/Statistical/Perfect Best-Possible Obfuscation**, For all large enough input lengths, for any polynomial size circuit adversary A, there exists a polynomial size simulator circuit S such that for any circuit $C_1 \in C_n$ and for any circuit $C_2 \in C_n$ that computes the same function as C_1 and such that $|C_1| = |C_2|$, the two distributions $A(O(C_1))$ and $S(C_2)$ are computationally/statiscally/perfectly indistinguishable.

Goldwasser and Rothblum [9] prove that all best-possible obfuscators are indistinguishability obfuscators. They also show that an efficient best-possible obfuscator is an efficient indistinguishability obfuscator. Unfortunately, there is no feasible obfuscator including the indistinguishability obfuscation or best-possible obfuscation concepts until now.

2.3 CORGI

Many obfuscators are widely available through both commercial and open source products that replace an original program with an obfuscated program. To facilitate research aimed at defining the fundamental nature of whitebox obfuscation in the circuit realm, McDonald *et al.* developed a circuit generating algorithm known as *CORGI*, Circuit Obfuscation via Randomization of Graphs Iteratively, at the Air Force Institute of Technology (AFIT). *CORGI* operates on programmatic logic embodied in Boolean logic circuits using iterative sequences of probabilistic and deterministic transforms. *CORGI* analysis primarily revolves around running experiments on some original combinational logic circuit and producing one or more variants under a set of experimental options. Analysis typically considers whether CORGI produces a distribution of circuits with a specific (hiding) property or characteristic of interest. Such experiments give basis for either characterizing one algorithm choice over another or with defining metrics that reflect security properties of interest. The major components of a *CORGI* experiment are circuit representation, experimental configuration, subcircuit selection, and subcircuit replacement. We give a brief summary of these components as follows:

2.3.1 Circuit Representation. While sequential circuits have memory and feedback loops (cycle), combinational circuits have no state. Since we can decompose sequential circuits into combinational circuits, *CORGI* considers only combinational logic. Independent observations by Huth and Ryan [11] also point out that we only need a simple grammar to compute everything that can be computed by large languages like *C* and *Java*. As another benefit, a simple logic grammar provides understandability because we need only three logic functions: NOT(!), AND(&), and OR(||). There are other commonly used logic functions such as $NAND$, NOR,



```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(6)
INPUT(7)

OUTPUT(22)
OUTPUT(23)

10 = NAND(1, 3)
11 = NAND(3, 6)
16 = NAND(2, 11)
19 = NAND(11, 7)
22 = NAND(10, 16)
23 = NAND(16, 19)
```

Figure 2.1: ISCAS Benchmark Circuit c17

XOR, and $XNOR$, but we can represent these functions using various combinations of NOT, AND, and OR.

Combinational logic circuits are widely used within both the hardware and software domain. At the 1985 International Symposium of Circuits and Systems (ISCAS), the Institute of Electrical and Electronics Engineers, Inc. (IEEE) introduced a set of benchmark circuits, ISCAS-85 benchmark circuits [10]. They are particularly useful for our research because they embody a variety of functions with a varying degree of input size, output size, and circuit size. A list of these circuits can be found at [4]. The smallest circuit, $c17$, is shown in Figure 2.1, with its syntactic listing in BENCH format. $c17$ is of interest to us because it represents a small-scale NAND-only logic component (i.e., small input size, small output size, small gate size) which might be found in abundance within a larger circuit or component.

2.3.2 Experimental Configuration. To create semantically equivalent circuit variants, *CORGI* uses a two-step iteration process which includes subcircuit selection followed by subcircuit replacement. We have two selection/replacement options such as random and smart [12]. If we leave the choices of the algorithm completely open to a probabilistic dice-roll made by the algorithm, we term the selection/replacement option as random. If some criteria or user preference is used to guide or replace a probabilistic choice made by the algorithm, we term the selection/replacement option as smart. *CORGI* has four options which guide subcircuit selection or replacement under these possibilities:

1. *Random selection*: Select a subcircuit $C_{sub} \subset C$ at random.
2. *Random replacement*: Select a replacement circuit $C_{rep} \in \delta_{C_{rep}}$ at random.
3. *Smart selection*: Only select subcircuits which have a particular property. If the subset of allowable selections contains more than one subcircuit, then one may be selected at random or based on another user-specified criteria.
4. *Smart replacement*: Similar to smart selection, only select replacement circuits from the library which have a particular property. If the subset of allowable selections contains more than one subcircuit, then one may be selected at random or based on another user-specified criteria.

We define an obfuscation experiment as a 5-tuple: $(C, n, \xi, \sigma, \tau)$ [12]. C is an original circuit, n is the number of iterations, ξ is a set of selection algorithms with cardinality $|\xi| = n$ where $s_i \in \xi$ implies the selection algorithm performed during iteration i, σ is a set of selection algorithms with cardinality $|\sigma| = n$ where $r_i \in \sigma$ implies the replacement algorithm performed during iteration i, and τ is a set of gates which are given selection priority during the incremental execution of the experiment.

2.3.3 Subcircuit Selection/Replacement. Given an experiment defined as the tuple $(C, n, \xi, \sigma, \tau)$, ξ represents a set of selection algorithms and $s_i \in \xi$ implies the selection algorithm used during iteration i. A subcircuit selection operation is defined as $C_{sub} = s(C, x, \gamma, \tau)$ [12] with several characteristic attributes. A circuit C is the input to the algorithm, x is the (intermediate gate) size of the selection subcircuit, $\gamma \in S$ is the particular strategy (whether smart or random), and τ is an optional set of gates that provide limiting criteria for the selection strategy itself.

The set S of possible selection strategies (described below) must include all members defined until now and *CORGI* has the following set currently: $S = \{$RandomSingleGate, RandomTwoGates, RandomLevelTwoGates, LargestLevelTwoGates, OutputLevelTwoGates, FixedLevelTwoGates, RandomAlgorithm}. The output of the algorithm C_{sub} is a circuit whose signature and $SIZE(C_{sub})$ constructs the basis for functionally equivalent alternatives and replacement. For example, iteration i that uses the `RandomLevelTwoGate` strategy would be described as $s_i = s(C, 1,$ `RandomLevelTwoGate`, $\emptyset)$, if we assume no experiment level criteria for selection/replacement.

Let Φ represent the set of all gates in a circuit C and let g_x represent a gate $g_x \in \Phi$. *CORGI* has six different subcircuit-selection strategies [12] for the experiment. The *RandomAlgorithm* strategy chooses any selection strategy in a random, uniform manner for a single iteration of the experiment and there are selection strategies as follow:

1. *RandomSingleGate*: Choose $g_1 \in \Phi$ in a random, uniform manner.
2. *RandomTwoGates*: Choose $g_1 \in \Phi$ in a random, uniform manner. Choose $g_2 \in \Phi$ where $g_2 \neq g_1$ and where the replacement of the circuit (g_1, g_2) does not create a cycle.
3. *RandomLevelTwoGates*: Choose $g_1 \in \Phi$ in a random, uniform manner. Choose $g_2 \in \Phi$ where $g_2 \neq g_1$ and where $level(g_2) = level(g_1) \pm 1$ or $level(g_2) = level(g_1)$ and where the replacement of the subcircuit $[g_1, g_2]$ does not create a cycle.
4. *LargestLevelTwoGates*: Choose $g_1 \in \Phi$ such that $|level(g_1)| = \ell_{max}$ where ℓ_{max} represents the maximum size of all levels within the circuit: $\ell_{max} = \sqcup\{|level(g_x)| \mid g_x \in \Phi\}$. Choose $g_2 \in \Phi$ where $g_2 \neq g_1$ and where $level(g_2)$ $= level(g_1) - 1$ or $level(g_2) = level(g_1)$ and where the replacement of the subcircuit $[g_1, g_2]$ does not create a cycle.
5. *OutputLevelTwoGates*: Choose $g_1 \in \Phi$ where g_1 is a distinguished intermediate gate (i.e, an output of the circuit). Choose $g_2 \in \Phi$ where $g_2 \neq g_1$ and where $level(g_2) = level(g_1) - 1$ or $level(g_2) = level(g_1)$ and where the replacement of the subcircuit $[g_1, g_2]$ does not create a cycle.

6. *FixedLevelTwoGates*: Choose $g_1 \in \Phi$ where, for some user-provided level criteria k, $level(g_1) = k$. Choose $g_2 \in \Phi$ where $g_2 \neq g_1$ and where $level(g_2) = level(g_1) - 1$ or $level(g_2) = level(g_1)$ and where the replacement of the subcircuit $[g_1, g_2]$ does not create a cycle.

7. *RandomAlgorithm*: Choose any selection strategy $\gamma \in S$ in a random, uniform manner. We may also weight one algorithm more heavily than others, and we do so programmatically.

Given an experiment defined as the tuple $(C, n, \xi, \sigma, \tau)$, σ represents a set of replacement algorithms and $r_i \in \sigma$ implies the replacement algorithm used during iteration i. A subcircuit replacement operation is defined as $C_{rep} = r(C_{sub}, z, \psi, \Omega)$ [12] with several characteristic attributes. C_{sub} is the circuit chosen for replacement, z is the requested gate size of the replacement circuit, ψ indicates criteria that determines how we generate the replacement circuit library, and Ω indicates the basis choice of the replacement circuit.

Let δ represent the circuit family, i.e. the set containing all circuits $C_{X\text{-}Y\text{-}S\text{-}\Omega}$. X is the input size, Y is the output size, S is the maximum number of gates, and Ω is the basis, where $\Omega \in \{$AND, NAND, OR, NOR, XOR, XNOR, NOT$\}$. *CORGI* first creates a set of circuits $\delta_{X\text{-}Y\text{-}S\text{-}\Omega}$. From this set of circuits, *CORGI* chooses randomly and uniformly as alternative variant for C_{sub} from the functionally equivalent subset $\delta_{C_{sub}} \subset \delta_{n\text{-}m\text{-}s\text{-}\Omega}$. *CORGI* then replaces C_{sub} with C_{rep}, where $C_{rep} \in \delta_{C_{sub}}$.

CORGI uses true/false queries that form a Boolean 6-tuple [12], which we define as ψ in the description of a replacement operation: $r(C_{sub}, z, \psi, \Omega)$. We may vary these options for every replacement opportunity in an experiment, but typically choose a set of options ψ that remain constant for the entire sequence of iterations. There are true/fase options for every replacement opportunity in the experiment as follow:

1. *RedundantGates*: Should we allow gates that are identical to other gates based on the inputs? That is to say, can we have two gates in a circuit such that the truth table for each gate, based on all *circuit* inputs, is the same?

2. *AllowConstants*: Should we allow the circuit immediate access to the constants $True$ and $False$? Gates that exhibit these properties may exist in a circuit, but it may change the properties of a set of circuits if these constants are available immediately.

3. *DoubleInputs*: Should we allow both inputs to a gate to originate in the same place? For example, if `DoubleInputs` = *true*, CORGI can produce buffer using AND($input_1, input_1$), NOT using NAND($input_1, input_1$), Constant 0 using XOR($input_1, input_1$), and Constant 1 using XNOR($input_1, input_1$).

4. *SymmetricGates*: Are gates symmetric? That is to say, should we consider a gate with inputs (X_1, X_2) as equivalent to a gate with inputs (X_2, X_1)? This will depend on which gate types are used.

5. *SimpleOutputs*: Which gates may be outputs? In an ideal circuit, any gate may be an output. However, if we want to index a circuit by output signature, outputs must be restricted to a specific set of gates. If `SimpleOutputs` = *true*, we require outputs of the circuit to all be lowest level sinks in the graph of the circuit. This option also prevents, for example, gates that are considered as outputs but never actually used–which we refer to as dangling gates.

6. *ExactCount*: Does the set contain all circuits within a certain size bound or only all circuits of an *exact* size? For example, if our circuit size is $S = 4$ and `ExactCount` = *false*, enumerating $\delta_{X\text{-}Y\text{-}S\text{-}\Omega}$ will create a family of circuits with gate size 1, 2, 3, and 4, versus a family of circuits with gate size 4 if `ExactCount` = *true*.

III. Methodology

3.1 Problem Definition

The fundamental and important goal of this research is to analyze obfuscated variants of combinational circuits to find effective methods for removing artifacts of the creation process itself. We compare this goal to how cryptanalysts use knowledge of a cryptosystem under study to find weaknesses which might compromise the security of the overall encryption scheme. In our case, we consider whether knowledge of the individual transformations that comprise an iterative selection/replacement algorithm help us remove the redundant logic pathways that are introduced by the algorithm into the variants that it produces. By examining not only the end-product of an obfuscation experiment but also the creational history that produced it, we can possibly determine if certain logic structures repeatedly arise. If those structures are conducive to minimization or reduction, then we may standardize such patterns (if they are found in the final variant) to a reduced form. The effort of finding variant logic that is conducive to standardization provides the background data by which we propose and implement a family of circuit reduction algorithms.

Once we find repeating logic patterns of interest, our next goal is to devise a method to remove those same identifiable patterns that appear in circuit variants produced by a polymorphic circuit generator (i.e, *CORGI*). We accomplish this by using the reverse logic of the identified patterns based on its specific logic attributes. As an example, one reverse pattern may change a structure from three gates into two gates as an inverse of a transformation that modifies two gates into three gates. To achieve this goal, we describe in this chapter the algorithms we introduce that use predetermined reduction patterns such as removing buffers, removing double inverters, removing constant 0/1, and so forth. We also implement algorithms that are based on predefined functions expressed as truth tables such as AND, OR, $NAND$, NOR, etc. to broaden the types of patterns we may identify. The development of such circuit reduction algorithms help to characterize the overall nature of whitebox structural changes applied to a circuit by an obfuscating engine, ultimately giving us a better approximation to how much real variation or obfuscation a polymorphic circuit generator creates.

As a last goal of this research, we desire to characterize and evaluate circuit obfuscation based on these techniques so that we may compare obfuscating algorithms one with another. An underlying assumption of our analysis is that circuits with smaller size are easier to analyze and reverse engineer than larger ones. We desire to know whether the ordering and arrangement of the reduction algorithms produce better results (in terms of reduced size) and understand the reasons why if that is the case. In terms of effectiveness, we ultimately want to determine at which point overlapping selections and replacements produce irreducible or hard to reduce variation. By moving towards these goals, we can begin to understand whether combinations of small (inefficient or rudimentary) changes to a circuit produce logic structures and pathways that cannot be removed (or understood) by reverse engineers. In this work, we would like to know specifically what the effect of highly random, small replacements on small circuits will produce towards that end.

3.2 Generating Circuit Variants

3.2.1 Defining Obfuscation Experiments. As explained in the Chapter II, *CORGI* implements experimentation by using a sequence of two-step iterations where each iteration includes a subcircuit selection followed by subcircuit replacement. In this research, we specifically consider random selection and random replacement options to generate circuit variants that we need. We also establish feasible *CORGI* obfuscation options such as subcircuit selection algorithm, library generation option and replacement gate size as described in next subsections. The number one criteria that establishes an experiment is the number of iterations. Larger iterations, as we would expect, produce circuits with larger size if we continually replace a selected subcircuit with a subcircuit that is larger in size.

3.2.2 Selection Options. As previously stated, *CORGI* has several different strategies that describe how to select a subcircuit for replacement. For our purposes, we consider only *RandomSingleGate*, *RandomTwoGates*, *RandomLevelTwoGates*, *LargestLevelTwoGates*, *OutputLevelTwoGates*, and *FixedLevelTwoGates* strategies. Of these, two are purely random (*RandomSingleGate* and *RandomTwoGates*) whereas the other four are smart in that they have predefined criteria. To maximize randomness of the algorithm while having some small degree of determinism, the *RamdomAlgorithm* strategy, for each iteration of an experiment, chooses one of the six algorithms mentioned here uniformly as the selection strategy. We can also guide a *CORGI* experiment to have certain high-level goals that are independent of the selection algorithm chosen. For example, we can define experiments with a goal that all gates within the circuit are replaced topologically over time. Such high-level goals help tailor individual selection strategies by providing a set of possible candidates for one or more of gates that are chosen. We consider such high-level goals to be deterministic, experiment-level options.

3.2.3 Replacement Options. When generating possible replacements for subcircuits during the iterations of an experiment, we choose `FFFTTT` and `TFTTTT` as Boolean values of the library generation option. As previously explained, *CORGI* uses true/false queries that form a Boolean 6-tuple as the library generation option. These options consist of *RedundantGates*, *AllowConstants*, *DoubleInputs*, *SymmetricGates*, *SimpleOutputs*, and *ExactCount.* Since the *RedundantGates* option and the *DoubleInputs* option generate more redundancies on the obfuscated circuit [12], we define two additional versions of each *CORGI* experiment, one using `FFFTTT` and one using `TFTTTT`. By using these two specific option families, we can evaluate circuits that have an expected high number of patterns that we expect would be easily reduced versus circuits that we believe would have higher irreducible logic.

As another experiment option, we can choose, for a given subcircuit selection, the size of the subcircuit that we replace it with. In our particular experiments, we choose `three` or `four` for the replacement gate size. Namely, if the *RandomAlgorithm* strategy chooses the *RandomSingleGate* algorithm, the experiment selects subcircuits containing only one gate and replaces it with subcircuits containing `three` or `four` gates. If the *RandomAlgorithm* strategy chooses any other algorithm in our subset of interest, the experiment selects subcircuits containing exactly two gates and replaces them with subcircuits containing `three` or `four` gates. Therefore, with the assumption that big replacement can be decomposed into a set of small replacements, we use only `three` or `four` for the replacement gate size.

3.2.4 Circuits. To conduct *CORGI* experiments using the above options, we must also consider the actual circuits under consideration. We use two different circuits that are merged (in parallel) from three independent circuits. Figure 3.1 illustrates the graphs of these circuits. Figure 3.1(a) shows a composite circuit with three copies of the $C17$ circuit; Figure 3.1(b) shows a composite circuit based on one $C17$ circuit and two randomly generated circuits with roughly equal number of inputs, outputs, and gate size to $C17$.

These circuits illustrate a rudimentary visual feature that is useful for considering the effects of both circuit variation and circuit reduction. When we examine the graphs of circuit variants produced by *CORGI*, the variants demonstrate one of two properties in terms of their graph: 1) either the variant circuit graph contains two or three independent (but larger) subgraphs, or 2) the variant circuit graph contains a single graph with merged nodes. As a very simple measure, variants that do not exhibit a merged graph would be considered harder to analyze or reverse engineer than variants that do exhibit independent subgraphs. If our proposed reduction sequences can take a variant with a merged single graph and reduce it to a circuit with independent subgraphs, we may also conclude that the actual obfuscation was not effective (at least in terms of hiding the topography of components).

For our purposes, we setup *CORGI* experiments to run 1000 iterations of selection/replacement and take checkpoints of the variation process at the 20, 50, 100, 500,

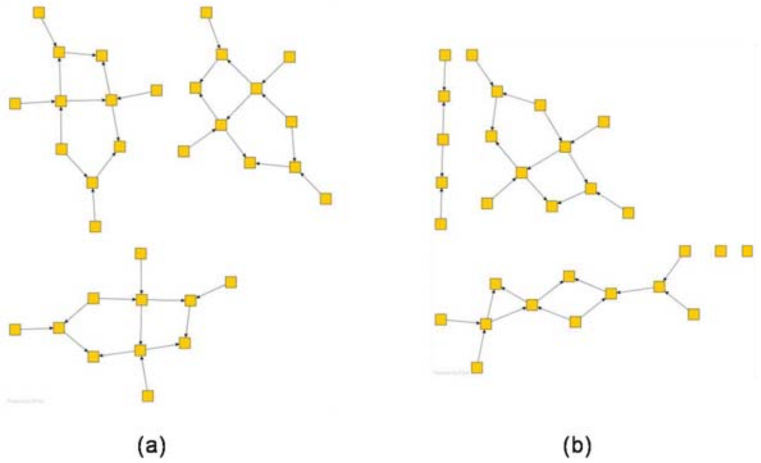


Figure 3.1: Original Circuit Graphs.
(a) circuit $C17 + C17 + C17$.
(b) circuit $R17' + C17 + R17''$.

and 1000 iteration points. Given the starting size of our original circuits at around 20 gates, 1000 iterations represents an extreme upper bound on circuit size blowup for circuits in real world applications. Figure 3.2 illustrates a variant circuit graph produced by an experiment on the $C17 + C17 + C17$ circuit seen in Figure 3.1(a). The experiment uses 500 iterations with a 3 gate replacement size and $FFFTTT$ as the library generation option. Appendix A.1 and B.1 shows the circuit variants for the original circuits $C17 + C17 + C17$ and $R17' + C17 + R17''$ at different iteration levels.

3.3 Analysis of Obfuscated Circuit Variants

3.3.1 Pattern Recognition. To discover kinds of patterns associated with iterative substitution/replacement algorithms, we manually analyze variants of combinational circuits using log files from *CORGI* experiments. For example, Figure 3.3 illustrates the log file and corresponding circuit variant graph generated by *CORGI*. It shows that two gates, $NAND(24, 26)$ and $NAND(25, 34)$, are replaced with three gates, $AND(25, 34)$, $NAND(24, 26)$ and $NAND(25, 46)$. Since $NAND(24, 26)$ does not change except gate number, we consider only the change induced by replacing

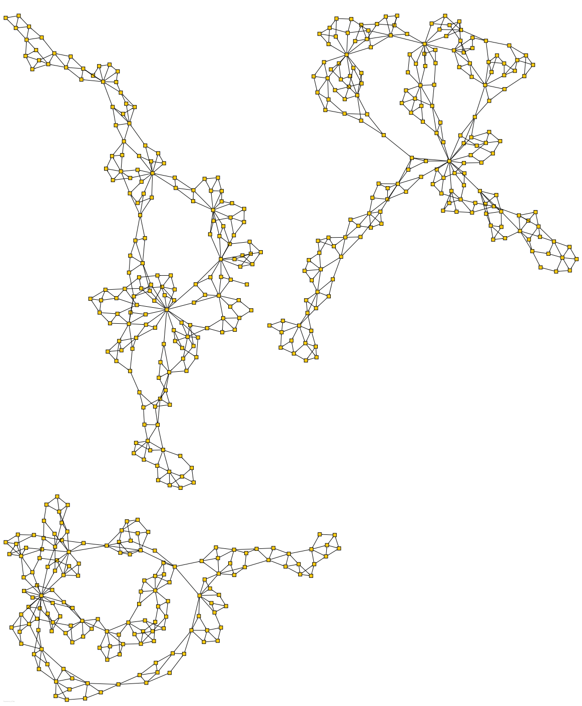
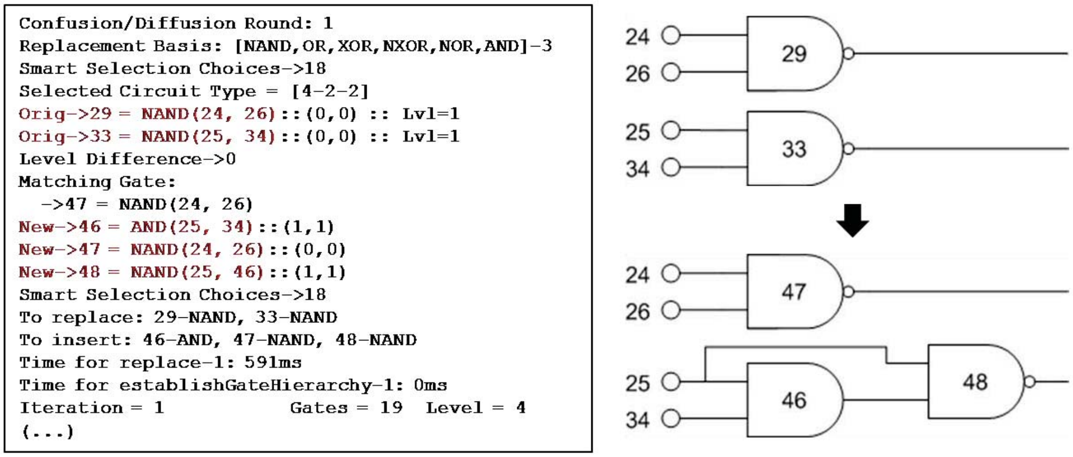


Figure 3.2: Obfuscated Circuit Variant Graph after applying 500 iterations, 3 gates replacement, and FFFTTT library generation option to circuit $C17 + C17 + C17$.

$NAND(25, 34)$ with $AND(25, 34)$ and $NAND(25, 46)$. By analyzing this transformation, we realize that subcircuits with a $NAND$ gate may be replaced with a subcircuit consisting of an AND gate followed by a $NAND$ gate. At the logic level, we may begin to reason about this using a pattern-based approach.

In current or typical *CORGI* experimentation, combinational replacement logic manifests primarily as a collection of dual (two-input) fan-in gates. We know also that all combinational circuits with multiple fan-in gates can be decomposed into dual fan-in gates as shown in Figure 3.4. Therefore, as a simplifying assumption for our research, we consider subcircuits with two input gates.

To further our analysis, we examined circuit variants generated by iteratively selecting two gates and replacing them with three gates. Our analysis of these variants revealed the presence of many redundant buffers, inverters, and constant 0/1 gates as shown in Figure 3.5. When using the *CORGI* replacement generation option $TFTTTT$, these patterns occur quite often, mainly because this option allows gates



```
Confusion/Diffusion Round: 1
Replacement Basis: [NAND,OR,XOR,NXOR,NOR,AND]-3
Smart Selection Choices->18
Selected Circuit Type = [4-2-2]
Orig->29 = NAND(24, 26)::(0,0) :: Lvl=1
Orig->33 = NAND(25, 34)::(0,0) :: Lvl=1
Level Difference->0
Matching Gate:
  ->47 = NAND(24, 26)
New->46 = AND(25, 34)::(1,1)
New->47 = NAND(24, 26)::(0,0)
New->48 = NAND(25, 46)::(1,1)
Smart Selection Choices->18
To replace: 29-NAND, 33-NAND
To insert: 46-AND, 47-NAND, 48-NAND
Time for replace-1: 591ms
Time for establishGateHierarchy-1: 0ms
Iteration = 1            Gates = 19  Level = 4
(...)
```

Figure 3.3: Log/Graph Example in CORGI

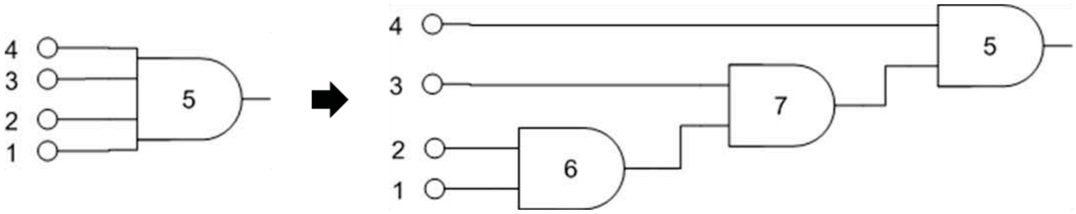


Figure 3.4: Decomposing Multiple Fan-out Gate

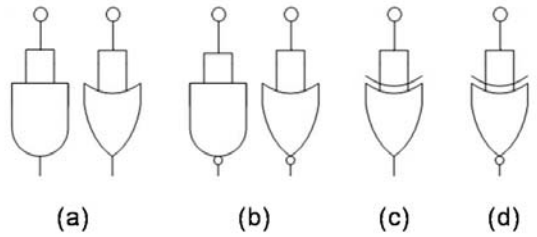
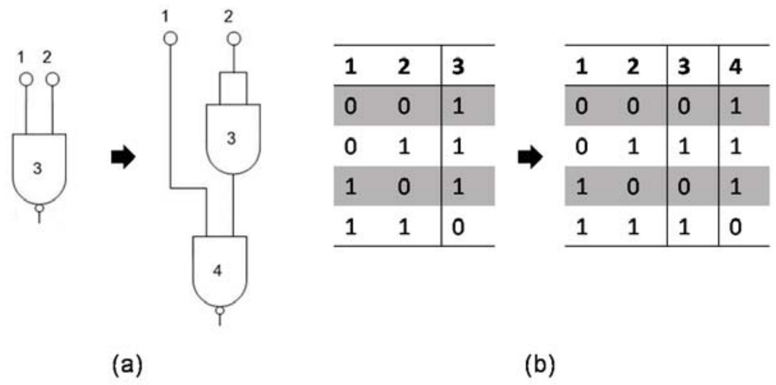


Figure 3.5: Redundant Gates
(a) Buffer
(b) Inverter
(c) Constant 0
(d) Constant 1

to have the same input value (i.e., a gate with a single input signal feeding both inputs of the gate).

3.3.2 Pattern Classification. Even though *CORGI* generates patterns that are typically considered reducible logic forms (at least when one gate and two gate selections are in view), the overall effect of iteration may exploit these rudimentary gates so that they combine with other gates in later substition/replacement operations. However, when these patterns are not reused and replaced in later experiments, they become easily identifiable by basic pattern matching. Figure 3.6, 3.7 and 3.8 illustrates the configurations of these basic patterns. The graph in Figure 3.6(a) shows that when *CORGI* replaces one gate with two gates, it is possible that a buffer can be inserted. We can relate this gate replacement operation to its truth table equivalent and show in Figure 3.6(b) their correlation. Similarly, the gate replacement using an inverter/constant would be possible using gate replacement from one gate into two gates as shown in Figure 3.7(a) and 3.8(a).

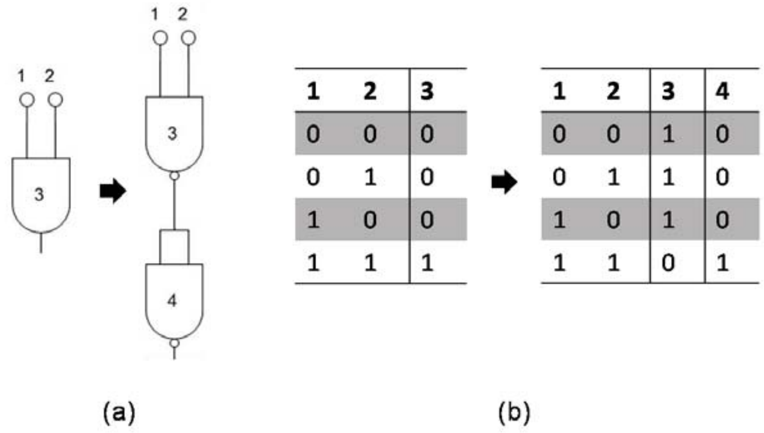
CORGI also generates more specialized gate replacement as shown in Figure 3.9. The graph in Figure 3.9(a) shows when there is one $NAND$ gate, the gate can be merged with $XNOR$ gate. Likewise, we can relate this structural gate replacement using its truth table form as shown in Figure 3.9(b). As an important facet about this type of replacement, we note that it is logically related to a larger family of gate



1	2	3
0	0	1
0	1	1
1	0	1
1	1	0

1	2	3	4
0	0	0	1
0	1	1	1
1	0	0	1
1	1	1	0

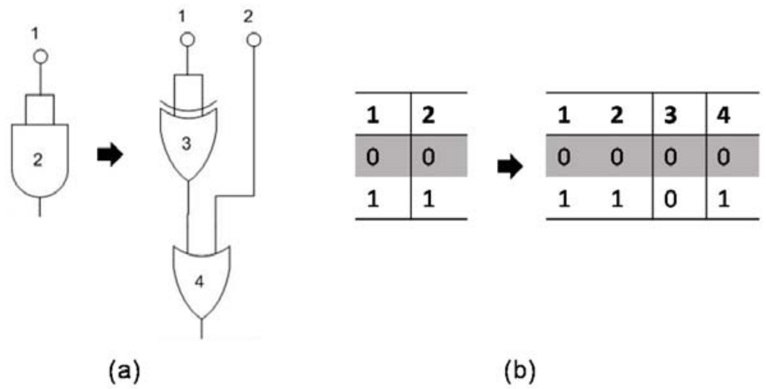
Figure 3.6: Gate Replacement using Buffer
(a) Gate Replacement
(b) Truth Table



1	2	3
0	0	0
0	1	0
1	0	0
1	1	1

1	2	3	4
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

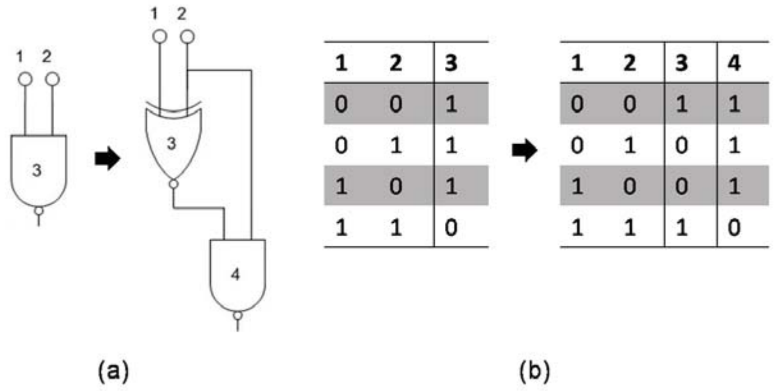
Figure 3.7: Gate Replacement using Inverter
(a) Gate Replacement
(b) Truth Table



1	2
0	0
1	1

1	2	3	4
0	0	0	0
1	1	0	1

Figure 3.8: Gate Replacement using Constant
(a) Gate Replacement
(b) Truth Table



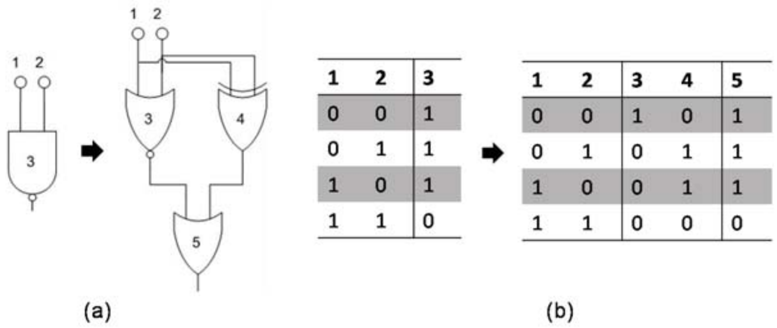
1	2	3
0	0	1
0	1	1
1	0	1
1	1	0

1	2	3	4
0	0	1	1
0	1	0	1
1	0	0	1
1	1	1	0

Figure 3.9: Gate Replacement Example for the Special Case
(a) Gate Replacement
(b) Truth Table

replacements. For example, if the original gate in Figure 3.9(a) is an AND gate, a functional equivalent to this would be the $XNOR$ and AND gate pair seen in the right side of the figure. The basis of our circuit reduction algorithm is to essentially find such patterns and use their smaller equivalent structural form, while retaining the overall circuit logic.

In contrast to two gate selection/three gate replacement, gate replacements from two gates to four gates generated a wider variety of structural patterns, as we would expect. Figure 3.10 illustrates an example of this replacement type. In this case,



1	2	3
0	0	1
0	1	1
1	0	1
1	1	0

1	2	3	4	5
0	0	1	0	1
0	1	0	1	1
1	0	0	1	1
1	1	0	0	0

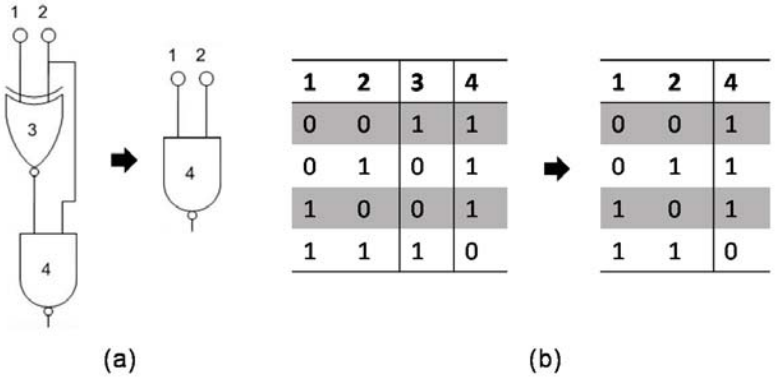
Figure 3.10: Complex Gate Replacement Example
(a) Gate Replacement
(b) Truth Table

we need to utilize truth table analysis in order to understand the logical effect of the replacement. We consider these complex gate replacements specifically when we implement circuit reduction techniques, which we describe in the next section.

3.4 Implementation of Circuit Reduction Algorithms

As the previous section describes, our methodology for creating reduction algorithms begins by analyzing the circuit variants produced by *CORGI*. If we consider the reverse pattern of the underlying logic that we observe, we have a basis to understand how to remove or reduce such logic that still exists in the final circuit variant. For example, if there exists a change from a single $NAND(1,2)$ to $XNOR(1,2)$ and $NAND(3,2)$ as shown in Figure 3.9, we can consider the reverse structural change of this from $XNOR(1,2)$ and $NAND(3,2)$ to $NAND(1,2)$ as shown in Figure 3.11.

Building upon our investigation of sample circuit variants, we build a wide variety of predetermined reduction patterns that appear most frequently, giving us a possibility to easily remove induced redundancy introduce by the variation process. Ultimately, our goal would be to remove all new gates introduced by an obfuscating algorithm. Realistically, we would like to remove as many as possible using just basic pattern matching. Using this approach, we define and implement 10 patterns for circuit reduction algorithms and describe them in sections 3.4.1 - 3.4.10.



1	2	3	4
0	0	1	1
0	1	0	1
1	0	0	1
1	1	1	0

1	2	4
0	0	1
0	1	1
1	0	1
1	1	0

Figure 3.11: Reverse Logic Example of Figure 3.9
(a) Gate Replacement
(b) Truth Table

When complex gate replacements are in view, like those seen in Figure 3.10, we use truth table analysis to increase our pattern search space above those provided by our 10 basic circuit reduction techniques. Using this approach, we define and implement two truth-table based reduction algorithms and describe them in sections 3.4.11 - 3.4.12.

3.4.1 Reduce Buffer. This algorithm implements the following reduction logic:

- For each gate, if it is an AND/OR gate that has both inputs from the same source, remove it as shown in Figure 3.12.
- For each gate, if it is a buffer, remove it as shown in Figure 3.12.

3.4.2 Reduce Inverter. This algorithm implements the following reduction logic:

- For each gate, if it is an inverter($NAND/NOR/NOT$) that has both inputs from the same source and a predecessor of the gate is a NOT gate, remove them as shown in Figure 3.13(a).

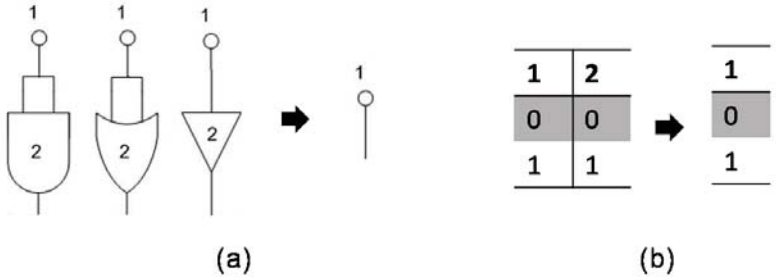
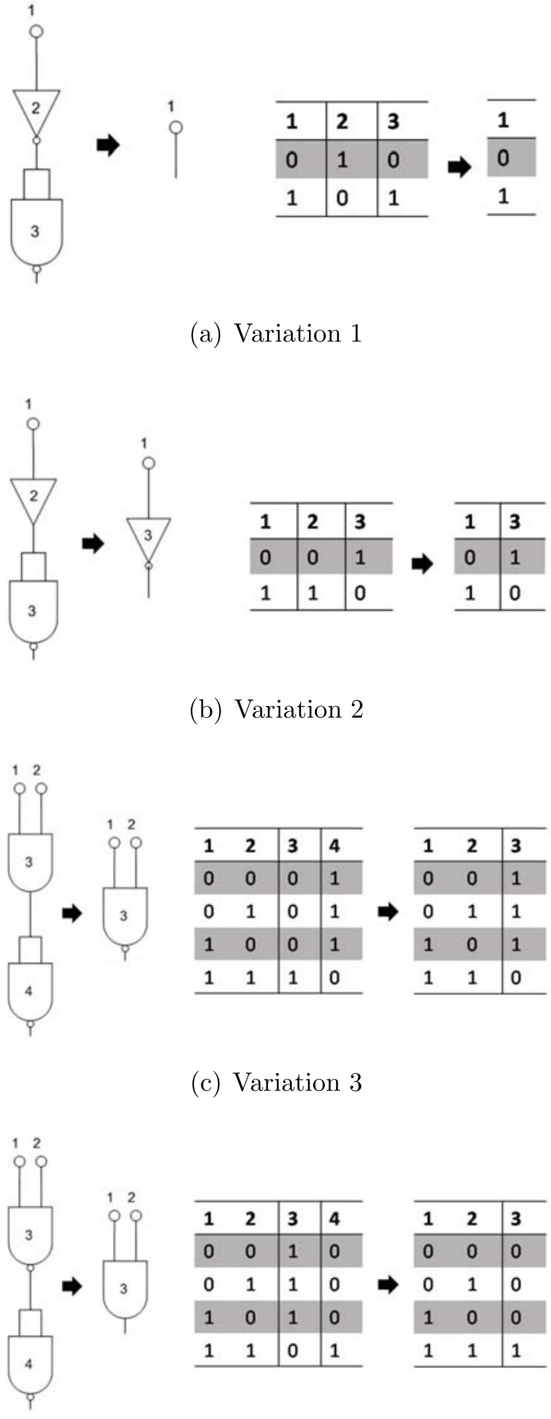


Figure 3.12: Reduction Algorithm 1(Reduce Buffer)
(a) Gate Replacement
(b) Truth Table

- For each gate, if it is an inverter($NAND/NOR/NOT$) that has both inputs from the same source and a predecessor of the gate is a buffer, remove it and change the buffer into a NOT gate as shown in Figure 3.13(b).
- For each gate, if it is an inverter($NAND/NOR/NOT$) that has both inputs from the same source and a predecessor of the gate is a positive gate($AND/OR/XOR$), remove it and change the positive gate($AND/OR/XOR$) into a negative gate($NAND/NOR/XNOR$) as shown in Figure 3.13(c).
- For each gate, if it is an inverter($NAND/NOR/NOT$) that has both inputs from the same source and a predecessor of the gate is a negative gate($NAND/NOR/XNOR$), remove it and change the negative gate($NAND/NOR/XNOR$) into a positive gate($AND/OR/XOR$) as shown in Figure 3.13(d).

3.4.3 Reduce Inverter with Successor $XOR/XNOR$. This algorithm implements the following reduction logic:

- For each gate, if it is an inverter($NAND/NOR/NOT$) that has both inputs from the same source and a successor of the gate is an XOR gate, remove it and change the XOR gate into an $XNOR$ gate as shown in Figure 3.14(a).
- For each gate, if it is an inverter($NAND/NOR/NOT$) that has both inputs from the same source and a successor of the gate is an $XNOR$ gate, remove it and change the $XNOR$ gate into an XOR gate as shown in Figure 3.14(b).



1	2	3
0	1	0
1	0	1

1
0
1

(a) Variation 1

1	2	3
0	0	1
1	1	0

1	3
0	1
1	0

(b) Variation 2

1	2	3	4
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

1	2	3
0	0	1
0	1	1
1	0	1
1	1	0

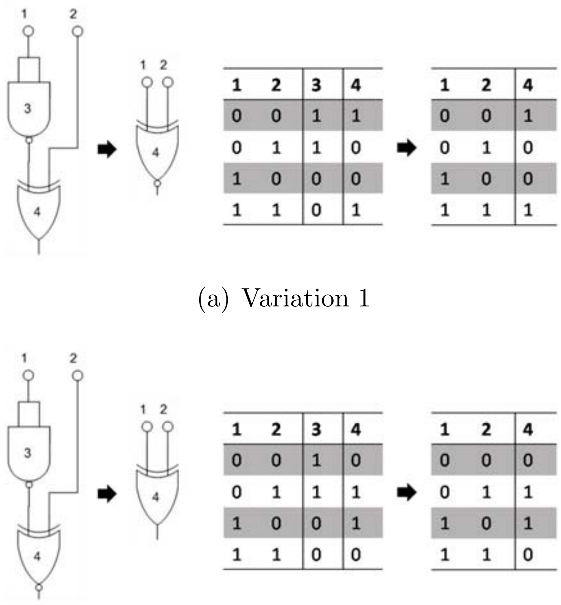
(c) Variation 3

1	2	3	4
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

1	2	3
0	0	0
0	1	0
1	0	0
1	1	1

(d) Variation 4

Figure 3.13: Gate Replacement and Truth Table of Reduction Algorithm 2 (Reduce Inverter)



1	2	3	4
0	0	1	1
0	1	1	0
1	0	0	0
1	1	0	1

1	2	4
0	0	1
0	1	0
1	0	0
1	1	1

(a) Variation 1

1	2	3	4
0	0	1	0
0	1	1	1
1	0	0	1
1	1	0	0

1	2	4
0	0	0
0	1	1
1	0	1
1	1	0

(b) Variation 2

Figure 3.14: Gate Replacement and Truth Table of Reduction Algorithm 3 (Reduce Inverter with Successor $XOR/XNOR$)

3.4.4 Reduce Constant 0/1. This algorithm implements the following reduction logic:

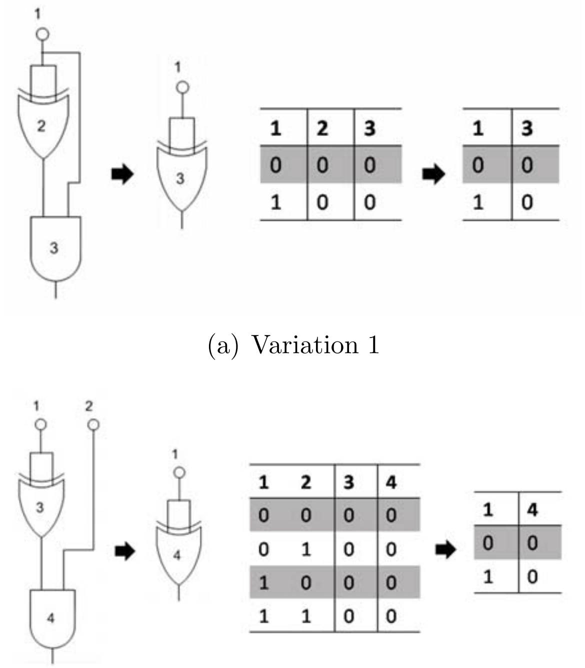
- For each gate, if it is a constant 0 that its next gate is an AND gate, remove it and change the AND gate into a constant 0 as shown in Figure 3.15(a) and 3.15(b).
- For each gate, if it is a constant 0 that its next gate is a $NAND$ gate, remove it and change the $NAND$ gate into a constant 1. The gate replacement and truth table of this logic resemble Figure 3.15(a) and 3.15(b) except change of gate types.
- For each gate, if it is a constant 0 that its next gate is an OR/XOR gate, remove it and change the OR/XOR gate into a buffer. The gate replacement and truth table of this logic resemble Figure 3.15(a) and 3.15(b) except change of gate types.

- For each gate, if it is a constant 0 that its next gate is a $NOR/XNOR$ gate, remove it and change the $NOR/XNOR$ gate into a NOT gate. The gate replacement and truth table of this logic resemble Figure 3.15(a) and 3.15(b) except change of gate types.
- For each gate, if it is a constant 1 that its next gate is a NOR gate, remove it and change the NOR gate into a constant 0. The gate replacement and truth table of this logic resemble Figure 3.15(a) and 3.15(b) except change of gate types.
- For each gate, if it is a constant 1 that its next gate is an OR gate, remove it and change the OR gate into a constant 1. The gate replacement and truth table of this logic resemble Figure 3.15(a) and 3.15(b) except change of gate types.
- For each gate, if it is a constant 1 that its next gate is an $AND/XNOR$ gate, remove it and change the $AND/XNOR$ gate into a buffer. The gate replacement and truth table of this logic resemble Figure 3.15(a) and 3.15(b) except change of gate types.
- For each gate, if it is a constant 1 that its next gate is a $NAND/XOR$ gate, remove it and change the $NAND/XOR$ gate into a NOT gate. The gate replacement and truth table of this logic resemble Figure 3.15(a) and 3.15(b) except change of gate types.

3.4.5 Reduce Constant 0/1 with Inverter Inputs. This algorithm implements the following reduction logic:

- For each gate, if it is a constant 0/1 that has only an inverter as input, remove the inverter as shown in Figure 3.16(a) and 3.16(b).

3.4.6 Reduce Two Gates to $AND/NAND/OR/NOR$. This algorithm implements the following reduction logic:



1	2	3
0	0	0
1	0	0

1	3
0	0
1	0

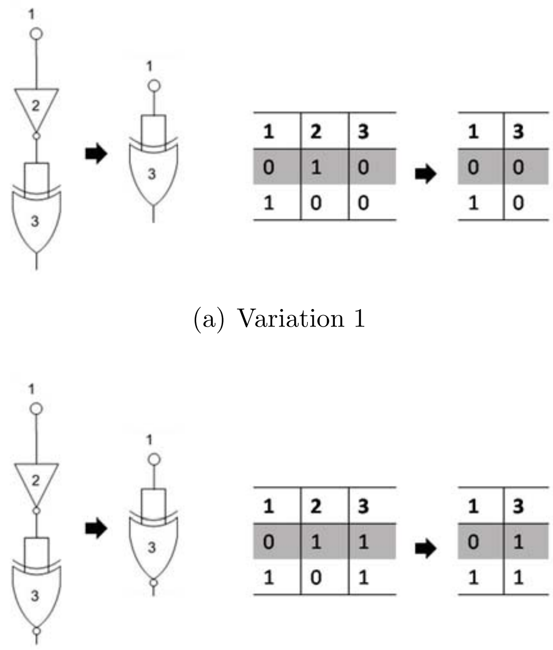
(a) Variation 1

1	2	3	4
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	0

1	4
0	0
1	0

(b) Variation 2

Figure 3.15: Gate Replacement and Truth Table of Reduction Algorithm 4 (Reduce Constant 0/1)



1	2	3
0	1	0
1	0	0

1	3
0	0
1	0

(a) Variation 1

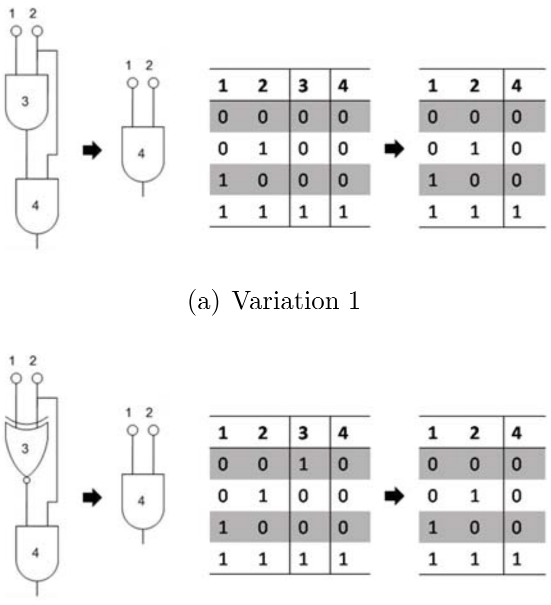
1	2	3
0	1	1
1	0	1

1	3
0	1
1	1

(b) Variation 2

Figure 3.16: Gate Replacement and Truth Table of Reduction Algorithm 5 (Reduce Constant 0/1 with Inverter Inputs)

- For each gate, if it is an $AND/XNOR$ gate and its next gate is an AND gate, remove it as shown in Figure 3.17(a) and 3.17(b).
- For each gate, if it is an $AND/XNOR$ gate and its next gate is a $NAND$ gate, remove it. The gate replacement and truth table of this logic resemble Figure 3.17(a) and 3.17(b) except change of gate types.
- For each gate, if it is an OR/XOR gate and its next gate is an OR gate, remove it. The gate replacement and truth table of this logic resemble Figure 3.17(a) and 3.17(b) except change of gate types.
- For each gate, if it is an OR/XOR gate and its next gate is a NOR gate, remove it. The gate replacement and truth table of this logic resemble Figure 3.17(a) and 3.17(b) except change of gate types.



1	2	3	4
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

1	2	4
0	0	0
0	1	0
1	0	0
1	1	1

(a) Variation 1

1	2	3	4
0	0	1	0
0	1	0	0
1	0	0	0
1	1	1	1

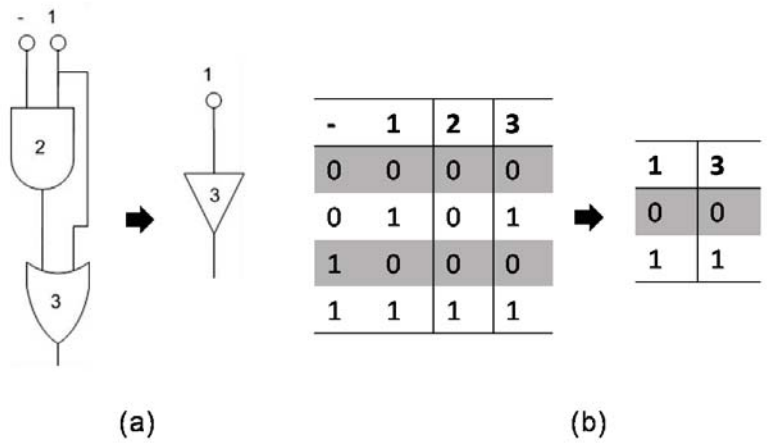
1	2	4
0	0	0
0	1	0
1	0	0
1	1	1

(b) Variation 2

Figure 3.17: Gate Replacement and Truth Table of Reduction Algorithm 6 (Reduce Two Gates to $AND/NAND/OR/NOR$)

3.4.7 Reduce Two Gates to Buffer/NOT/Constant 0/1. This algorithm implements the following reduction logic:

- For each gate, if it is an AND gate and its next gate is an OR gate, remove it and change the OR gate into a buffer as shown in Figure 3.18.
- For each gate, if it is an OR gate and its next gate is an AND gate, remove it and change the AND gate into a buffer. The gate replacement and truth table of this logic resemble Figure 3.18 except change of gate types.
- For each gate, if it is an AND gate and its next gate is a NOR gate, remove it and change the NOR gate into a NOT gate. The gate replacement and truth table of this logic resemble Figure 3.18 except change of gate types.
- For each gate, if it is an OR gate and its next gate is a $NAND$ gate, remove it and change the $NAND$ gate into a NOT gate. The gate replacement and truth table of this logic resemble Figure 3.18 except change of gate types.



-	1	2	3
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	1

1	3
0	0
1	1

Figure 3.18: Reduction Algorithm 7
(Reduce Two Gates to Buffer/NOT/Constant 0/1)
(a) Gate Replacement
(b) Truth Table

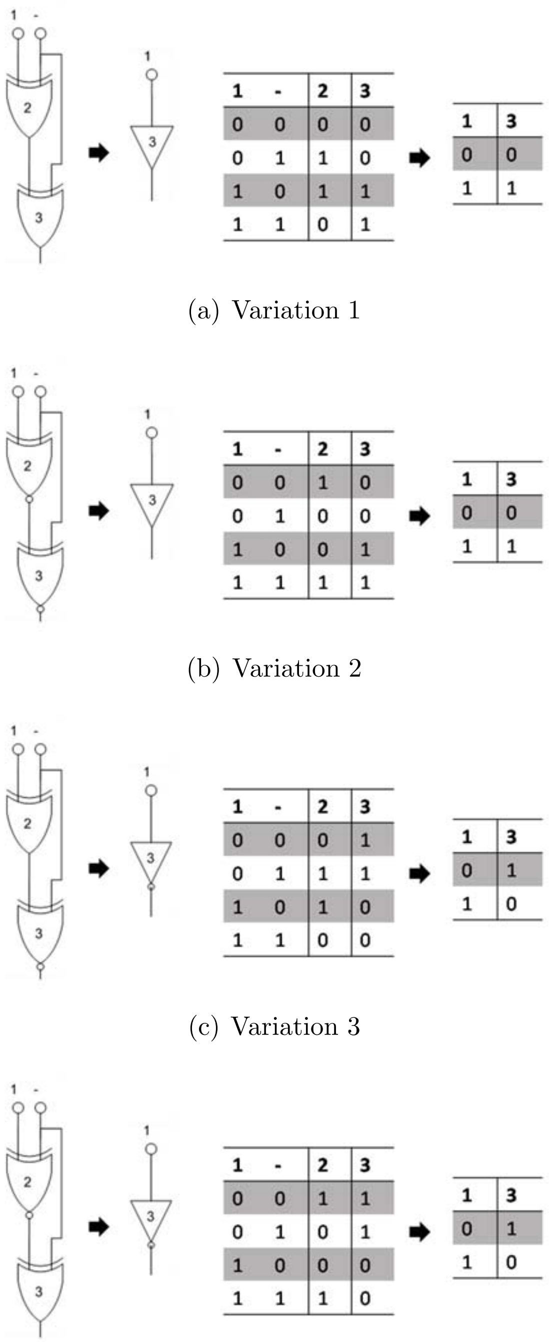
- For each gate, if it is a NOR gate and its next gate is an AND gate, remove it and change the AND gate into a constant 0. The gate replacement and truth table of this logic resemble Figure 3.18 except change of gate types.

- For each gate, if it is a NOR gate and its next gate is a $NAND$ gate, remove it and change the $NAND$ gate into a constant 1. The gate replacement and truth table of this logic resemble Figure 3.18 except change of gate types.

3.4.8 Reduce Two XOR/XNOR Gates to Buffer/NOT. This algorithm implements the following reduction logic:

- For each gate, if it is an $XOR/XNOR$ gate and its next gate is an $XOR/XNOR$ gate, remove it and change the $XOR/XNOR$ gate into a buffer as shown in Figure 3.19(a) and 3.19(b).

- For each gate, if it is an $XOR/XNOR$ gate and its next gate is an $XNOR/XOR$ gate, remove it and change the $XNOR/XOR$ gate into a NOT gate as shown in Figure 3.19(c) and 3.19(d).



1	-	2	3
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

1	3
0	0
1	1

(a) Variation 1

1	-	2	3
0	0	1	0
0	1	0	0
1	0	0	1
1	1	1	1

1	3
0	0
1	1

(b) Variation 2

1	-	2	3
0	0	0	1
0	1	1	1
1	0	1	0
1	1	0	0

1	3
0	1
1	0

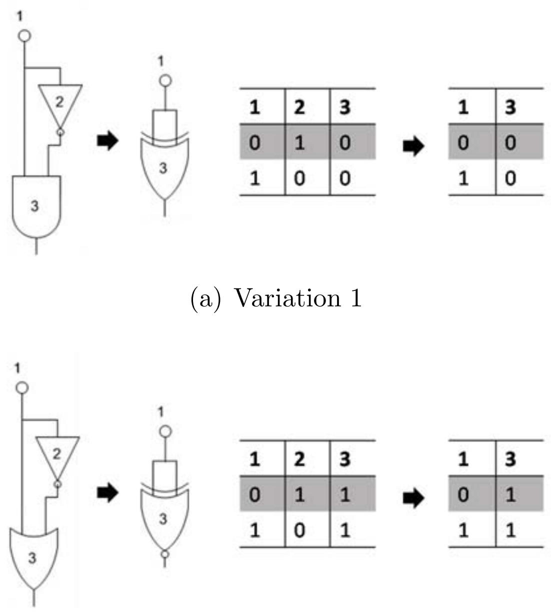
(c) Variation 3

1	-	2	3
0	0	1	1
0	1	0	1
1	0	0	0
1	1	1	0

1	3
0	1
1	0

(d) Variation 4

Figure 3.19: Gate Replacement and Truth Table of Reduction Algorithm 8 (Reduce Two $XOR/XNOR$ Gates to Buffer/NOT)



1	2	3
0	1	0
1	0	0

1	3
0	0
1	0

(a) Variation 1

1	2	3
0	1	1
1	0	1

1	3
0	1
1	1

(b) Variation 2

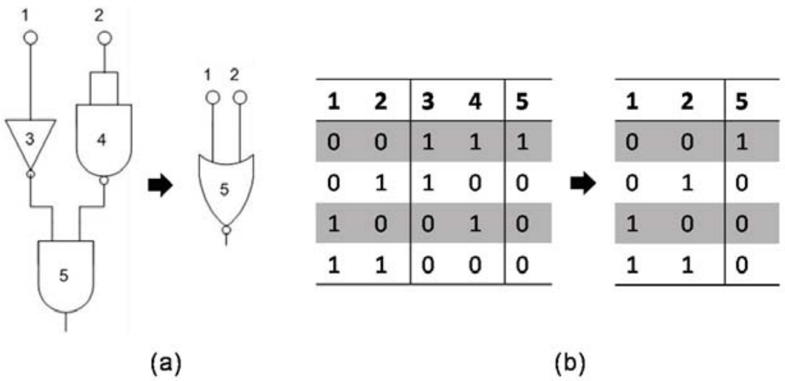
Figure 3.20: Gate Replacement and Truth Table of Reduction Algorithm 9 (Reduce Gate with Opposite Inputs)

3.4.9 Reduce Gate with Opposite Inputs. This algorithm implements the following reduction logic:

- For each gate, if it is an $AND/NOR/XNOR$ gate that has only opposite inputs, change them into a constant 0 as shown in Figure 3.20(a).
- For each gate, if it is an $OR/NAND/XOR$ gate that has only opposite inputs, change them into a constant 1 as shown in Figure 3.20(b).

3.4.10 Reduce $AND/OR/NAND/NOR$ Gates with Inverter Inputs. This algorithm implements the following reduction logic:

- For each gate, if it is an AND gate that has only inverters as inputs, change it into a NOR gate and remove the inverters as shown in Figure 3.21.
- For each gate, if it is an OR gate that has only inverters as inputs, change it into a $NAND$ gate and remove the inverters. The gate replacement and truth table of this logic resemble Figure 3.21 except change of gate types.



1	2	3	4	5
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

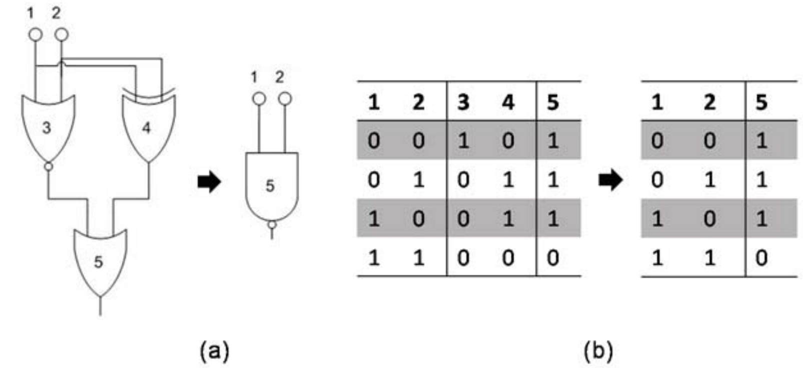
1	2	5
0	0	1
0	1	0
1	0	0
1	1	0

Figure 3.21: Reduction Algorithm 10
(Reduce $AND/OR/NOT$ Gates with Inverter Inputs)
(a) Gate Replacement
(b) Truth Table

- For each gate, if it is a $NAND$ gate that has only inverters as inputs, change it into an OR gate and remove the inverters. The gate replacement and truth table of this logic resemble Figure 3.21 except change of gate types.
- For each gate, if it is a NOR gate that has only inverters as inputs, change it into an AND gate and remove the inverters. The gate replacement and truth table of this logic resemble Figure 3.21 except change of gate types.

3.4.11 Reduce V pattern. For each V pattern, if it has a truth table equal to predefined functions such as AND, OR, $NAND$, NOR, XOR, $XNOR$, buffer, NOT, constant 0/1, and combinations of these, change the pattern into the equivalent reduced form as shown in Figure 3.22.

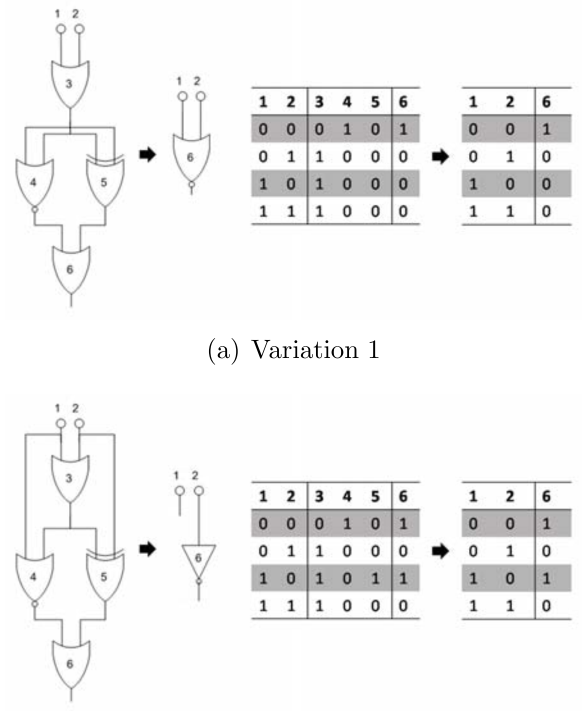
3.4.12 Reduce Diamond pattern. For each Diamond pattern, if it has a truth table equal to predefined function such as AND, OR, $NAND$, NOR, XOR, $XNOR$, buffer, NOT, constant 0/1, and combinations of these, change the pattern into the equivalent reduced form as shown in Figure 3.23(a) and 3.23(b).



1	2	3	4	5
0	0	1	0	1
0	1	0	1	1
1	0	0	1	1
1	1	0	0	0

1	2	5
0	0	1
0	1	1
1	0	1
1	1	0

Figure 3.22: Reduction Algorithm 11(Reduce V pattern)
(a) Gate Replacement
(b) Truth Table



1	2	3	4	5	6
0	0	0	1	0	1
0	1	1	0	0	0
1	0	1	0	0	0
1	1	1	0	0	0

1	2	6
0	0	1
0	1	0
1	0	0
1	1	0

(a) Variation 1

1	2	3	4	5	6
0	0	0	1	0	1
0	1	1	0	0	0
1	0	1	0	1	1
1	1	1	0	0	0

1	2	6
0	0	1
0	1	0
1	0	1
1	1	0

(b) Variation 2

Figure 3.23: Gate Replacement and Truth Table of Reduction Algorithm 12 (Reduce Diamond pattern)

Table 3.1: Experimental Setup for Circuit Reduction

Obfuscated Options				**Reduction Options**		
Selection Algorithm	**Gate Replaced**	**Library Generation**	**Obfuscation Iteration**	**Circuit Type**	**Reduction Algorithm**	**Reduction Round**
Random Algorithm	3 4	$FFFTTT$ $TFTTTT$	20 50 100 500 1000	$C17-C17-C17$ $R17'-C17-R17''$	*Random Algorithm*	10

3.5 Evaluation of Circuit Obfuscation

3.5.1 Experimental Setup. As described in Section 3.2, we generate functionally equivalent variants of two different circuits ($C17-C17-C17$, $R17'-C17-R17''$) using a specific set of experiment options. We summarize these experimental configuration options in Table 3.1. Using these options, we produce a total of 20 different variants for each circuit, giving us a set of 40 circuits with which to consider the effect of reduction. For each particular circuit in our variant set, we normalize them by applying our set of 12 reduction techniques (described in Sections 3.4.1 - 3.4.12) in a sequential manner. We define a reduction round as the application of these 12 individual reduction algorithms one after the other, where the reduced circuit form produced by one reduction technique becomes the input to the next reduction technique. We note that there exists at least 12! possibilities for ordering the individual techniques themselves within a reduction round. We therefore define a reduction experiment as a starting circuit variant and the application of an ordered sequence of our 12 reduction algorithms, in some iterative fashion. As the final experiment parameter, we may run multiple reduction rounds (i.e., apply the same 12 individual techniques again) on the same variant in an iterative manner. As Table 3.1 illustrates, we use 10 reduction rounds and a random ordering of the individual reduction algorithms for our experimental purposes. Once we choose a random ordering, that ordering remains the same for the entire 10 rounds of a reduction experiment. Because full enumeration of all possible algorithm orderings is not computationally feasible ($O(n!)$), we chose a random approach as the best starting point to characterize how much circuit reduction is affected by algorithm ordering. We also look to see if there is a lower bound on the number of rounds where we no longer observe size reduction on the variant circuit. We use 10 as a rough upper bound on number of rounds to run for each reduction experiment based on our observations while running over 500 different reduction experiments. Based on a fixed reduction round size of 10, we can apply our set of circuit reduction algorithms at most 120 times, where no algorithm is applied more than 12 times itself.

Given the set of 40 circuit variants under consider, we describe the basic algorithm for our experimental approach in the following pseudocode. Let each reduction technique be represented by a function R_k which takes as input a circuit C and produces a functionally equivalent version C'; k is in the range 1..12, representing one of the algorithms described in Sections 3.4.1 - 3.4.12. The specific algorithm R_k takes a circuit and iterates through the entire gate list of the circuit. As this process occurs, if the particular pattern indicated by the reduction technique is encountered, the appropriate reduction is performed. The algorithm accomplishes this by traversing through a single pass of the circuit gate list.

We define a reduction round RR as a 12-tuple consisting of an ordered sequence of reduction techniques. For example, $RR = R_2$, R_4, R_5, R_{10}, R_1, R_{11}, R_{12}, R_7, R_3, R_8, R_6, R_9 represents a reduction round where the techniques are applied in the following sequential order: 2, 4, 5, 10, 1, 11, and so forth. The index order of the reduction algorithms represents a single permutation of the total ordering possibilities. We let

```
procedure reductionExperiment(circuit $C$)
$C_0 \leftarrow C$
Generate reduction round $RR$ using random permutation // 12 reduction algorithms
for $rounds$ = 1 to MAXROUNDS do // MAXROUNDS = 10 reduction rounds
    for $x = 1$ to 12 do
        $C_x \leftarrow RR_x(C_{x-1})$
    end for
end for
$C' \leftarrow C_{12}$
Generate log file
```

RR_x represent the xth element of the 12-tuple RR. In our example, RR_1 represents reduction algorithm 2 (R_2), which is Reduce Inverter. In other words, for that particular reduction round, Reduce Inverter will be the first reduction technique applied to a circuit. Given an original circuit variant C, a reduction experiments produces 12 reduced circuit forms in the following manner: for each round x, $C_x = RR_x(C_{x-1})$. The reduction experiment outputs a final reduced circuit form C', corresponding to the output of the last circuit reduction technique applied.

We analyze log files generated by reduction experiments and make observations based on different measures. The log file includes gate size, circuit level, CPU time, execution number of each algorithm per a reduction round and so on. We use this information to characterize and evaluate the strength of circuit obfuscation. We describe results of experimentation and our subsequent analysis further in Chapter IV.

3.5.2 Experimental Speculations. An important research question of interest in this thesis concerns how to characterize and evaluate the strength of circuit obfuscation using metrics such as gate size. We assume, in general, that a circuit with reduced size is easier to reverse engineer or perform adversarial analysis on than one that is larger. An underlying assumption is that if we reduce a circuit variant close to its original size, we will actually recover a structure that is largely similar to the original. This gives us a measurable framework to know to what extent true obfuscation (or complexity in analysis) has actually taken place. We offer the following speculations as assumptions which will guide analysis of the data produced by executing our experimental methodology.

- If the obfuscating algorithm uses four gates as a replacement size, we suppose that reduction experiments would reduce those variants *less* when compared to variants produced by obfuscating algorithms that use three gate replacement options (assuming all other experimental options are the same). Our insight here is that four gate replacements have a wider variety of structures to choose from and thus provide possibility for more complex entanglement among gates in the obfuscated circuit variant which would not be covered by our 12 basic patterns. Also, four gate replacement, when two gates are chosen for each subcircuit, doubles the size of the obfuscated circuit variant as well.

- If the obfuscating algorithm uses $TFTTTT$ as a library generation option for replacement circuits, we suppose that reduction experiments would reduce those variants *more* when compared to variants produced by obfuscating algorithms that use $FFFTTT$ as a library generation option (assuming all other experimental options are the same). Our insight here centers on the fact that $TFTTTT$ replacement subcircuits have a higher number of basic (easily reducible) patterns based on duplicated signals and dual input gates (which are logical buffers and 0/1 constants). Therefore, we would assume that basic pattern matching would produce higher reduction rates (as a percentage of total circuit size) based on empirical observation.

- If the obfuscating algorithm uses a higher number of selection/replacement iterations (or rounds), we suppose that reduction experiments would reduce those variants *less* when compared to variants produced by obfuscating algorithms that use lower number of rounds (assuming all other experimental options are the same). This speculation is based on the general assumption that some complexity is introduced over an iterative transformation sequence, and that longer iterations give more possibility for producing patterns that are not covered by our basic forms.

- We speculate that the specific order of circuit reduction algorithms (the permutation that represents the sequence of algorithms in a reduction round tuple RR) will affect reduction. Our insight here comes from an observation that some reduction techniques will produce structures that can then be reduced further by other techniques.

Chapter IV presents detailed analysis of the methodology we present in this chapter. We detail the data results of the experiments we performed, detail summary analysis and data trends, and analyze our speculations regarding reduction trends.

IV. Results

4.1 Overview

This chapter discusses the results of executing the research methodology described in Chapter III. We analyze a set of 40 circuit variants, perform reduction experiments on them, and then analyze their minimized form using gate size as our primary metric to characterize and evaluate circuit obfuscation. Additionally, we explore circuit reduction effects based on order of circuit reduction algorithms.

4.2 Circuit Reduction

As previously stated, in order to apply our circuit reduction algorithms, we used obfuscated circuit variants using three/four gate replacement, $FFFTTT/TFTTTT$ library generation options, and iteration numbers in the range 20/50/100/500/1000. This CORGI experiment setup classified the experiment results by four categories, $3/FFFTTT$, $3/TFTTTT$, $4/FFFTTT$ and $4/TFTTTT$.

In order to evaluate each circuit obfuscation option, we calculate reduction percentages after 20/50/100/500/1000 iterations using our circuit reduction algorithms with obfuscated circuit $C17-C17-C17$ and $R17'-C17-R17''$. We calculate the reduction percentage as (removed gate counts)÷(original gate counts)×100 as shown in Table 4.1. Figure 4.1 provides a plot of these data.

As shown in Table 4.1 and Figure 4.1, the circuit obfuscation using four gate replacement and FFFTTT library generation option shows a reduction result that has removed gates less than other circuit obfuscations. Therefore, we can know that the FFFTTT library generation option makes less redundant circuit because the option does not allow redundant gates that are identical to other gates based on the inputs and double inputs to a gate to originate in the same place. We also know that four gate replacement makes less redundant circuit because four gate replacement causes more complex gate replacement than three gate replacement.

Also shown in Table 4.1 and Figure 4.1, the obfuscated circuit $C17-C17-C17$ after 1000 iterations with 3 gate replacement and TFTTTT library generation option

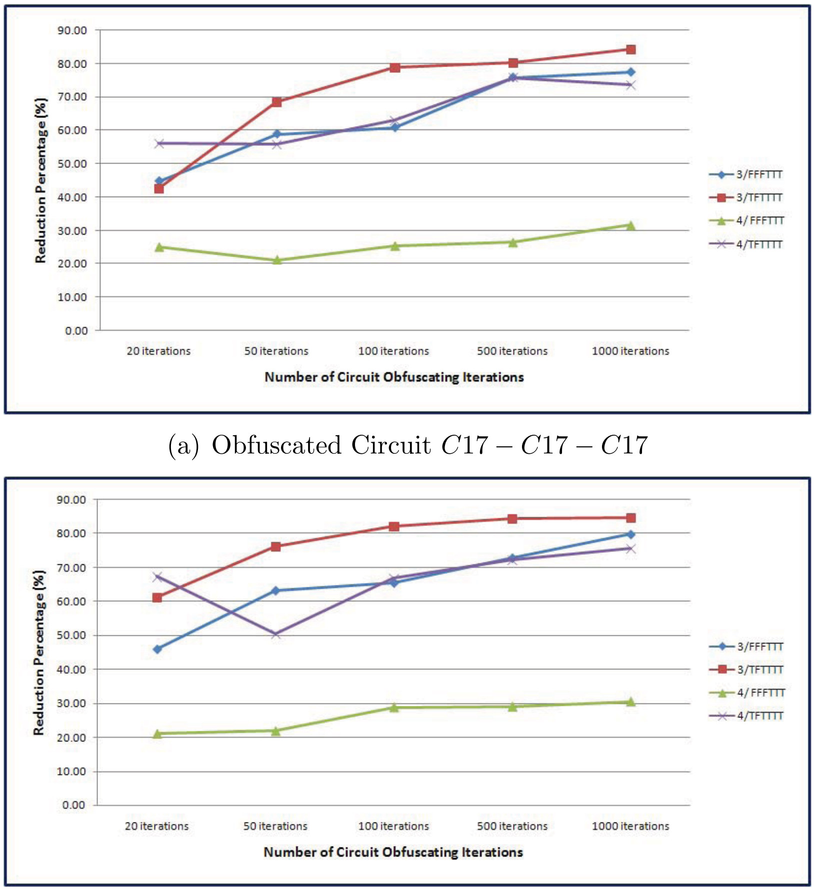
Table 4.1: Average Reduction Percentage using Obfuscated Circuit Variants

(a) Obfuscated Circuit $C17 - C17 - C17$

	3/FFFTTT	**3/TFTTTT**	**4/FFFTTT**	**4/TFTTTT**
20 Iterations	44.74	42.50	25.00	55.93
50 Iterations	58.90	68.49	21.14	55.74
100 Iterations	60.80	78.74	25.33	62.88
500 Iterations	75.80	80.18	26.35	75.62
1000 Iterations	77.41	84.22	31.56	73.46

(b) Obfuscated Circuit $R17' - C17 - R17''$

	3/FFFTTT	**3/TFTTTT**	**4/FFFTTT**	**4/TFTTTT**
20 Iterations	45.95	61.11	21.05	67.24
50 Iterations	63.24	76.12	21.85	50.41
100 Iterations	65.49	82.11	28.83	66.81
500 Iterations	72.81	84.37	29.00	72.16
1000 Iterations	79.77	84.59	30.47	75.55



(a) Obfuscated Circuit $C17 - C17 - C17$

(b) Obfuscated Circuit $R17' - C17 - R17''$

Figure 4.1: Average Reduction Percentage using Obfuscated Circuit Variants

shows that the best reduction result, 84.22% reduction percentage, and the obfuscated circuit $R17' - C17 - R17''$ after 20 iterations with 4 gate replacement and FFFTTT library generation option shows that the worst reduction result, 21.05% reduction percentage. These results are analyzed in next two subsections.

4.2.1 The Best Reduction. As previously stated, the obfuscated circuit $C17-C17-C17$ after 1000 iterations with 3 gate replacement and TFTTTT library generation option generates the best reduced circuit. The metrics such as gate counts and level counts are as shown in Table 4.2. Figure 4.2 provides a plot of these data. As shown in Table 4.2 and Figure 4.2, reduction percentage of gate counts is in between 83.12% and 85.58% and reduction percentage of level count is in between 84.53% and 86.79%. This statistics shows that this option brings about less powerful circuit obfuscation because our reduction algorithms reduce more gates and levels than other circuit obfuscation options.

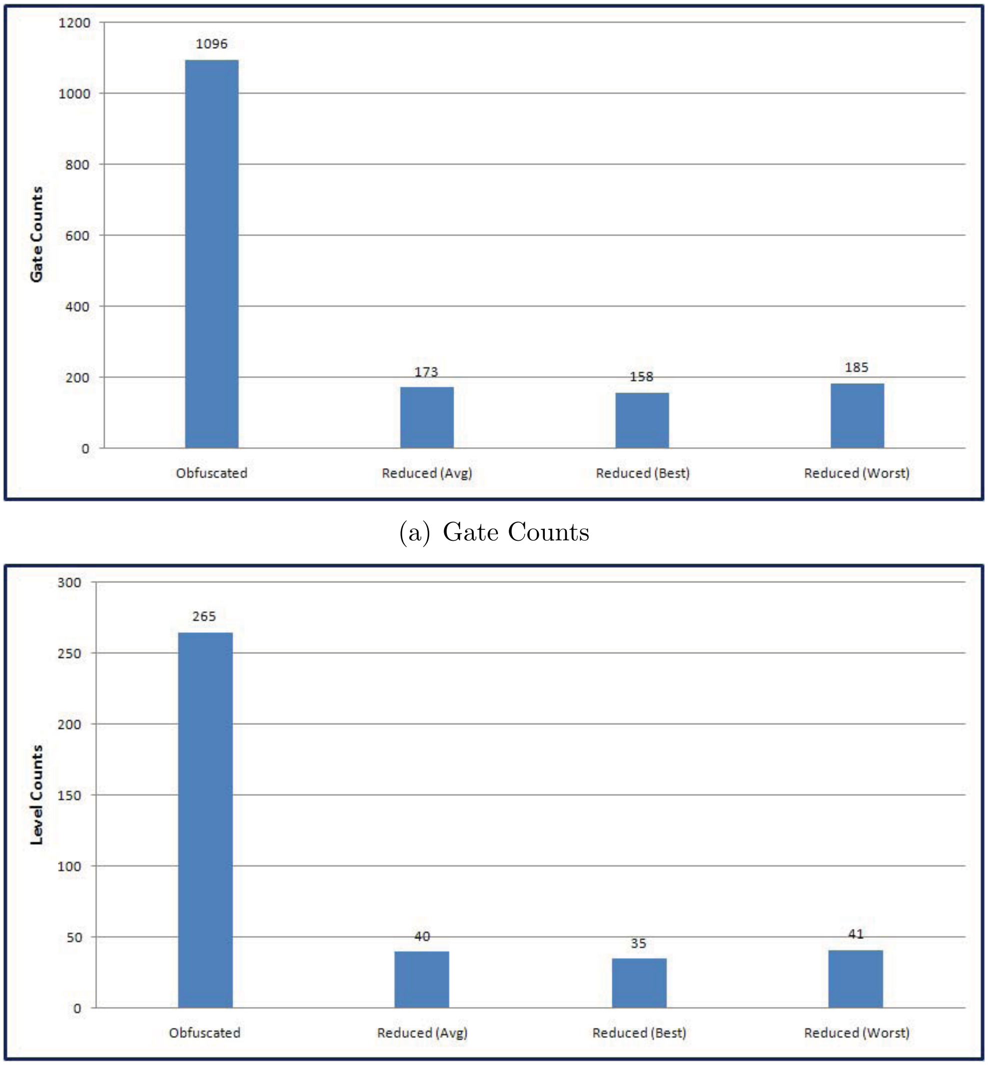
4.2.2 The Worst Reduction. As previously stated, the obfuscated circuit $R17' - C17 - R17''$ after 20 iterations with 4 gate replacement and FFFTTT library generation option generates the best reduced circuit. The metrics such as gate counts and level counts are as shown in Table 4.3. Figure 4.3 provides a plot of these data. As shown in Table 4.3 and Figure 4.3, reduction percentage of gate count is 21.05% and reduction percentage of level count is 21.43%. This statistics shows that this option brings about more powerful circuit obfuscation because our reduction algorithms reduce less gates and levels than other circuit obfuscation options.

4.3 Ordering and Reduction Round

To evaluate each reduction algorithm's ordering and reduction round, we calculate removed gate counts per each reduction round using the best case, the obfuscated circuit $C17 - C17 - C17$ after 1000 iterations with 3 gate replacement and TFTTTT library generation option described in Table 4.1 and Figure 4.1. However, the worst case, the obfuscated circuit $R17' - C17 - R17''$ after 20 iterations with 4

Table 4.2: Gate Counts and Level Counts of the Best Reduction Result ($C17 - C17 - C17$, 3/TFTTTT, 1000 iteration)

	Original	**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%)



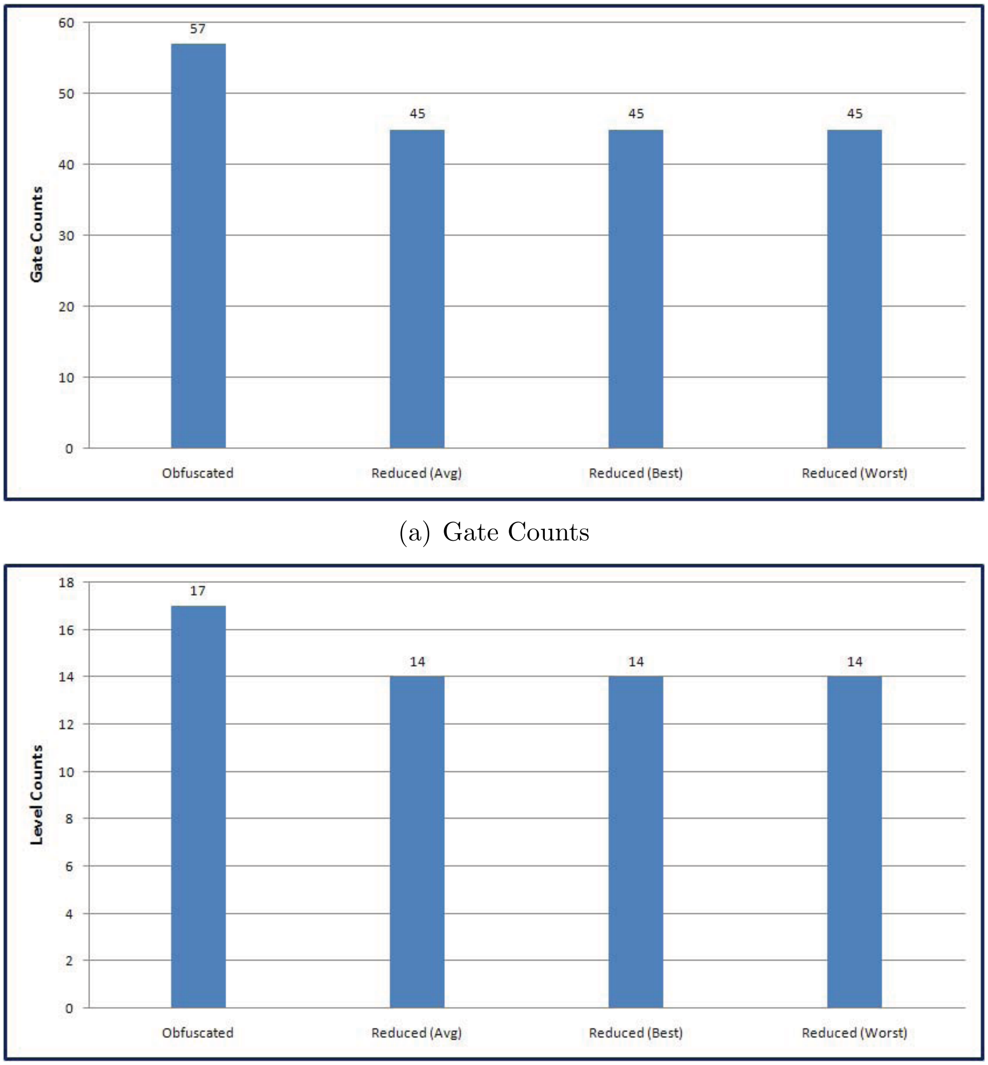
(a) Gate Counts

(b) Level Counts

Figure 4.2: Gate Counts and Level Counts of the Best Reduction Result ($C17 - C17 - C17$, 3/TFTTTT, 1000 iteration)

Table 4.3: Gate Counts and Level Counts of the Worst Reduction Result ($R17' - C17 - R17''$, 4/FFFTTT, 20 iteration)

	Original	**Reduced(Avg)**	**Reduced(Best)**	**Reduced(Worst)**
Gates	57	45(21.05%)	45(21.05%)	45(21.05%)
Levels	17	14(21.43%)	14(21.43%)	14(21.43%)



(a) Gate Counts

(b) Level Counts

Figure 4.3: Gate Counts and Level Counts of the Worst Reduction Result ($R17' - C17 - R17''$, 4/FFFTTT, 20 iteration)

gate replacement and FFFTTT library generation option, is not suitable to explore algorithm's order and reduction round because the option has too few reduced gate counts. Therefore, we choose the worst case with 1000 iterations; that is the obfuscated circuit $R17' - C17 - R17''$ after 1000 iterations with 4 gate replacement and FFFTTT library generation option.

4.3.1 The Best Reduction. In order to evaluate each reduction algorithm at each reduction round, we calculate removed gate counts after 1000 iterations using our circuit reduction algorithms with obfuscated circuit $C17 - C17 - C17$ as shown in Table 4.4(a). Figure 4.4(a) provides a plot of these data. We tried to find the best result when our circuit reduction algorithms are randomly executed. We found that the sequence 1→12→3→7→9→10→5→6→8→11→2→4 generates the best reduction result. This result also shows that algorithm 1, reduces buffer, discharged its duty during this reduction process because TFTTTT library generation option makes more buffers than FFFTTT library generation option. We can also know that some reduction algorithm releases a complex link among some gates. For example, algorithm 6, reduces two gates by making AND/NAND/OR/NOR, of the second reduction round removes two gates that can't be found in the first reduction round.

In order to evaluate each reduction algorithm at each reduction round, we calculate removed gate counts after 1000 iterations using our circuit reduction algorithms with obfuscated circuit $C17 - C17 - C17$ as shown in Table 4.4(b). Figure 4.4(b) provides a plot of these data. We tried to find the worst result when our circuit reduction algorithms are randomly executed. We found that the sequence 4→1→6→12→10→7→5→3→8→11→9→2 generates the worst reduction result. This result also shows that algorithm 2, reduces inverter, discharged its duty during this reduction process. This result supports our speculation that TFTTTT library generation option generates more redundant gates such as inverter. We also see that algorithm 10, reduces AND/OR/NAND/NOR gates that have only inverters as inputs, in the second reduction round removes two gates that can't be found in the first

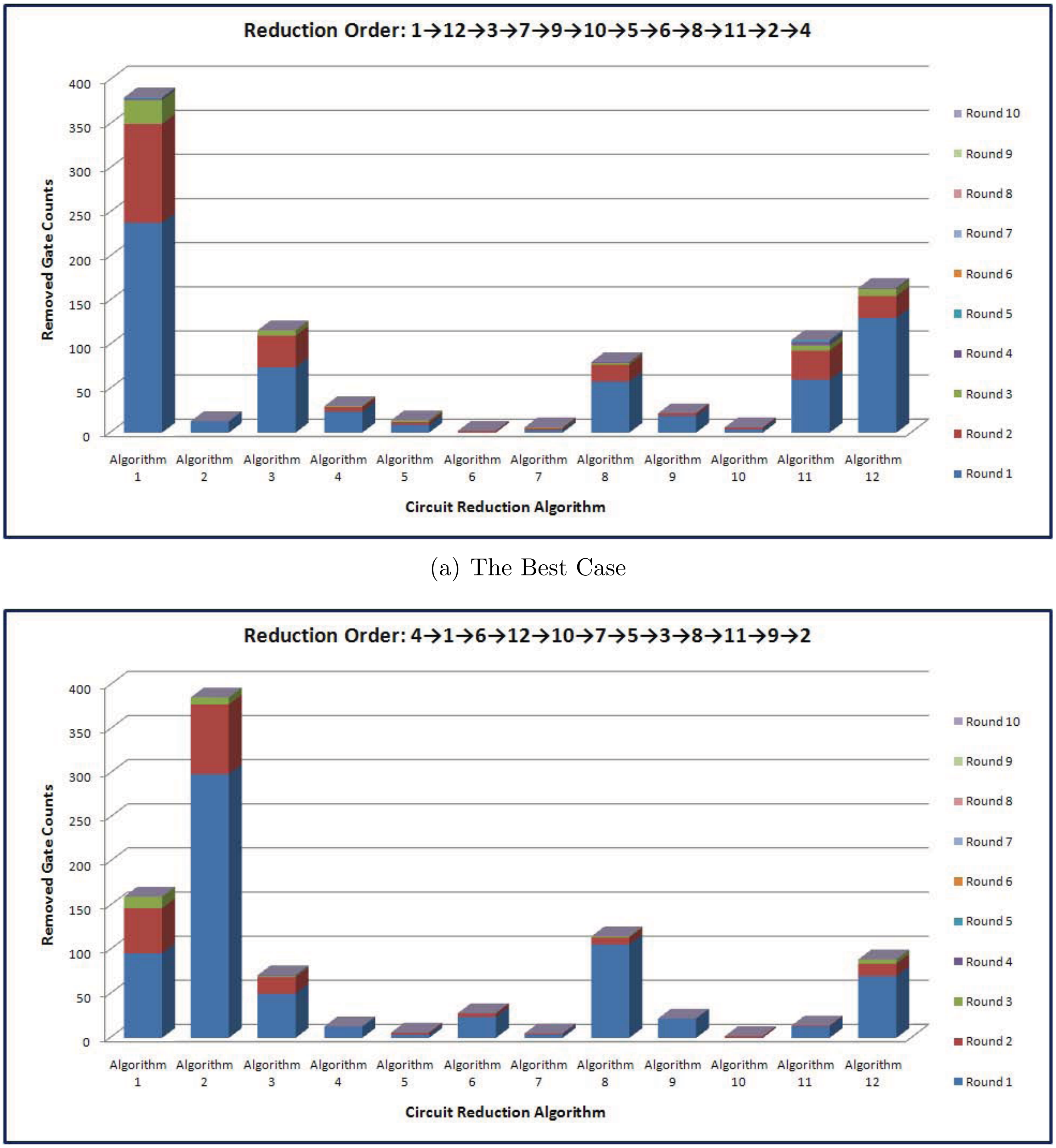
Table 4.4: Removed Gate Counts of each Reduction Algorithm using Obfuscated Circuit $C17 - C17 - C17$ with 3/TFTTTT Option

(a) The Best Case

	A1	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	**A8**	**A9**	**A10**	**A11**	**A12**
Round 1	238	13	74	24	9	0	3	58	19	4	60	130
Round 2	112	0	36	5	3	2	2	19	3	2	33	25
Round 3	27	0	6	1	2	0	1	2	0	0	6	8
Round 4	2	0	0	0	0	0	0	1	0	0	4	1
Round 5	1	0	0	0	0	0	0	0	0	0	2	0
Round 6	0	0	0	0	0	0	0	0	0	0	0	0
Round 7	0	0	0	0	0	0	0	0	0	0	0	0
Round 8	0	0	0	0	0	0	0	0	0	0	0	0
Round 9	0	0	0	0	0	0	0	0	0	0	0	0
Round 10	0	0	0	0	0	0	0	0	0	0	0	0

(b) The Worst Case

	A1	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	**A8**	**A9**	**A10**	**A11**	**A12**
Round 1	96	299	50	13	4	24	4	106	22	0	13	70
Round 2	51	79	19	0	2	4	1	8	0	2	1	14
Round 3	13	8	1	0	0	0	0	1	0	0	0	5
Round 4	0	0	1	0	0	0	0	0	0	0	0	0
Round 5	0	0	0	0	0	0	1	0	0	0	0	0
Round 6	0	0	0	0	0	0	0	0	0	0	0	0
Round 7	0	0	0	0	0	0	0	0	0	0	0	0
Round 8	0	0	0	0	0	0	0	0	0	0	0	0
Round 9	0	0	0	0	0	0	0	0	0	0	0	0
Round 10	0	0	0	0	0	0	0	0	0	0	0	0



(a) The Best Case

(b) The Worst Case

Figure 4.4: Removed Gate Counts of each Reduction Algorithm using Obfuscated Circuit $C17 - C17 - C17$ with 3/TFTTTT

reduction round. Therefore, it verifies some reduction algorithm unties a complicated link among some gates.

4.3.2 The Worst Reduction. In order to evaluate each reduction algorithm at each reduction round, we calculate removed gate counts after 1000 iterations using our circuit reduction algorithms with obfuscated circuit $R17' - C17 - R17''$ as shown in Table 4.5(a). Figure 4.5(a) provides a plot of these data. We tried to find the best result when our circuit reduction algorithms are randomly executed. We found that the sequence 12→3→5→6→10→2→11→9→8→1→4→7 generates the best reduction result. This result also shows that algorithm 7, reduces two gates by making buffer-/NOT/constant, discharged its duty during this reduction process. This supports that FFFTTT library generation option generates more obfuscation pattern than redundant patterns including buffer/inverter. As described in the previous section, we can also know that some reduction algorithm releases a complex connection among some gates. For example, algorithm 2, reduces inverter, of the fifth reduction round removes one gate that can't be found in the fourth reduction round.

In order to evaluate each reduction algorithm at each reduction round, we calculate removed gate counts after 1000 iterations using our circuit reduction algorithms with obfuscated circuit $R17' - C17 - R17''$ as shown in Table 4.5(b). Figure 4.5(b) provides a plot of these data. We tried to find the worst result when our circuit reduction algorithms are randomly executed. We found that the sequence 4→9→7→11→10→8→2→5→1→12→3→6 generates the worst reduction result. This result also shows that algorithm 4, reduces constant 0/1, discharged its duty during this reduction process. This result supports our speculation that FFFTTT library generation option generates more complex obfuscation patterns than other obfuscation options. We also see that algorithm 1, reduces buffer, in the second reduction round removes twelve gates that can't be found in the first reduction round. Therefore, it verifies some reduction algorithm unties a complicated link among some gates.

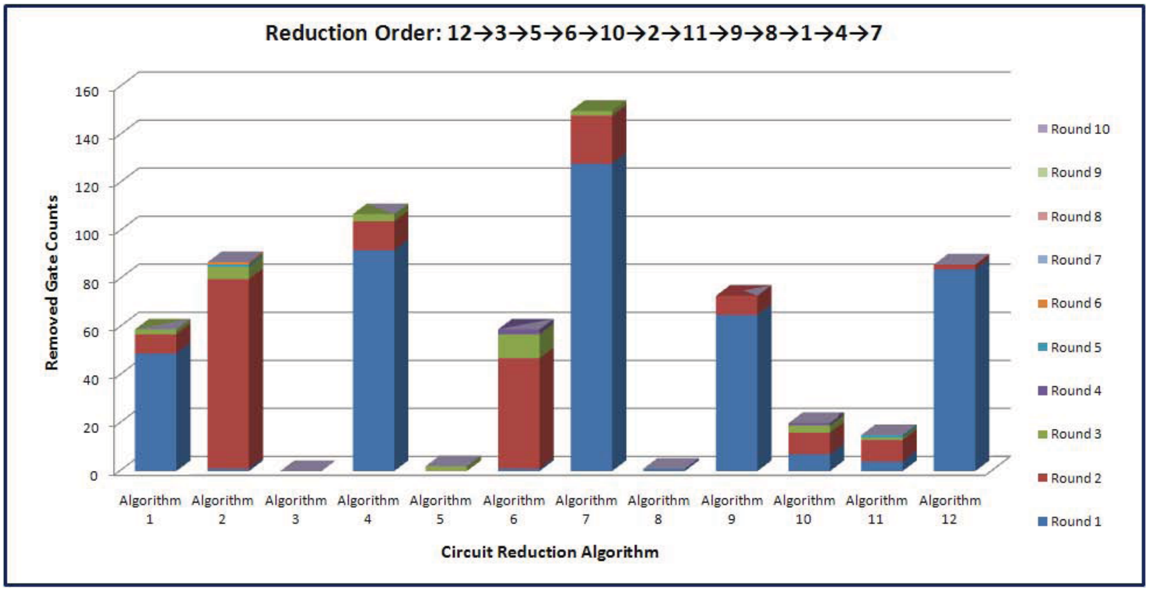
Table 4.5: Removed Gate Counts of each Reduction Algorithm using Obfuscated Circuit $R17' - C17 - R17''$ with 4/FFFTTT Option

(a) The Best Case

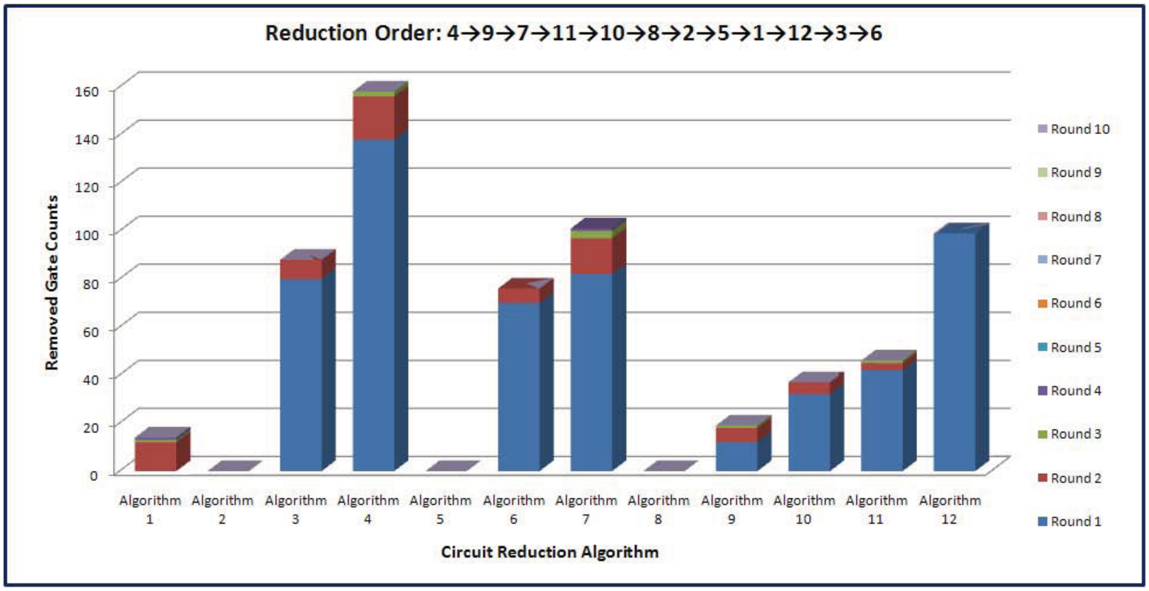
	A1	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	**A8**	**A9**	**A10**	**A11**	**A12**
Round 1	49	1	0	92	0	1	128	1	65	7	4	84
Round 2	8	79	0	12	0	46	20	0	8	9	9	2
Round 3	2	5	0	3	2	10	2	0	0	3	1	0
Round 4	0	0	0	0	0	2	0	0	0	1	0	0
Round 5	0	1	0	0	0	0	0	0	0	0	1	0
Round 6	0	1	0	0	0	0	0	0	0	0	0	0
Round 7	0	0	0	0	0	0	0	0	0	0	0	0
Round 8	0	0	0	0	0	0	0	0	0	0	0	0
Round 9	0	0	0	0	0	0	0	0	0	0	0	0
Round 10	0	0	0	0	0	0	0	0	0	0	0	0

(b) The Worst Case

	A1	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	**A8**	**A9**	**A10**	**A11**	**A12**
Round 1	0	0	80	138	0	70	82	0	12	32	42	99
Round 2	12	0	8	18	0	6	15	0	6	5	3	0
Round 3	1	0	0	2	0	0	3	0	1	0	1	0
Round 4	1	0	0	0	0	0	1	0	0	0	0	0
Round 5	0	0	0	0	0	0	0	0	0	0	0	0
Round 6	0	0	0	0	0	0	0	0	0	0	0	0
Round 7	0	0	0	0	0	0	0	0	0	0	0	0
Round 8	0	0	0	0	0	0	0	0	0	0	0	0
Round 9	0	0	0	0	0	0	0	0	0	0	0	0
Round 10	0	0	0	0	0	0	0	0	0	0	0	0



(a) The Best Case



(b) The Worst Case

Figure 4.5: Removed Gate Counts of each Reduction Algorithm using Obfuscated Circuit $R17' - C17 - R17''$ with 4/FFFTTT

Table 4.6: Remaining Redundant Gate Counts in the Reduced Circuit Variants

(a) Reduced Circuit $C17 - C17 - C17$

	3/FFFTTT	**3/TFTTTT**	**4/FFFTTT**	**4/TFTTTT**
Buffer	0	0	0	0
Inverter	4	25	18	83
Constant 0/1	0	19	0	21

(b) Reduced Circuit $R17' - C17 - R17''$

	3/FFFTTT	**3/TFTTTT**	**4/FFFTTT**	**4/TFTTTT**
Buffer	0	0	0	0
Inverter	1	28	15	88
Constant 0/1	0	12	0	26

4.4 Evaluation

4.4.1 Remaining Gates. As shown in Table 4.6, all buffers in the obfuscated circuit variants are removed. However, there are some remaining inverters and constants after applying our circuit reduction algorithms. Why are these gates in the reduced circuit variants? The answer is that the connection between the gate and other gates is tangled in accordance with a repeated circuit obfuscation. Therefore, we can know that the obfuscation iteration is a critical factor about the circuit reduction.

4.4.2 Obfuscation Options. Until now, we explored the circuit obfuscation option such as gate replacement size, library generation option and obfuscation iteration and then acquired evaluation data about the circuit obfuscation option's strength as shown in Table 4.7.

First of all, the four gate replacement size generates a stronger obfuscated circuit than three gate replacement size because it four gate replacement size brings about complex entanglement among each gate of the obfuscated circuit variant. Secondly, the FFFTTT library generation option makes a robust obfuscated circuit than TFTTTT library generation option because FFFTTT option has less reducible gates than TFTTTT option. Lastly, more obfuscation iterations produces a powerful obfus-

Table 4.7: Evaluation of Circuit Obfuscation Option's Strength

	Comparison of Circuit Obfuscation Options
Gate Replacement Size	$4 > 3$
Library Generation Option	$FFFTTT > TFTTTT$
Obfuscation Iteration	$1000 > 500 > 100 > 50 > 20$

cated circuit than fewer obfuscation iterations because more obfuscation causes more twisted connection among with each gate than fewer obfuscation.

V. Conclusions

In this chapter, we summarize the contribution of our research effort and provide recommendations for future work in the area of circuit reduction and obfuscation analysis.

5.1 Contributions

5.1.1 Characterizing Circuit Obfuscation. We present a concrete characterization of circuit obfuscation based on knowledge of the obfuscating engine and application of basic rules of Boolean logic. We present in this research a comprehensive experimental framework that considers a wide range of experimental options used to produce functionally equivalent circuit variants from an obfuscating engine. We use our reduction experiments to characterize obfuscation according to several factors: gate replacement size (three and four gate replacement), subcircuit generation options ($FFFTTT$/standard and $TFTTTT$/redundant), and iteration number (20/50/100/500/1000). We show how such options affect reduction and provide general conclusions along these configuration lines.

5.1.2 Applying and Ordering Circuit Reduction Algorithms. We present a concrete set of reduction algorithms using basic structural patterns and truth-table logic forms. These algorithms form the basis for a larger characterization suite which may be used to analyze circuits variants produced from a wide-variety of circuit obfuscating engines. Though we apply our analysis to the *CORGI* generation engine, our methodology works equally well on any obfuscator that transforms combinational logic gates and provides a concrete measurement scheme. We also show how ordering of individual algorithms produces higher levels of reduction and give insight as to why certain orderings are preferable. Our data provides insight into lower and upper bounds for reduction rounds and gives us average case measures for trends in obfuscation strength.

5.2 Future Work

5.2.1 Improve Reduction Algorithm for the Remaining Gates. Our set of reduction algorithms does not remove some types of redundant gates such as inverters and constants where those gates have complicated links between gates. However, since the circuits of interest in our experimental framework ($C17 - C17 - C17$ and $R17' - C17 - R17''$) did not include those specific gate patterns, it is possible we may reduce them with other patterns and reduction algorithms if they are present in other circuit forms. Future new patterns and reduction algorithms would need to consider complicated and twisted connections among gates in the circuit. Future patterns would also need to consider recursive combinations of algorithms since the current reduction techniques only uses a single pass through the gate list of the circuit. We expect that such modifications would provide further gate size reduction on the same sample set.

5.2.2 Increase Search Space. One drawback to our current algorithmic approach is that our search space for finding patterns is limited. Since we have limited memory and processing time available, we cannot use best case algorithms that would fully explore all possible patterns and pattern size spaces. Our current patterns do not extend the boundaries of the search beyond a small number of successor and predecessor gates. For example, our circuit reduction algorithms examine subcircuits with only two/three/four gates. However, if we allow an expansion of search space to catch larger subcircuits, our reduction techniques may reduce more patterns. We expect future research would characterize the tradeoff space (in terms of computational power/time/memory) of increasing the search space and correlate that to higher size reduction levels. Ultimately, such characterization would give a more concrete relation to the amount of work an adversary would need to exert to accomplish reverse engineering or circuit analysis.

5.2.3 Increase Metric Space. In this research, we only consider gate size for evaluating obfuscation effect. Future research should incorporate a wide variety of metrics and consider measures which have more clear correlation to specific reverse engineering goals.

5.2.4 Increase Circuit Space. In this research, we only use two very small and basic circuits. Future research would widen the set of candidate circuits (starting with those in the ISCAS-85 Benchmark set) and produce variants along the same lines that are laid out in this experimental framework. This analysis would provide a much more general understanding of how the type of circuit may specifically effect both obfuscation and reduction.

5.2.5 Generate Colored Graph of Reduced Circuit. In his research, Williams [20] introduces a technique that provides a quick, visual description of what *CORGI* iterations accomplish through the use of colors, shape, numbers and ancestry values. However, the graphs do not support coloring methods when circuit reduction techniques are used. Since William's effort centers on measuring ancestral entropy, our techniques for reduction would provide an independent verification method to compare entropy values against. If we improve William's colored graph to show the coloring of reduced circuits, we can also compare a reduced circuit with an original circuit to visually understand the effect of obfuscation algorithms. As future work, we would also seek to correlate the entropy values of William's to our reduction statistics.

Appendix A. Circuit Graphs for $C17 - C17 - C17$

A.1 Obfuscated Circuit Variant Graphs

1. *Obfuscated* $C17$–$C17$–$C17$ *(3 gates replacement, FFFTTT)*: Figure B.1, Figure B.2, $\cdots$, and Figure B.5 display graphs of obfuscated circuit variants that we use in this research. The RandomAlgorithm strategy is applied to the original circuits with 20, 50, 100, 500, and 1000 iterations. We also select `three` gates as a replacement size, and then choose FFFTTT as a library generation option.

2. *Obfuscated* $C17 - C17 - C17$ *(3 gates replacement, TFTTTT)*: Figure B.6, Figure B.7, $\cdots$, and Figure B.10 display graphs of obfuscated circuit variants that we use in this research. The RandomAlgorithm strategy is applied to the original circuits with 20, 50, 100, 500, and 1000 iterations. We also select `three` gates as a replacement size, and then choose TFTTTT as a library generation option.

3. *Obfuscated* $C17 - C17 - C17$ *(4 gates replacement, FFFTTT)*: Figure B.11, Figure B.12, $\cdots$, and Figure B.15 display graphs of obfuscated circuit variants that we use in this research. The RandomAlgorithm strategy is applied to the original circuits with 20, 50, 100, 500, and 1000 iterations. We also select `four` gates as a replacement size, and then choose FFFTTT as a library generation option.

4. *Obfuscated* $C17 - C17 - C17$ *(4 gates replacement, TFTTTT)*: Figure B.16, Figure B.17, $\cdots$, and Figure B.20 display graphs of obfuscated circuit variants that we use in this research. The RandomAlgorithm strategy is applied to the original circuits with 20, 50, 100, 500, and 1000 iterations. We also select `four` gates as a replacement size, and then choose TFTTTT as a library generation option.

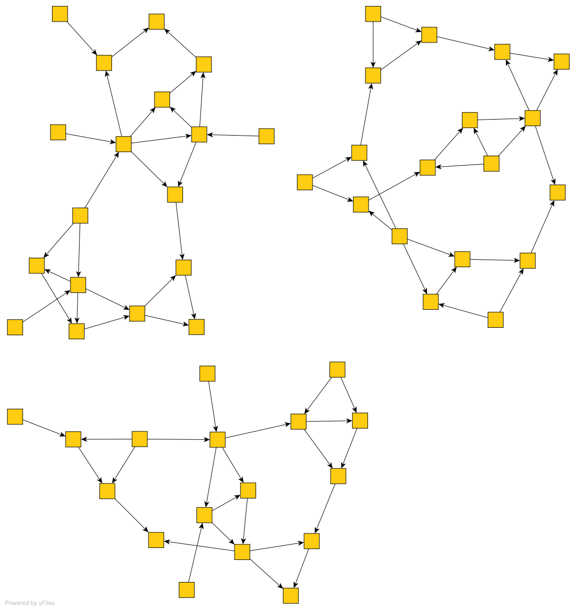


Figure A.1: Obfuscated Circuit Variant Graph after applying 20 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $C17 + C17 + C17$. (38 gates, 6 levels)

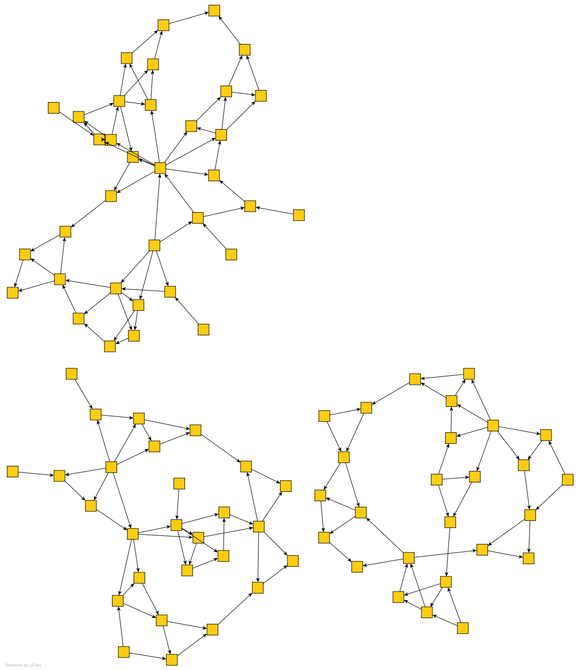


Figure A.2: Obfuscated Circuit Variant Graph after applying 50 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $C17 + C17 + C17$. (73 gates, 11 levels)

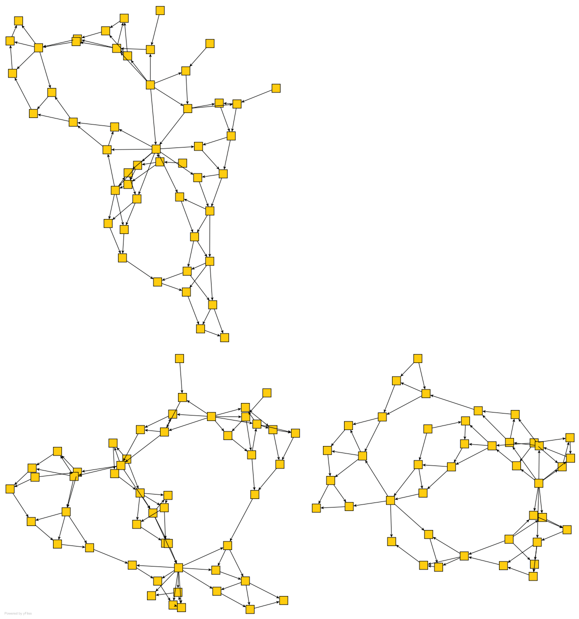


Figure A.3: Obfuscated Circuit Variant Graph after applying 100 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $C17$+$C17$+$C17$. (125 gates, 21 levels)

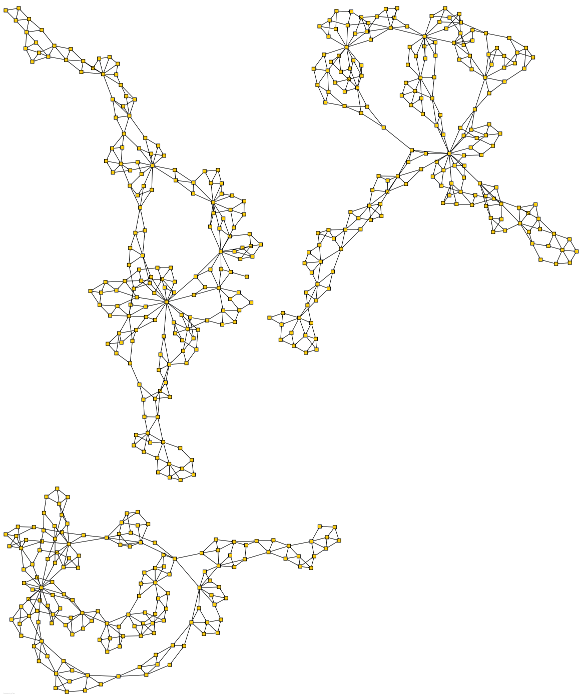


Figure A.4: Obfuscated Circuit Variant Graph after applying 500 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $C17$+$C17$+$C17$. (500 gates, 84 levels)

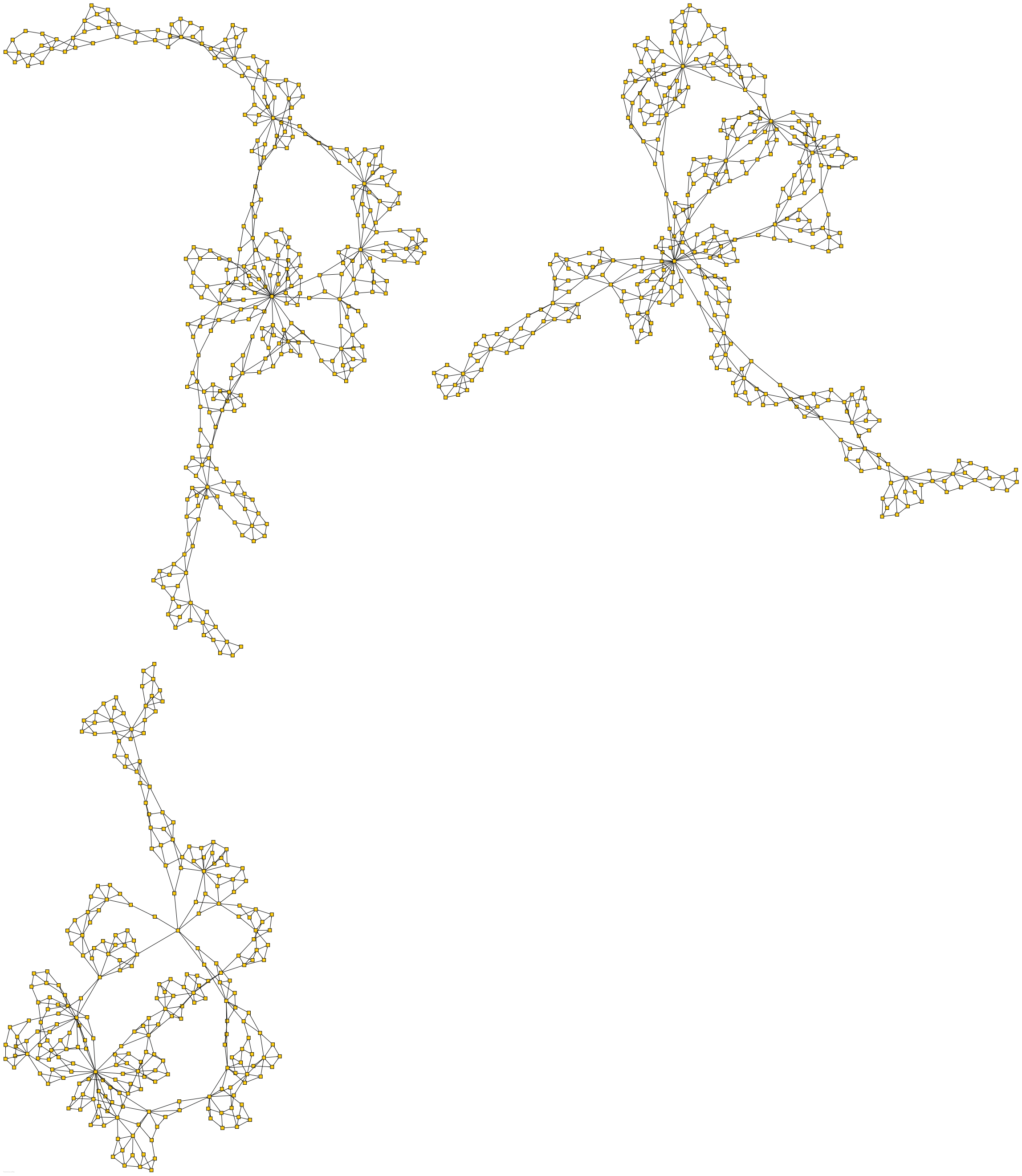


Figure A.5: Obfuscated Circuit Variant Graph after applying 1000 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $C17$+$C17$+$C17$. (943 gates, 155 levels)

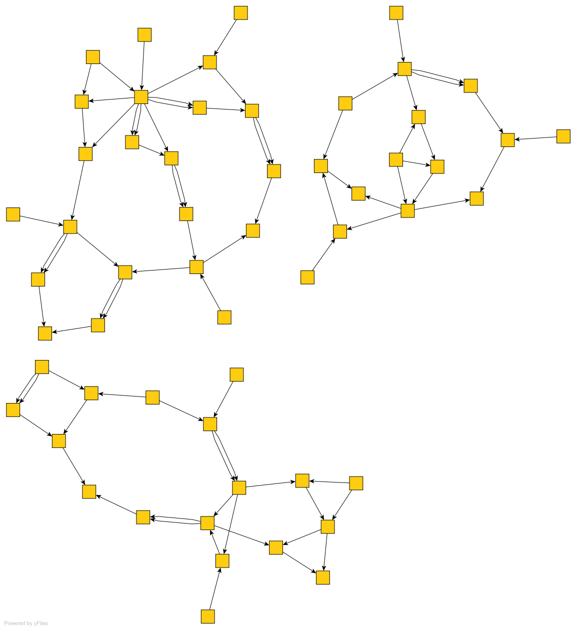


Figure A.6: Obfuscated Circuit Variant Graph after applying 20 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $C17 + C17 + C17$. (40 gates, 8 levels)

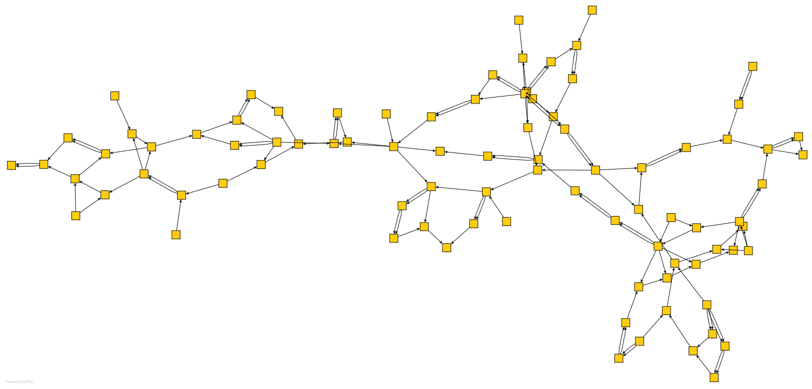


Figure A.7: Obfuscated Circuit Variant Graph after applying 50 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $C17 + C17 + C17$. (73 gates, 15 levels)

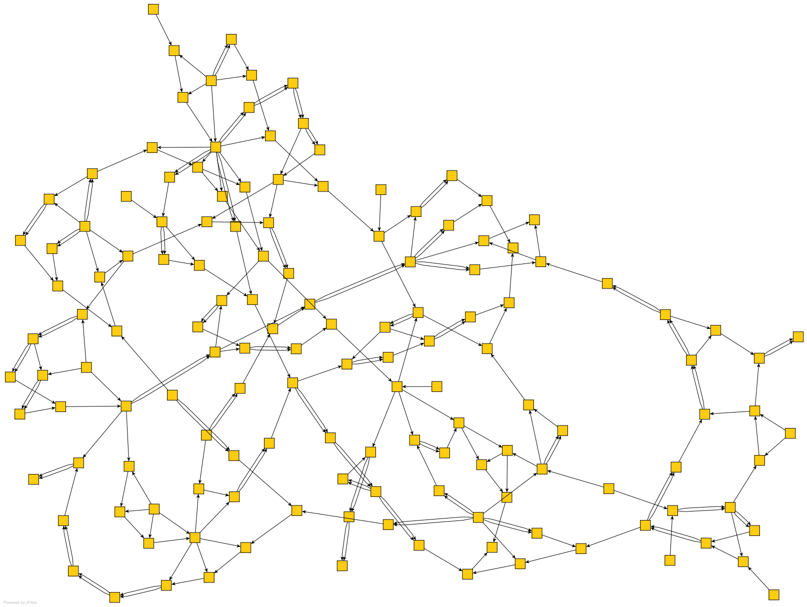


Figure A.8: Obfuscated Circuit Variant Graph after applying 100 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $C17$+$C17$+$C17$. (127 gates, 28 levels)

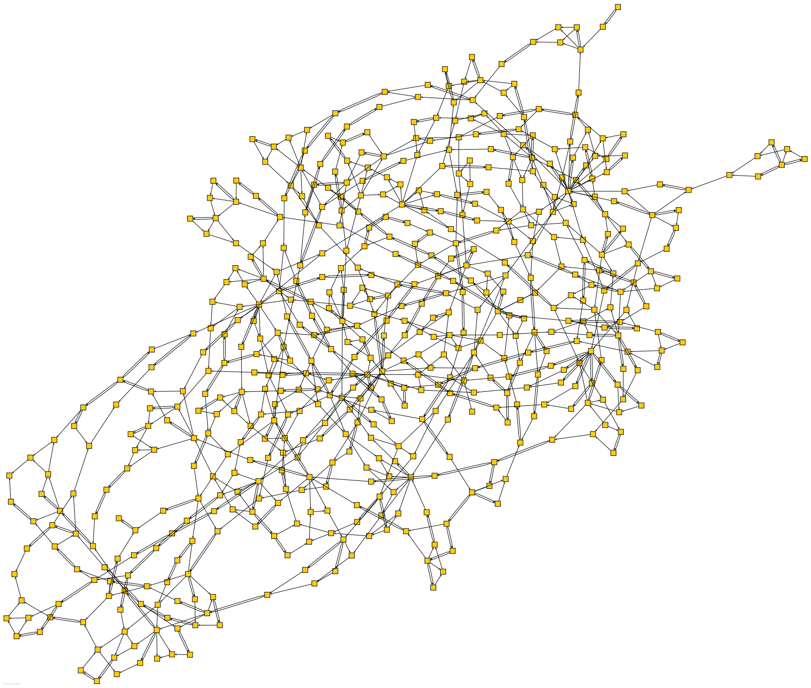


Figure A.9: Obfuscated Circuit Variant Graph after applying 500 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $C17$+$C17$+$C17$. (550 gates, 127 levels)

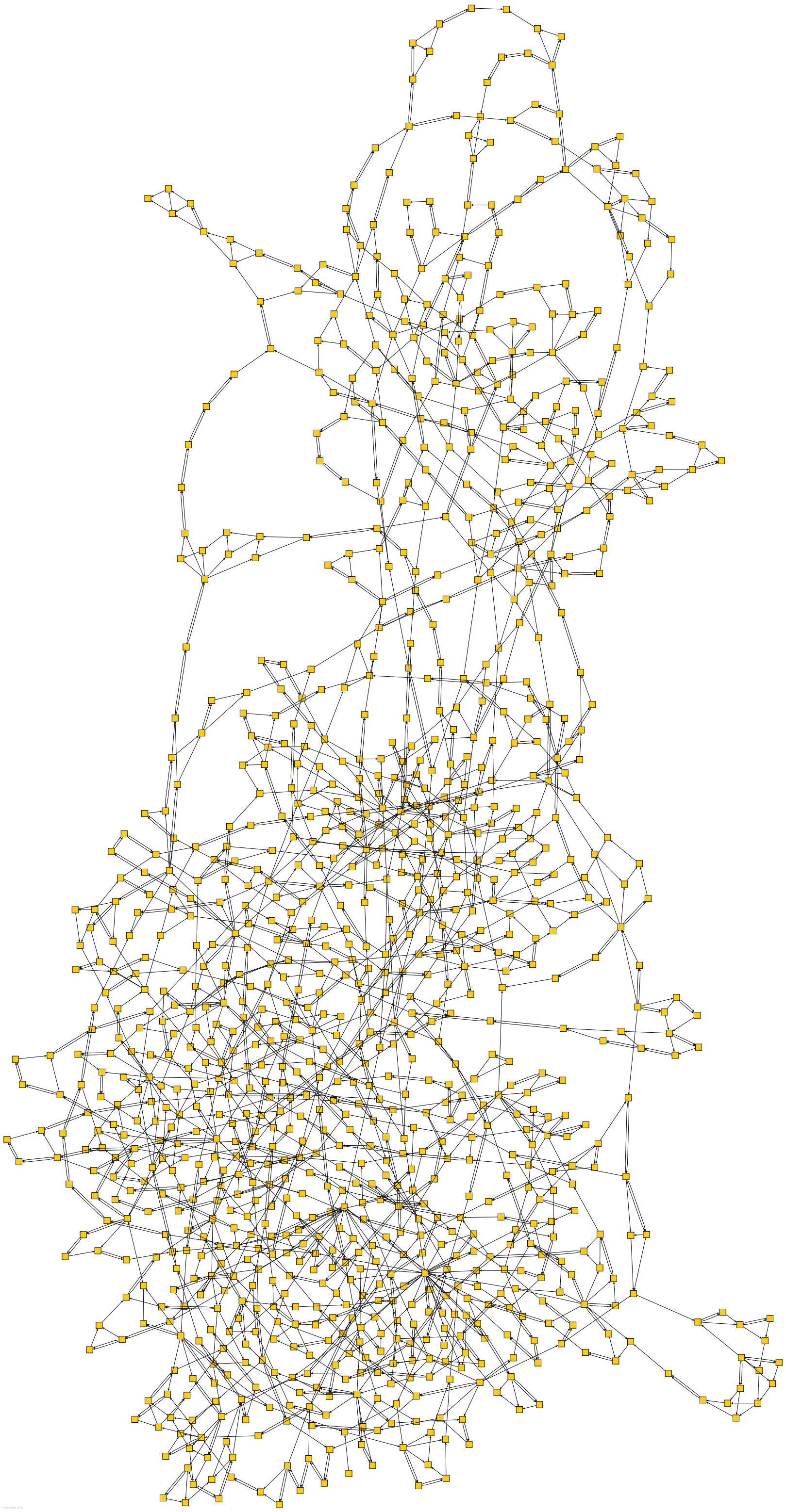


Figure A.10: Obfuscated Circuit Variant Graph after applying 1000 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $C17$+$C17$+$C17$. (1096 gates, 265 levels)

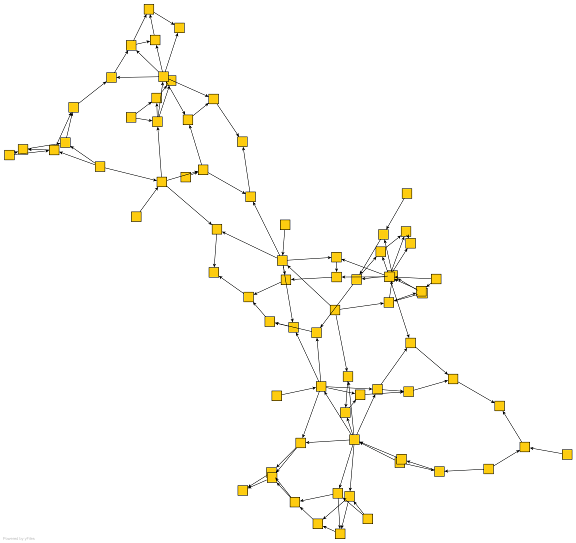


Figure A.11: Obfuscated Circuit Variant Graph after applying 20 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $C17+C17+C17$. (60 gates, 10 levels)

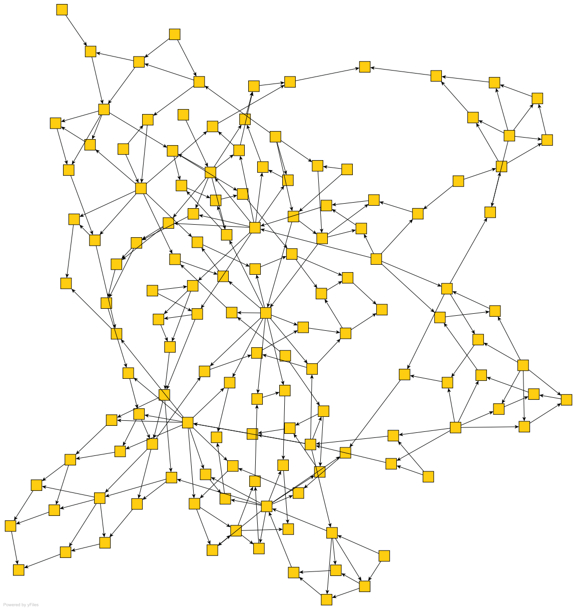


Figure A.12: Obfuscated Circuit Variant Graph after applying 50 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $C17+C17+C17$. (123 gates, 22 levels)

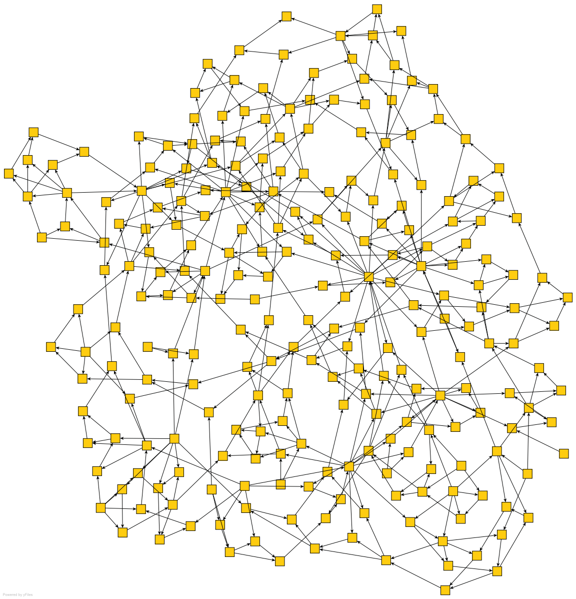


Figure A.13: Obfuscated Circuit Variant Graph after applying 100 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $C17+C17+C17$. (229 gates, 41 levels)

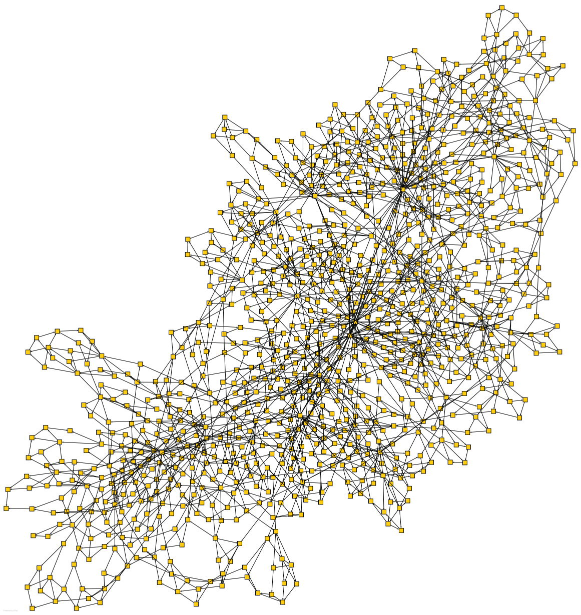


Figure A.14: Obfuscated Circuit Variant Graph after applying 500 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $C17+C17+C17$. (1074 gates, 322 levels)

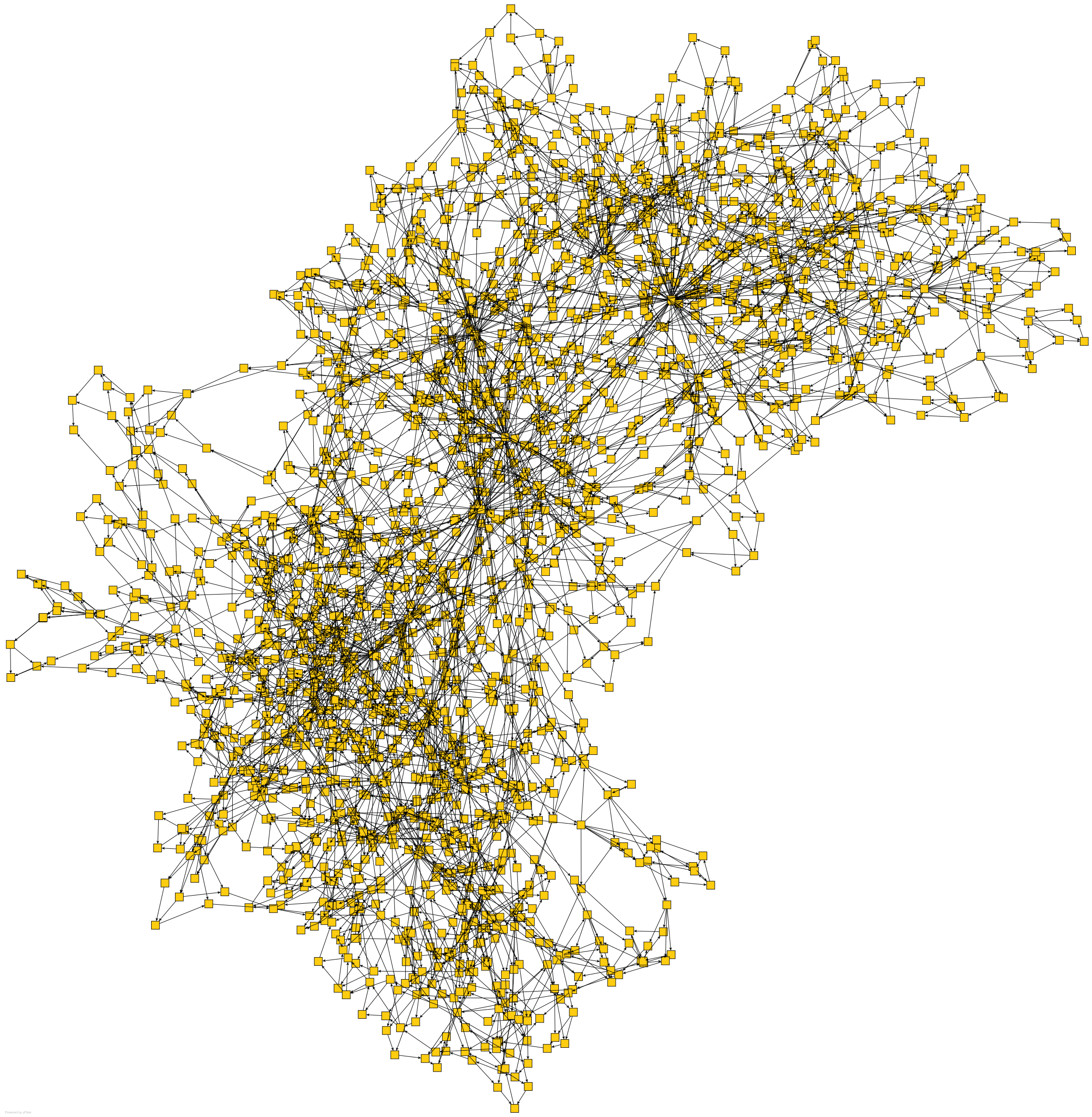


Figure A.15: Obfuscated Circuit Variant Graph after applying 1000 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $C17{+}C17{+}C17$. (2129 gates, 692 levels)

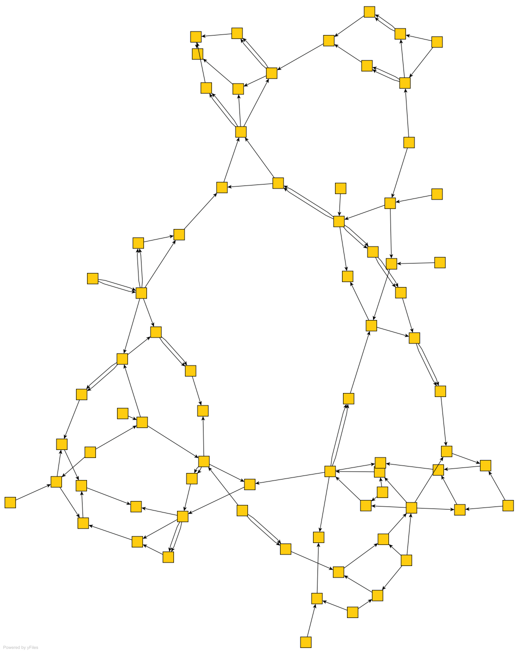


Figure A.16: Obfuscated Circuit Variant Graph after applying 20 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $C17+C17+C17$. (59 gates, 14 levels)

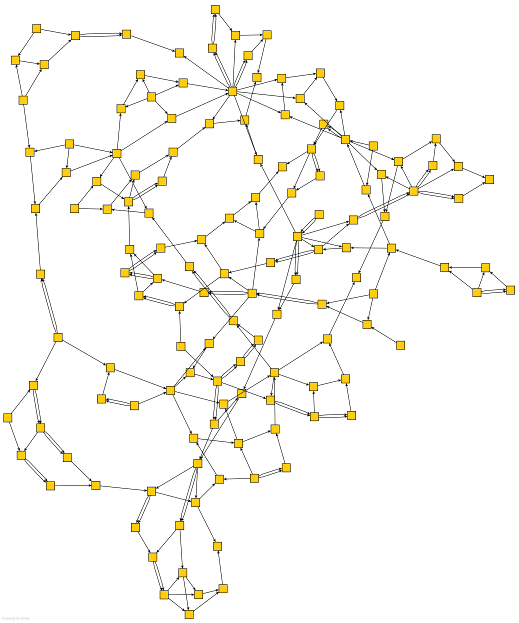


Figure A.17: Obfuscated Circuit Variant Graph after applying 50 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $C17+C17+C17$. (122 gates, 25 levels)

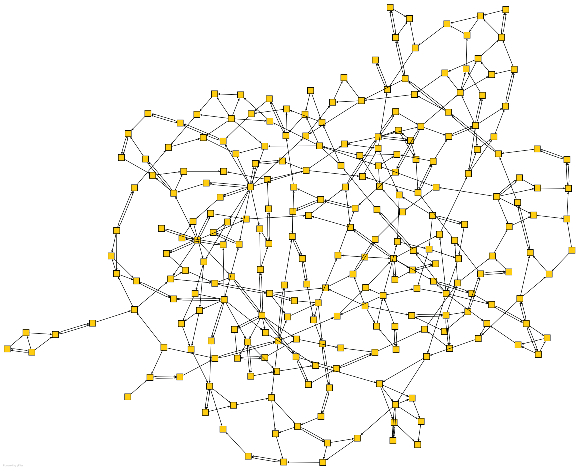


Figure A.18: Obfuscated Circuit Variant Graph after applying 100 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $C17+C17+C17$. (229 gates, 57 levels)

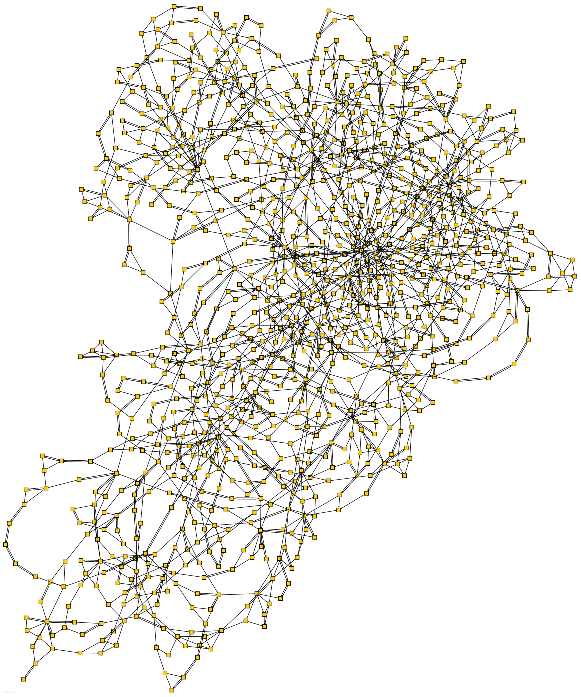


Figure A.19: Obfuscated Circuit Variant Graph after applying 500 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $C17+C17+C17$. (1054 gates, 290 levels)

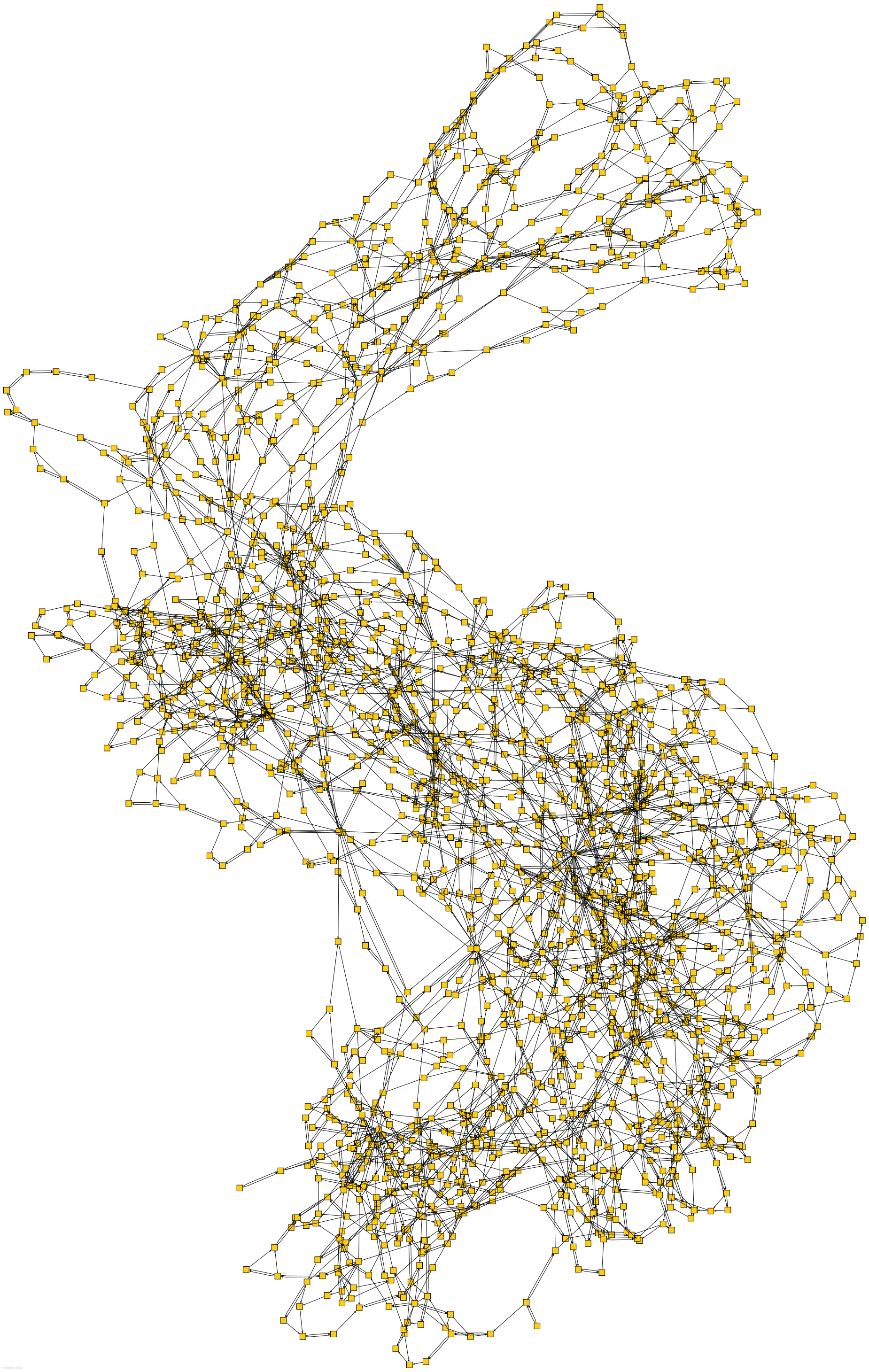


Figure A.20: Obfuscated Circuit Variant Graph after applying 1000 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $C17+C17+C17$. (2099 gates, 605 levels)

A.2 Reduced Circuit Variant Graphs

1. *Reduced C17-C17-C17 (3 gates replacement, FFFTTT)*: Figure B.21, Figure B.22, · · ·, and Figure B.25 display graphs of reduced circuit variants that we produce in this research. The twelve circuit reduction algorithms are randomly applied to the obfuscated circuit variants with 10 reduction rounds.

2. *Reduced C17-C17-C17 (3 gates replacement, TFTTTT)*: Figure B.26, Figure B.27, · · ·, and Figure B.30 display graphs of reduced circuit variants that we produce in this research. The twelve circuit reduction algorithms are randomly applied to the obfuscated circuit variants with 10 reduction rounds.

3. *Reduced C17-C17-C17 (4 gates replacement, FFFTTT)*: Figure B.31, Figure B.32, · · ·, and Figure B.35 display graphs of reduced circuit variants that we produce in this research. The twelve circuit reduction algorithms are randomly applied to the obfuscated circuit variants with 10 reduction rounds.

4. *Reduced C17-C17-C17 (4 gates replacement, TFTTTT)*: Figure B.36, Figure B.37, · · ·, and Figure B.40 display graphs of reduced circuit variants that we produce in this research. The twelve circuit reduction algorithms are randomly applied to the obfuscated circuit variants with 10 reduction rounds.

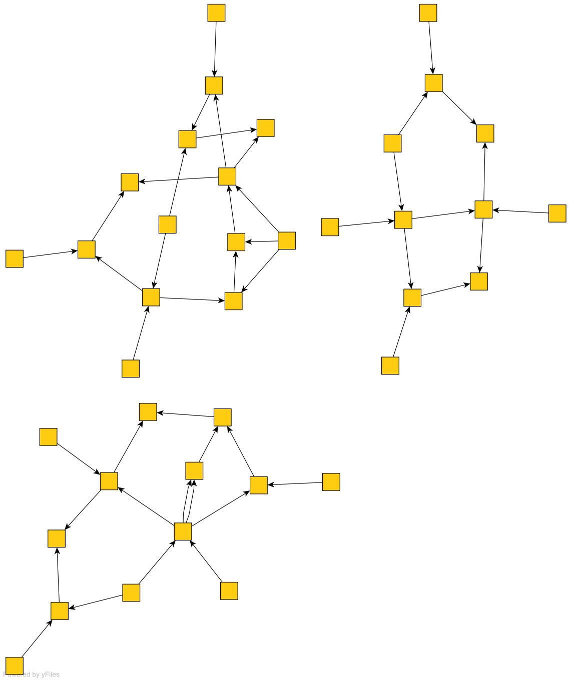


Figure A.21: Reduced Circuit Variant Graph after applying 20 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $C17 + C17 + C17$. (21 gates, 4 levels)

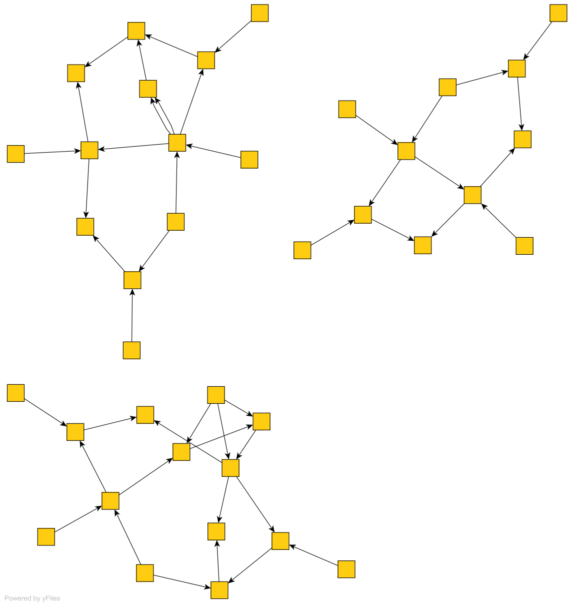


Figure A.22: Reduced Circuit Variant Graph after applying 50 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $C17 + C17 + C17$. (30 gates, 8 levels)

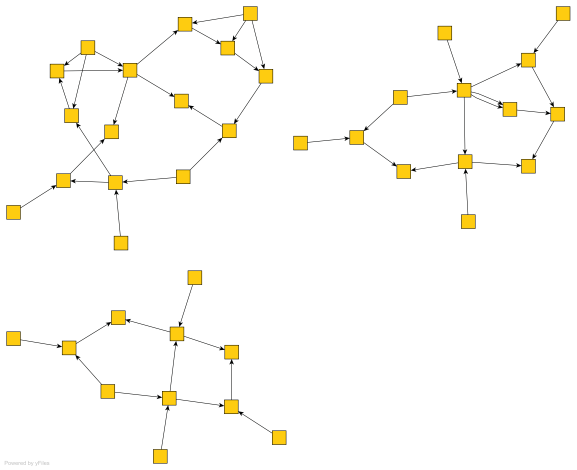


Figure A.23: Reduced Circuit Variant Graph after applying 100 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $C17 + C17 + C17$. (49 gates, 10 levels)

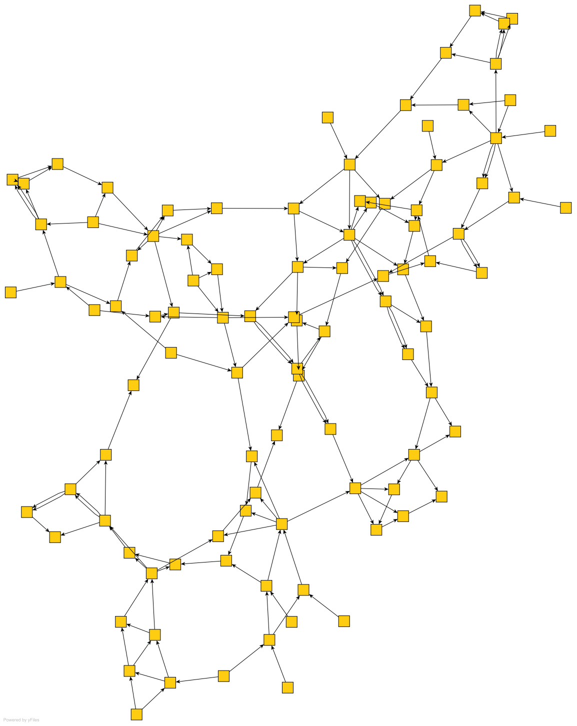


Figure A.24: Reduced Circuit Variant Graph after applying 500 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $C17 + C17 + C17$. (100 gates, 19 levels)

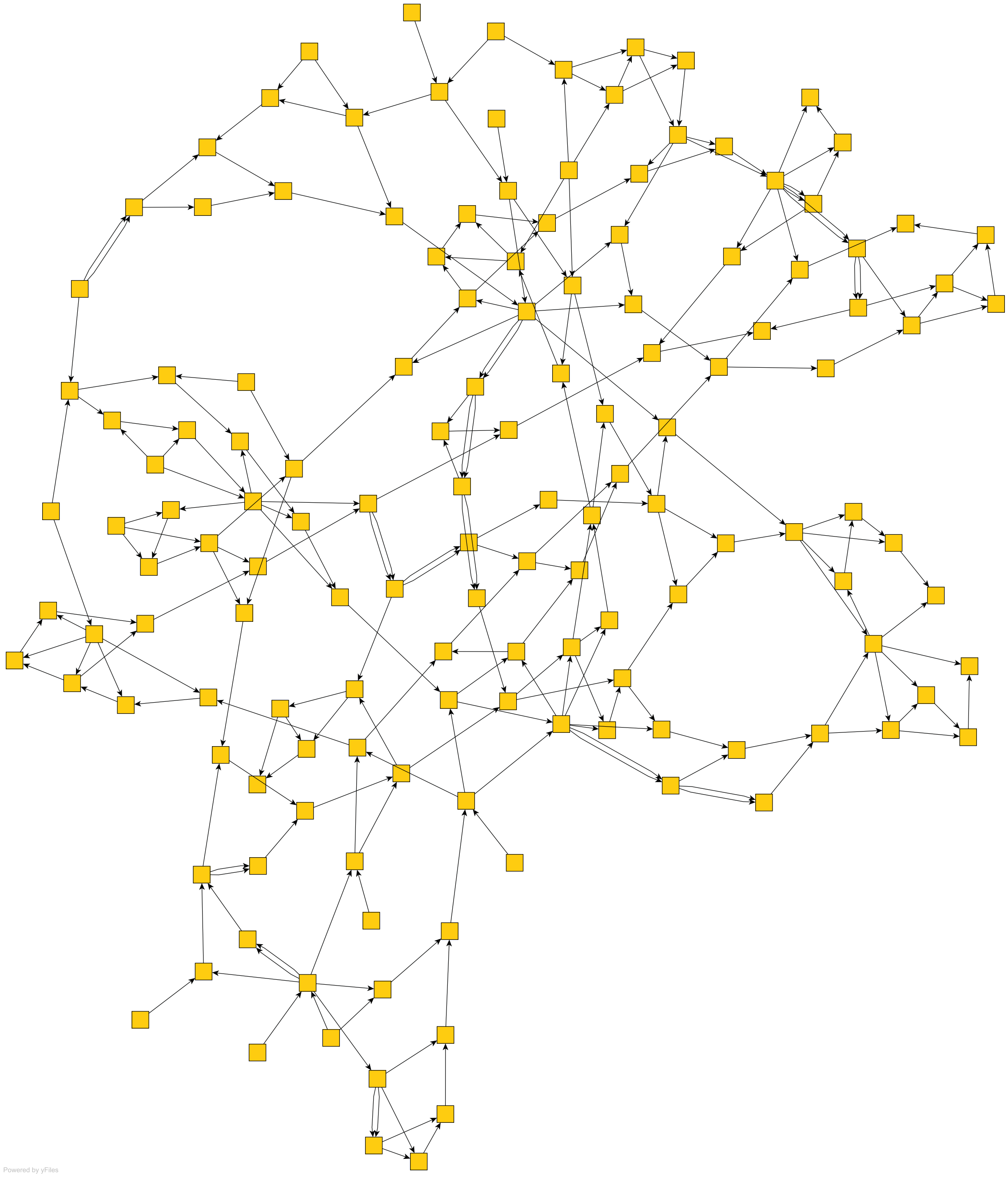


Figure A.25: Reduced Circuit Variant Graph after applying 1000 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $C17 + C17 + C17$. (192 gates, 43 levels)

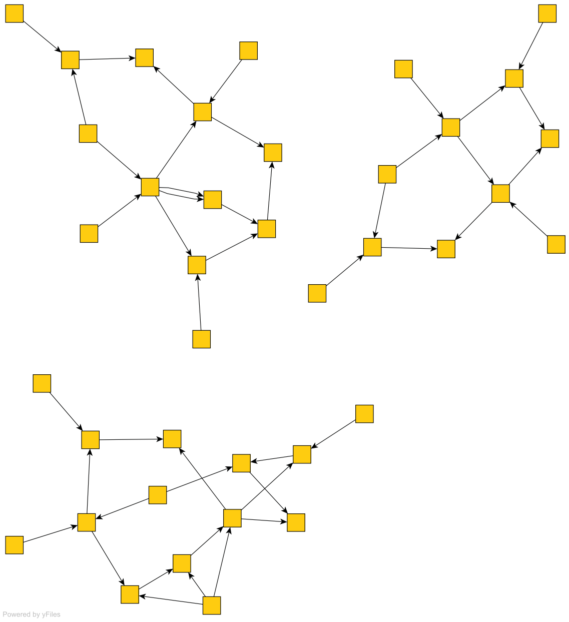


Figure A.26: Reduced Circuit Variant Graph after applying 20 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $C17 + C17 + C17$. (23 gates, 7 levels)

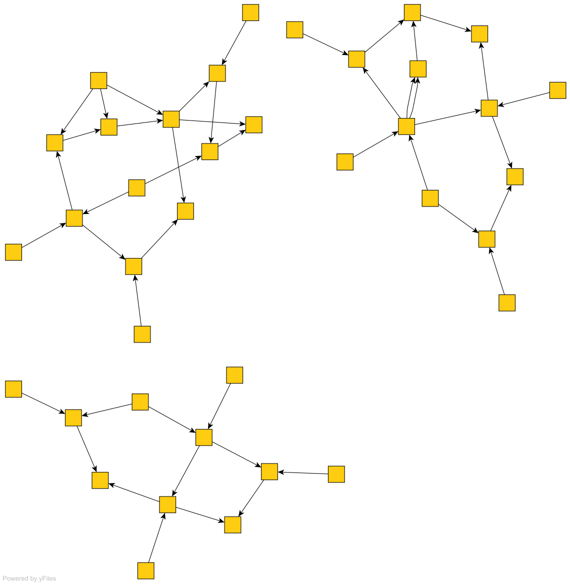


Figure A.27: Reduced Circuit Variant Graph after applying 50 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $C17 + C17 + C17$. (23 gates, 7 levels)

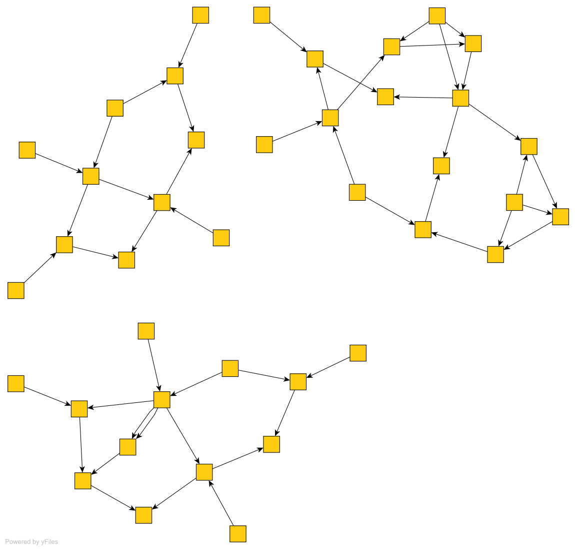


Figure A.28: Reduced Circuit Variant Graph after applying 100 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $C17+C17+C17$. (25 gates, 9 levels)

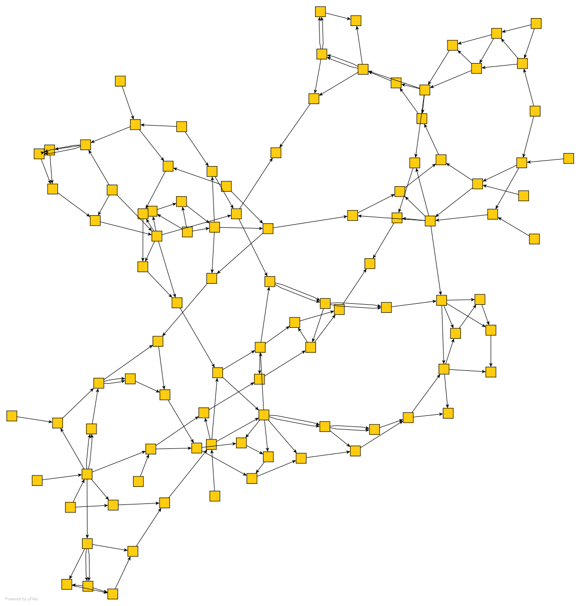


Figure A.29: Reduced Circuit Variant Graph after applying 500 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $C17+C17+C17$. (83 gates, 26 levels)

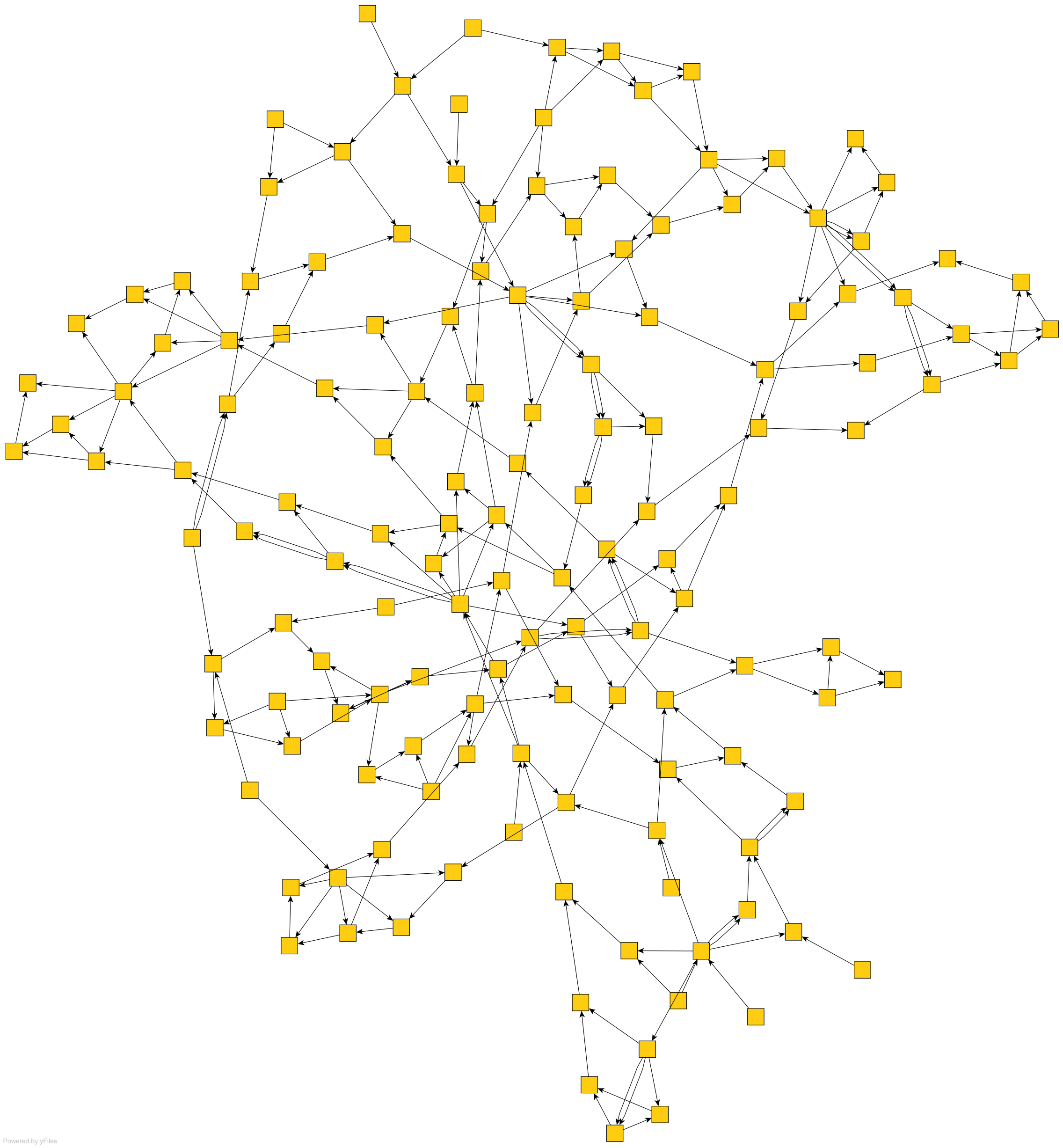


Figure A.30: Reduced Circuit Variant Graph after applying 1000 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $C17 + C17 + C17$. (123 gates, 30 levels)

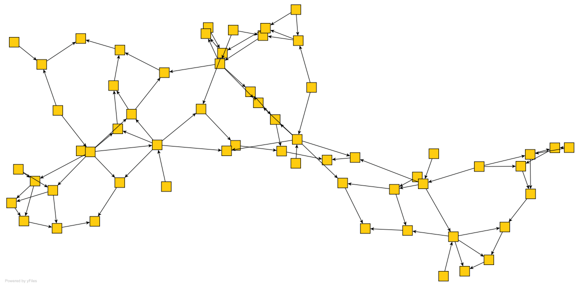


Figure A.31: Reduced Circuit Variant Graph after applying 20 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $C17 + C17 + C17$. (45 gates, 8 levels)

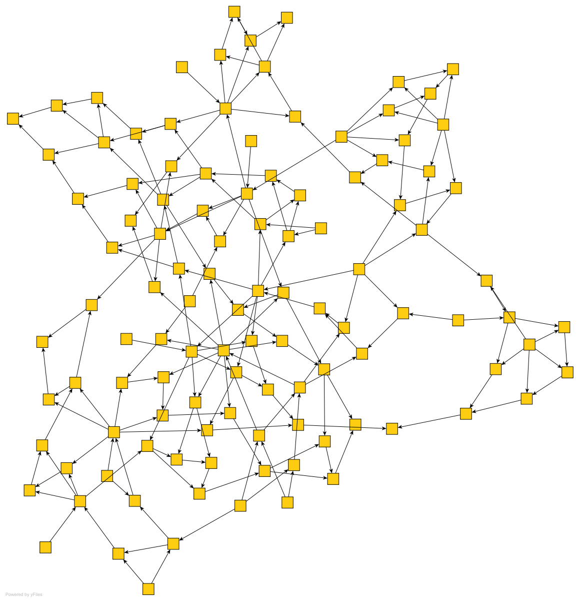


Figure A.32: Reduced Circuit Variant Graph after applying 50 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $C17 + C17 + C17$. (96 gates, 17 levels)

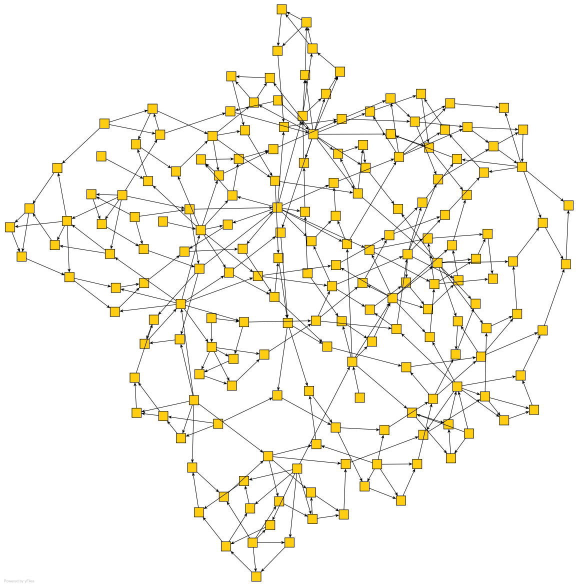


Figure A.33: Reduced Circuit Variant Graph after applying 100 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $C17 + C17 + C17$. (170 gates, 34 levels)

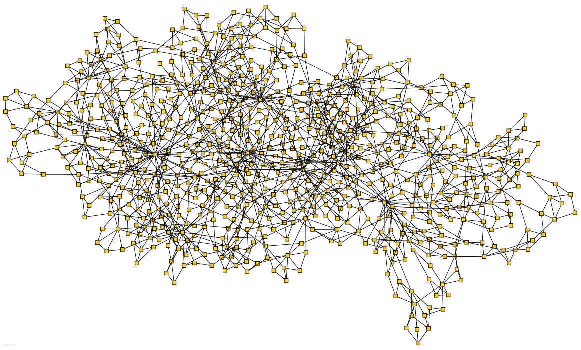


Figure A.34: Reduced Circuit Variant Graph after applying 500 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $C17 + C17 + C17$. (782 gates, 223 levels)

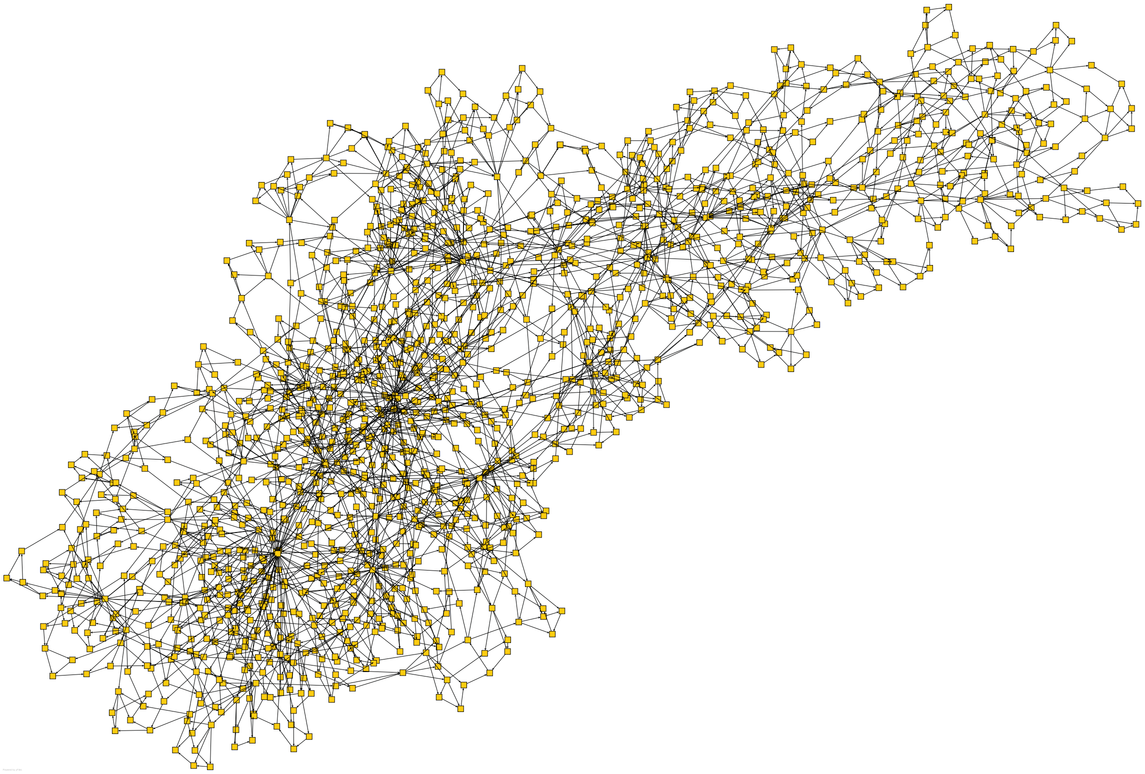


Figure A.35: Reduced Circuit Variant Graph after applying 1000 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $C17 + C17 + C17$. (1458 gates, 436 levels)

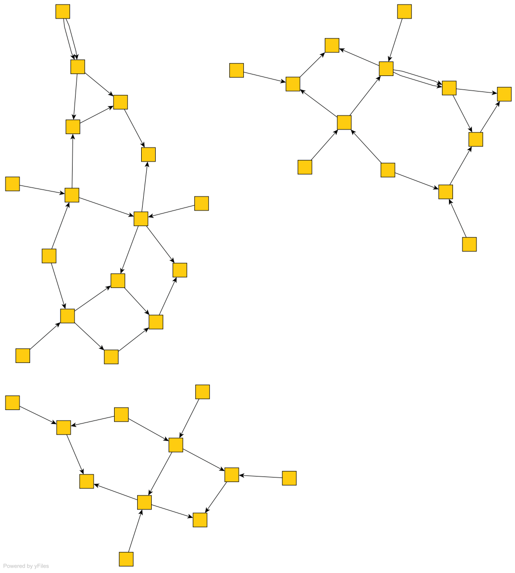


Figure A.36: Reduced Circuit Variant Graph after applying 20 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $C17+C17+C17$. (25 gates, 5 levels)

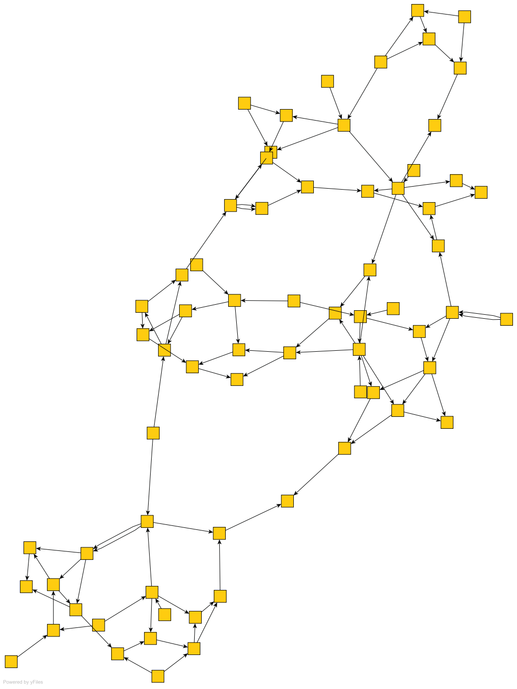


Figure A.37: Reduced Circuit Variant Graph after applying 50 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $C17+C17+C17$. (53 gates, 12 levels)

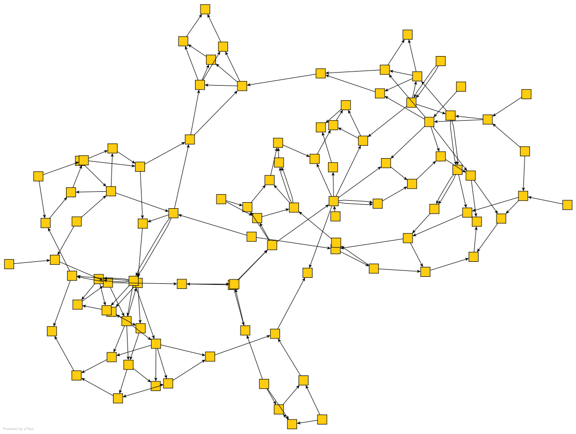


Figure A.38: Reduced Circuit Variant Graph after applying 100 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $C17 + C17 + C17$. (83 gates, 23 levels)

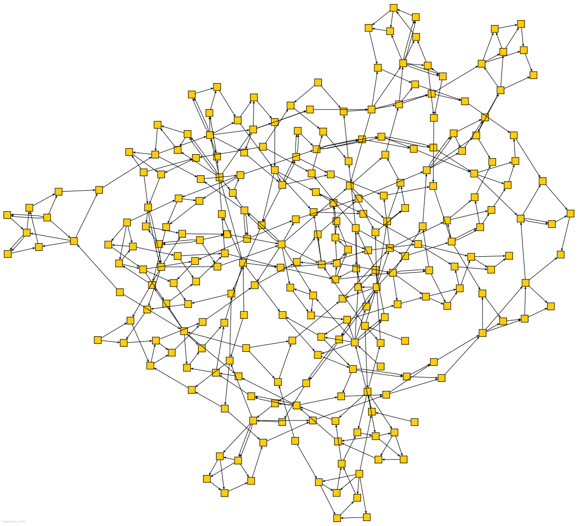


Figure A.39: Reduced Circuit Variant Graph after applying 500 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $C17 + C17 + C17$. (232 gates, 76 levels)

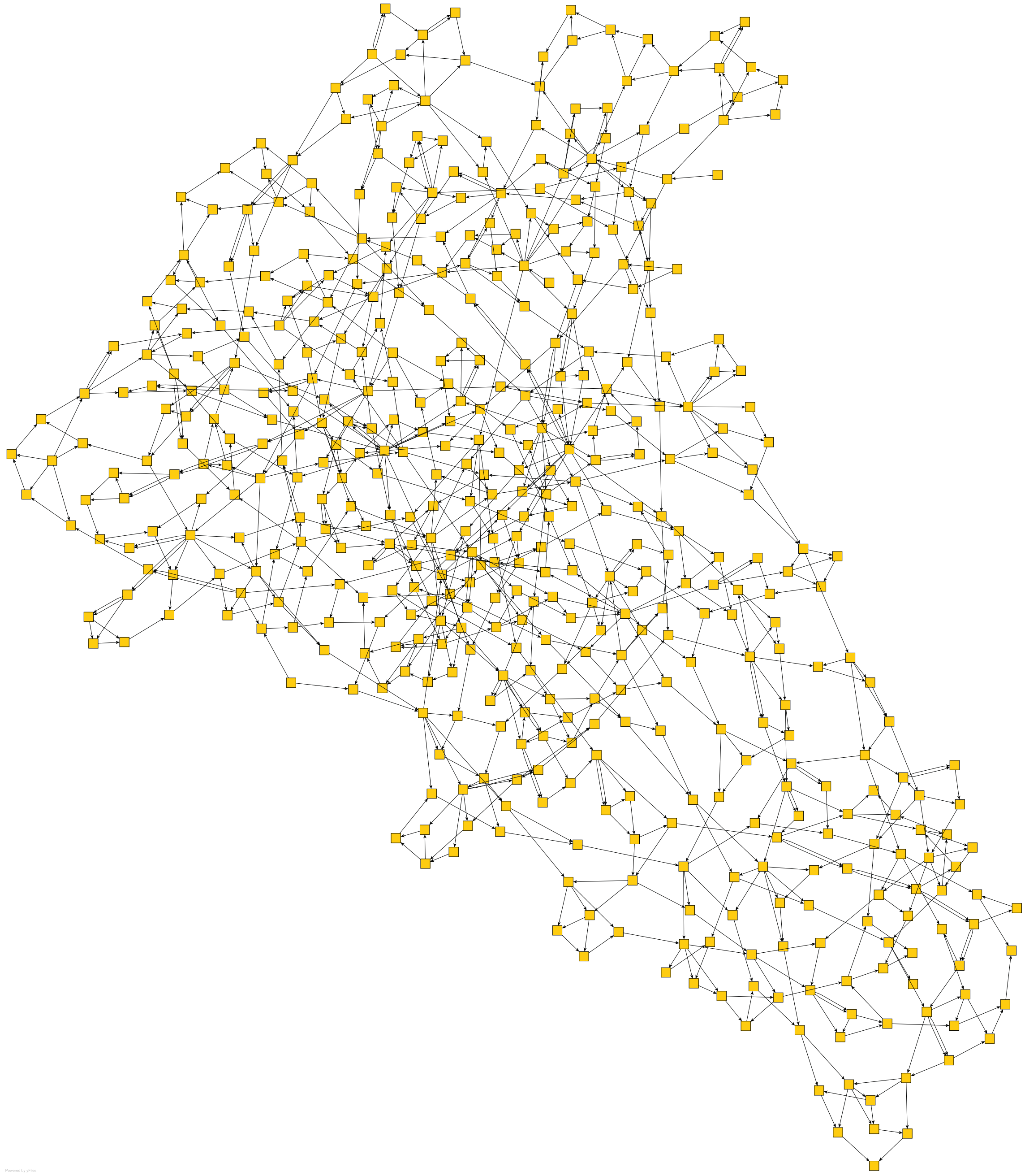


Figure A.40: Reduced Circuit Variant Graph after applying 1000 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $C17 + C17 + C17$. (480 gates, 130 levels)

Appendix B. Circuit Graphs for $R17' - C17 - R17''$

B.1 Obfuscated Circuit Variant Graphs

1. *Obfuscated* $R17' - C17 - R17''$ *(3 gates replacement, FFFTTT)*: Figure B.1, Figure B.2, $\cdots$, and Figure B.5 display graphs of obfuscated circuit variants that we use in this research. The RandomAlgorithm strategy is applied to the original circuits with 20, 50, 100, 500, and 1000 iterations. We also select `three` gates as a replacement size, and then choose FFFTTT as a library generation option.

2. *Obfuscated* $R17' - C17 - R17''$ *(3 gates replacement, TFTTTT)*: Figure B.6, Figure B.7, $\cdots$, and Figure B.10 display graphs of obfuscated circuit variants that we use in this research. The RandomAlgorithm strategy is applied to the original circuits with 20, 50, 100, 500, and 1000 iterations. We also select `three` gates as a replacement size, and then choose TFTTTT as a library generation option.

3. *Obfuscated* $R17' - C17 - R17''$ *(4 gates replacement, FFFTTT)*: Figure B.11, Figure B.12, $\cdots$, and Figure B.15 display graphs of obfuscated circuit variants that we use in this research. The RandomAlgorithm strategy is applied to the original circuits with 20, 50, 100, 500, and 1000 iterations. We also select `four` gates as a replacement size, and then choose FFFTTT as a library generation option.

4. *Obfuscated* $R17' - C17 - R17''$ *(4 gates replacement, TFTTTT)*: Figure B.16, Figure B.17, $\cdots$, and Figure B.20 display graphs of obfuscated circuit variants that we use in this research. The RandomAlgorithm strategy is applied to the original circuits with 20, 50, 100, 500, and 1000 iterations. We also select `four` gates as a replacement size, and then choose TFTTTT as a library generation option.

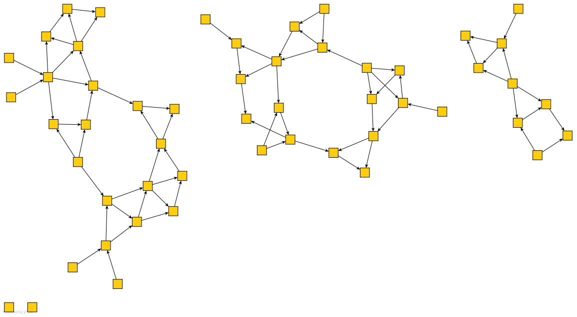


Figure B.1: Obfuscated Circuit Variant Graph after applying 20 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $R17' + C17 + R17''$. (37 gates, 9 levels)

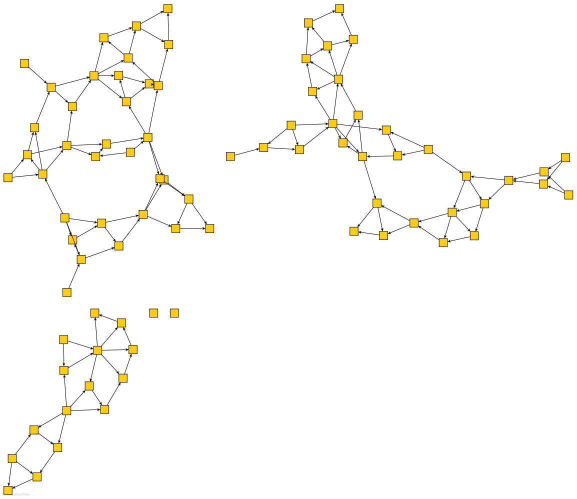


Figure B.2: Obfuscated Circuit Variant Graph after applying 50 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $R17' + C17 + R17''$. (68 gates, 15 levels)

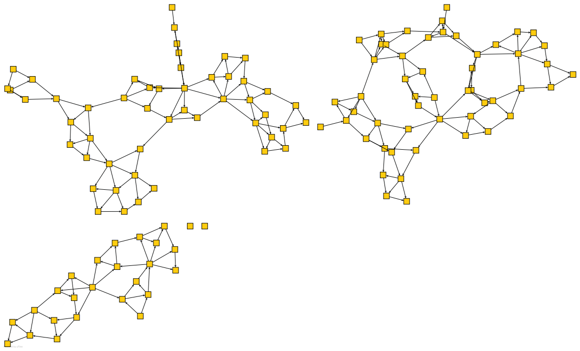


Figure B.3: Obfuscated Circuit Variant Graph after applying 100 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $R17'$+$C17$+$R17''$. (113 gates, 27 levels)

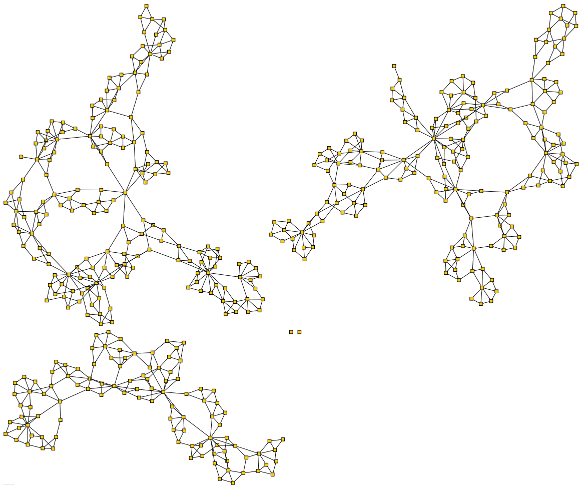


Figure B.4: Obfuscated Circuit Variant Graph after applying 500 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $R17'$+$C17$+$R17''$. (434 gates, 78 levels)

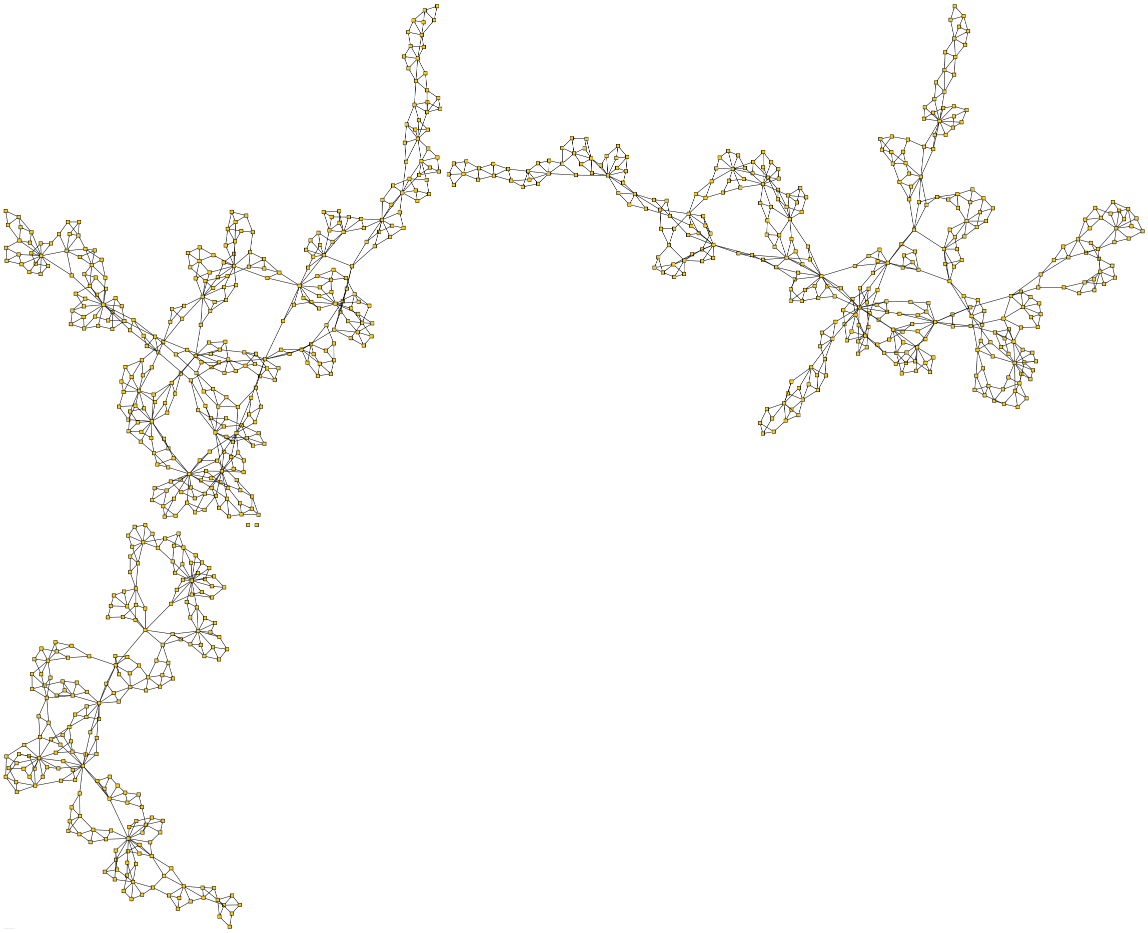


Figure B.5: Obfuscated Circuit Variant Graph after applying 1000 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $R17'$+$C17$+$R17''$. (880 gates, 166 levels)

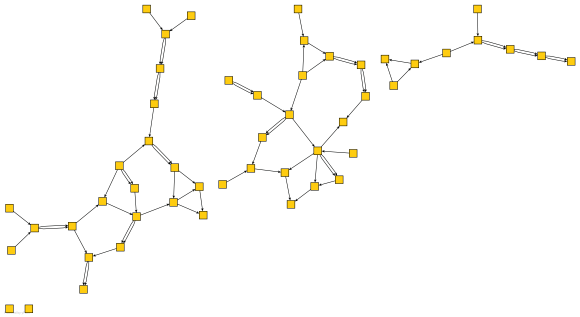


Figure B.6: Obfuscated Circuit Variant Graph after applying 20 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $R17'+C17+R17''$. (36 gates, 8 levels)

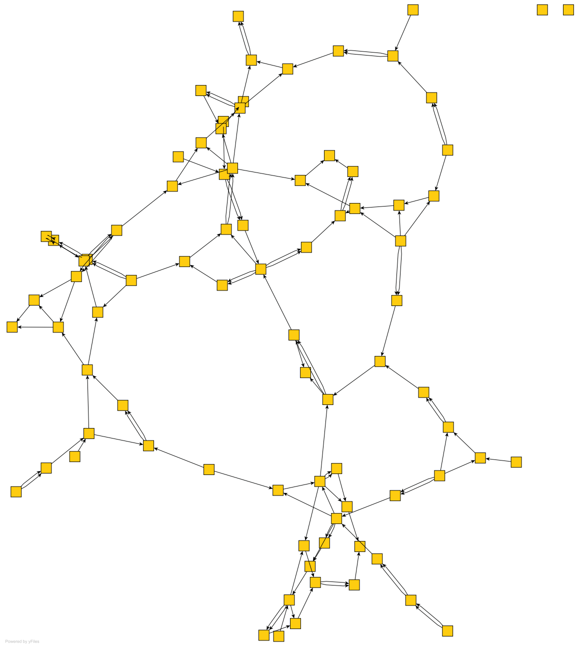


Figure B.7: Obfuscated Circuit Variant Graph after applying 50 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $R17'+C17+R17''$. (67 gates, 18 levels)

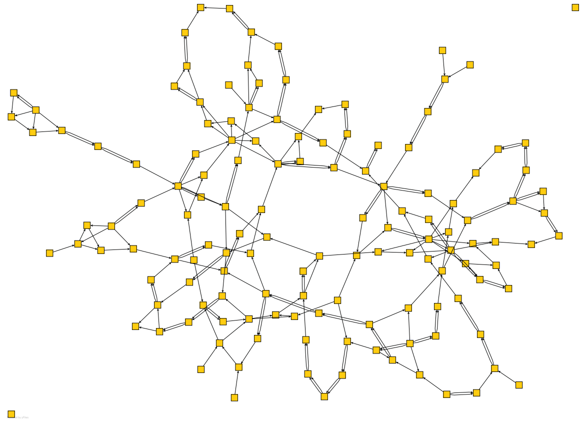


Figure B.8: Obfuscated Circuit Variant Graph after applying 100 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $R17' + C17 + R17''$. (123 gates, 43 levels)

Figure B.9: Obfuscated Circuit Variant Graph after applying 500 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $R17' + C17 + R17''$. (563 gates, 177 levels)

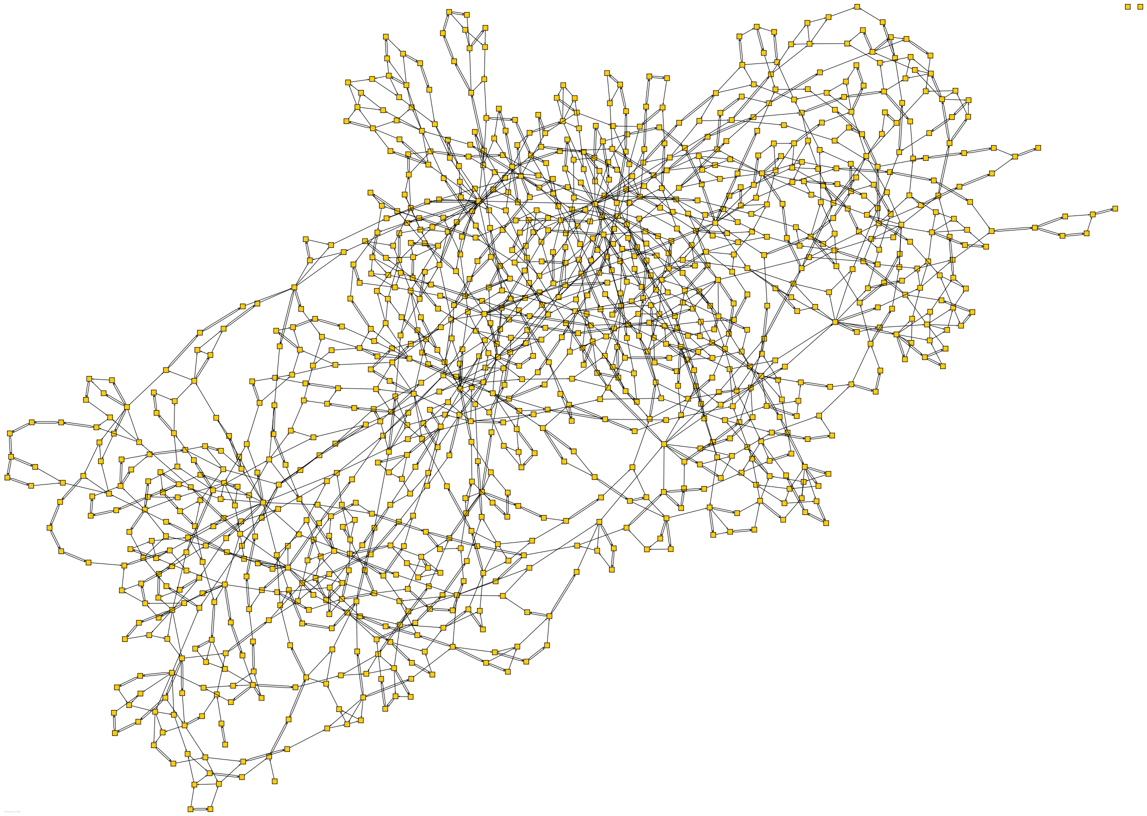


Figure B.10: Obfuscated Circuit Variant Graph after applying 1000 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $R17' + C17 + R17''$. (1116 gates, 364 levels)

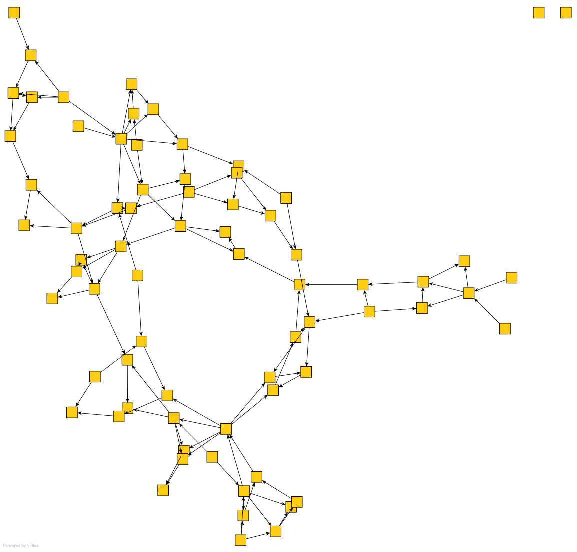


Figure B.11: Obfuscated Circuit Variant Graph after applying 20 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $R17'$+$C17$+$R17''$. (57 gates, 17 levels)

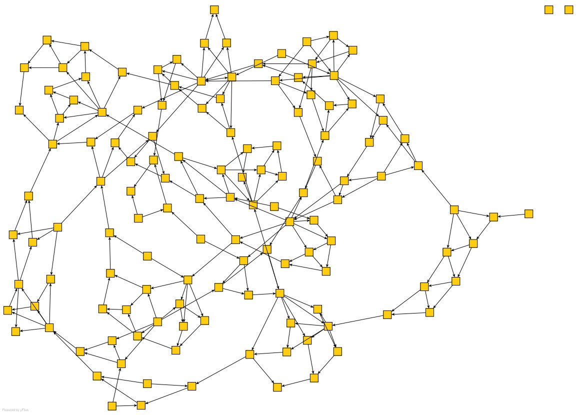


Figure B.12: Obfuscated Circuit Variant Graph after applying 50 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $R17'$+$C17$+$R17''$. (119 gates, 40 levels)

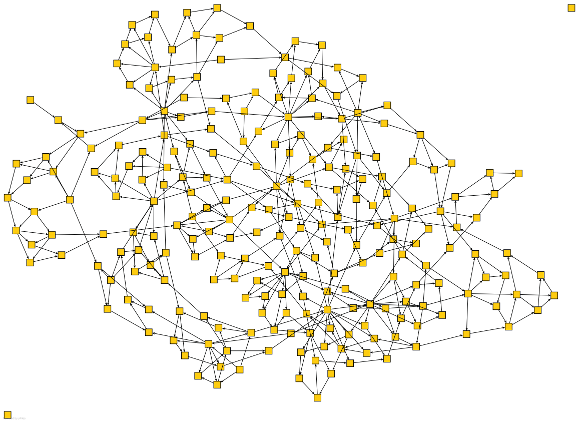


Figure B.13: Obfuscated Circuit Variant Graph after applying 100 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $R17'$+$C17$+$R17''$. (222 gates, 60 levels)

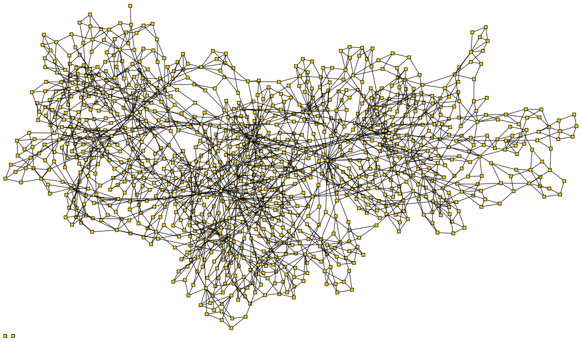


Figure B.14: Obfuscated Circuit Variant Graph after applying 500 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $R17'$+$C17$+$R17''$. (1076 gates, 322 levels)

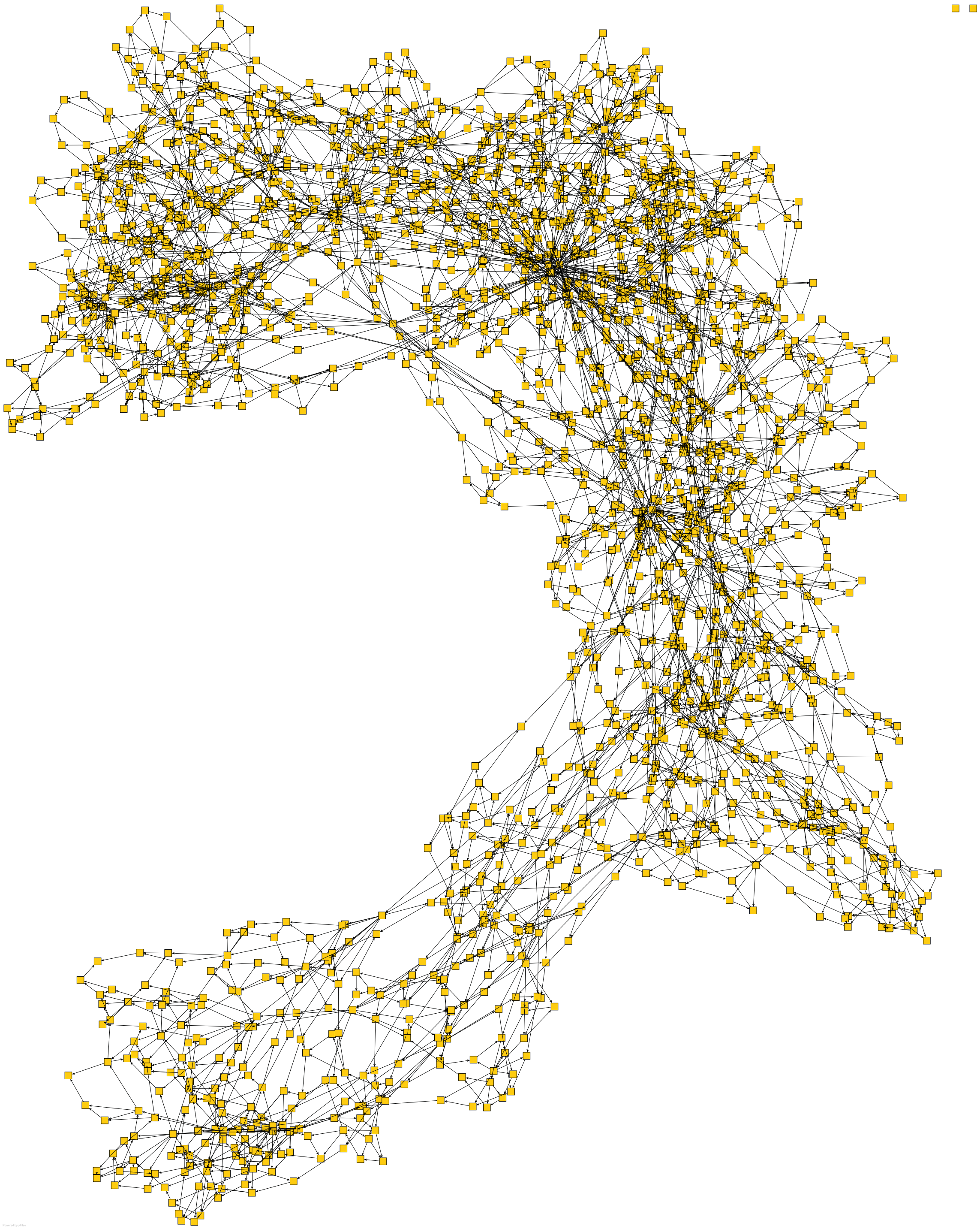


Figure B.15: Obfuscated Circuit Variant Graph after applying 1000 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $R17'$+$C17$+$R17''$. (2133 gates, 614 levels)

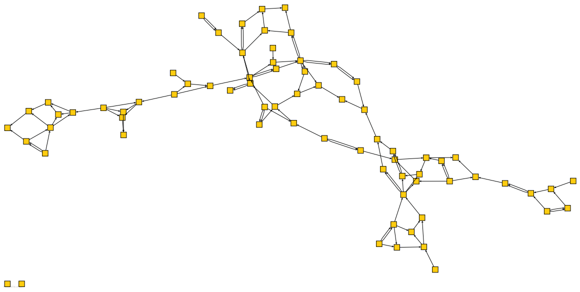


Figure B.16: Obfuscated Circuit Variant Graph after applying 20 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $R17' + C17 + R17''$. (58 gates, 15 levels)

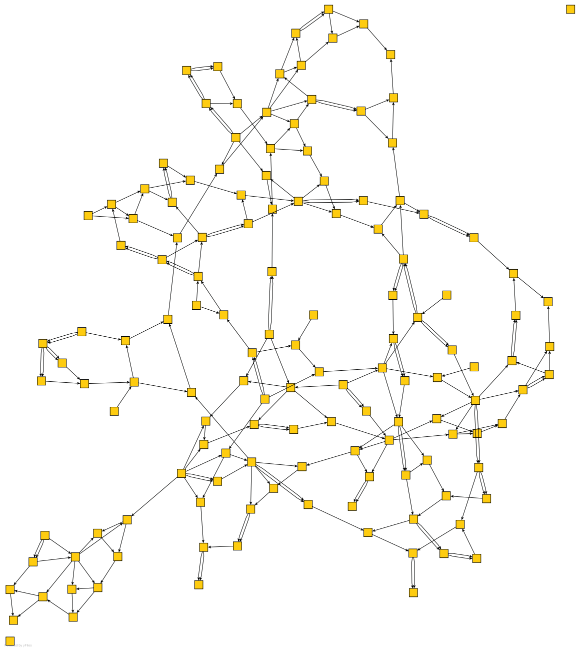


Figure B.17: Obfuscated Circuit Variant Graph after applying 50 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $R17' + C17 + R17''$. (123 gates, 26 levels)

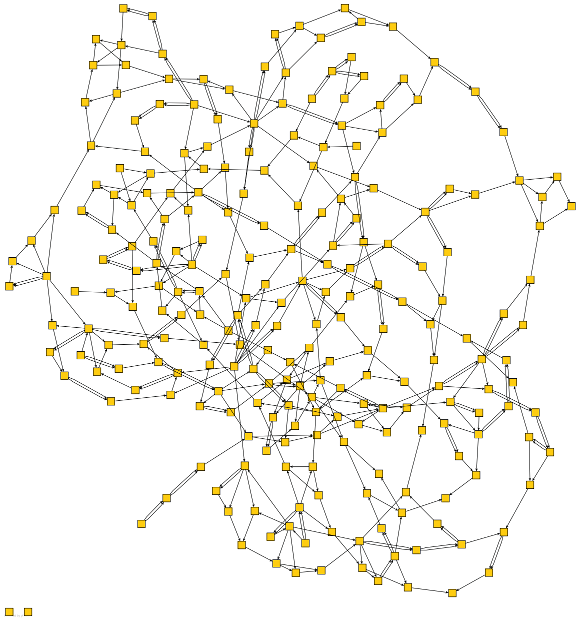


Figure B.18: Obfuscated Circuit Variant Graph after applying 100 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $R17' + C17 + R17''$. (226 gates, 56 levels)

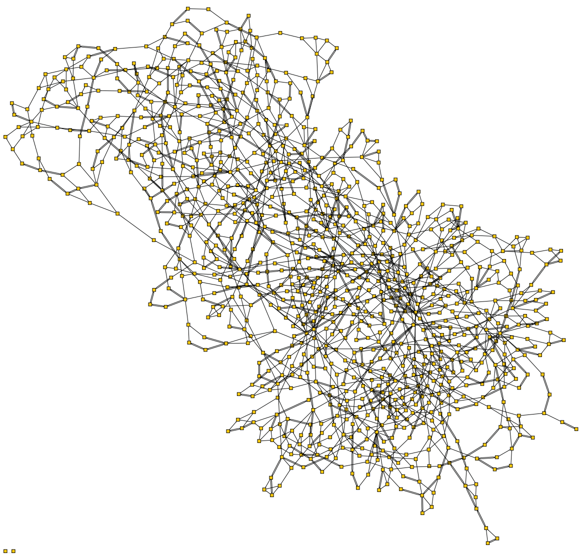


Figure B.19: Obfuscated Circuit Variant Graph after applying 500 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $R17' + C17 + R17''$. (1049 gates, 336 levels)

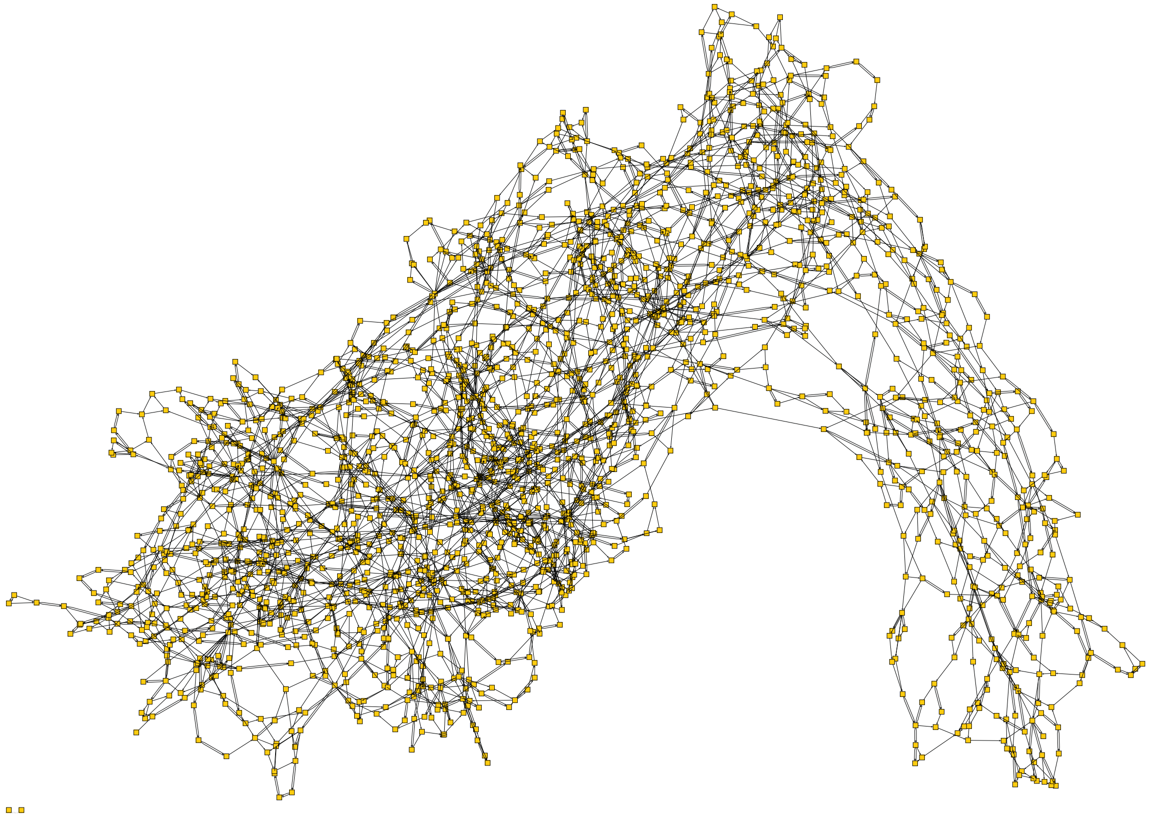


Figure B.20: Obfuscated Circuit Variant Graph after applying 1000 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $R17' + C17 + R17''$. (2094 gates, 607 levels)

B.2 Reduced Circuit Variant Graphs

1. *Reduced* $R17'-C17-R17''$ *(3 gates replacement, FFFTTT)*: Figure B.21, Figure B.22, $\cdots$, and Figure B.25 display graphs of reduced circuit variants that we produce in this research. The twelve circuit reduction algorithms are randomly applied to the obfuscated circuit variants with 10 reduction rounds.

2. *Reduced* $R17'-C17-R17''$ *(3 gates replacement, TFTTTT)*: Figure B.26, Figure B.27, $\cdots$, and Figure B.30 display graphs of reduced circuit variants that we produce in this research. The twelve circuit reduction algorithms are randomly applied to the obfuscated circuit variants with 10 reduction rounds.

3. *Reduced* $R17'-C17-R17''$ *(4 gates replacement, FFFTTT)*: Figure B.31, Figure B.32, $\cdots$, and Figure B.35 display graphs of reduced circuit variants that we produce in this research. The twelve circuit reduction algorithms are randomly applied to the obfuscated circuit variants with 10 reduction rounds.

4. *Reduced* $R17'-C17-R17''$ *(4 gates replacement, TFTTTT)*: Figure B.36, Figure B.37, $\cdots$, and Figure B.40 display graphs of reduced circuit variants that we produce in this research. The twelve circuit reduction algorithms are randomly applied to the obfuscated circuit variants with 10 reduction rounds.

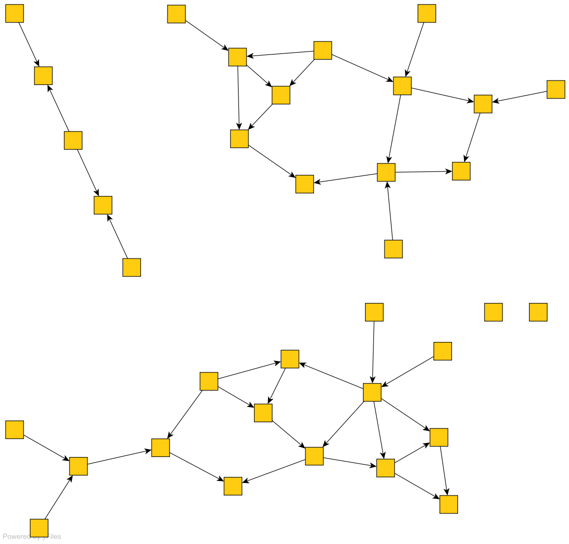


Figure B.21: Reduced Circuit Variant Graph after applying 20 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $R17' + C17 + R17''$. (20 gates, 7 levels)

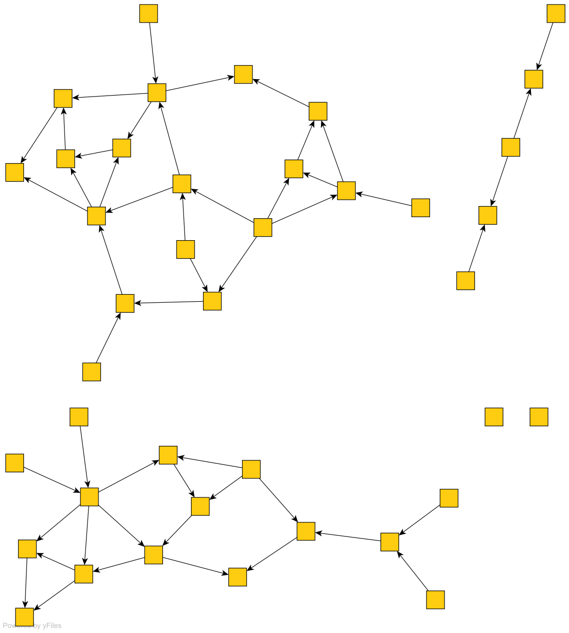


Figure B.22: Reduced Circuit Variant Graph after applying 50 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $R17' + C17 + R17''$. (25 gates, 7 levels)

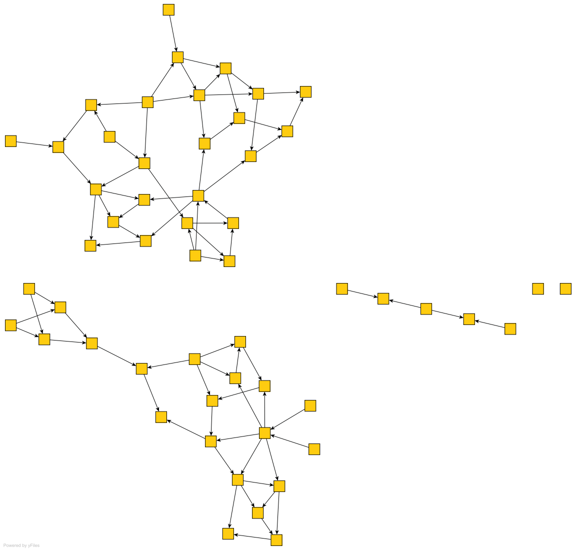


Figure B.23: Reduced Circuit Variant Graph after applying 100 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $R17' + C17 + R17''$. (39 gates, 11 levels)

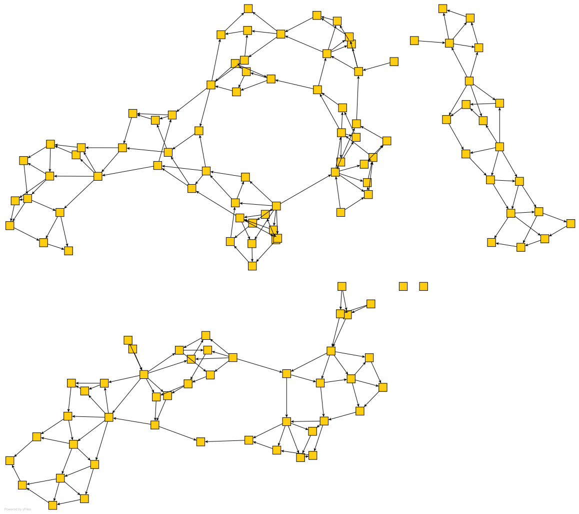


Figure B.24: Reduced Circuit Variant Graph after applying 500 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $R17' + C17 + R17''$. (115 gates, 29 levels)

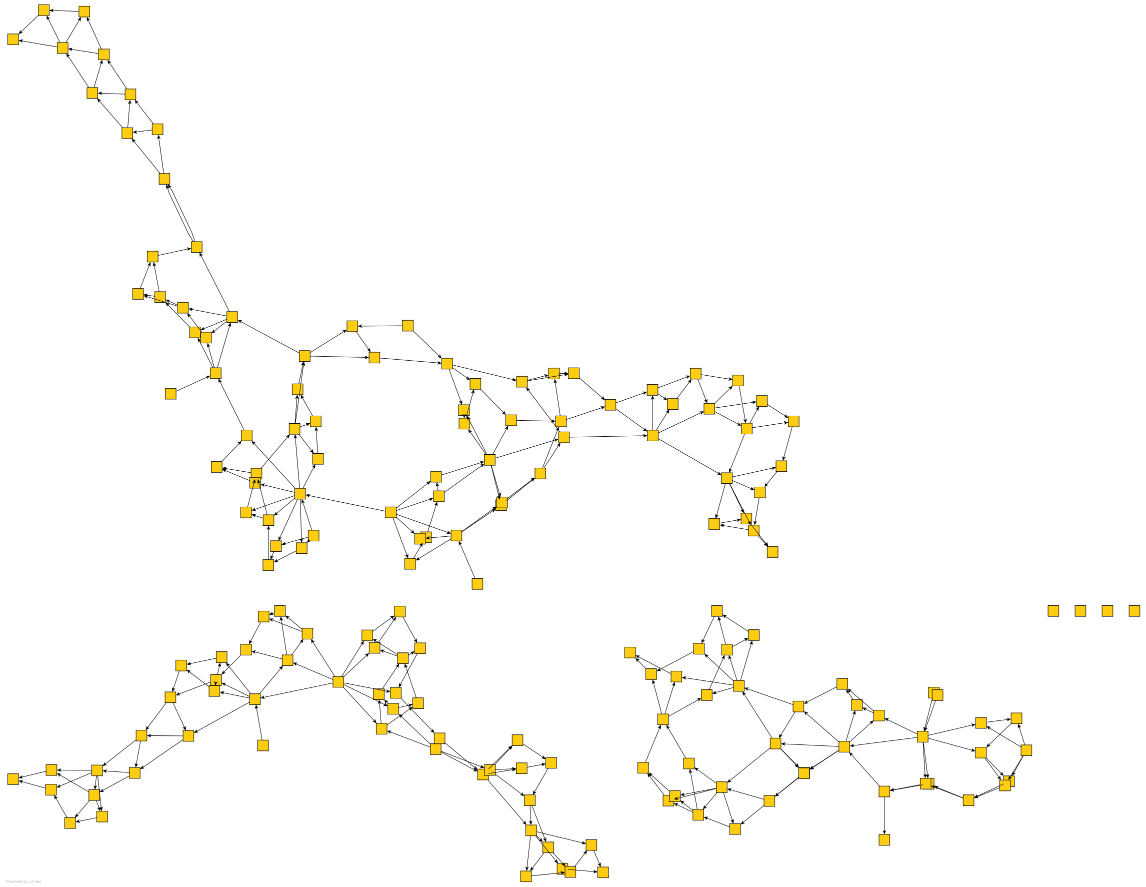


Figure B.25: Reduced Circuit Variant Graph after applying 1000 iterations, 3 replacement sizes, and FFFTTT library generation option to circuit $R17' + C17 + R17''$. (155 gates, 37 levels)

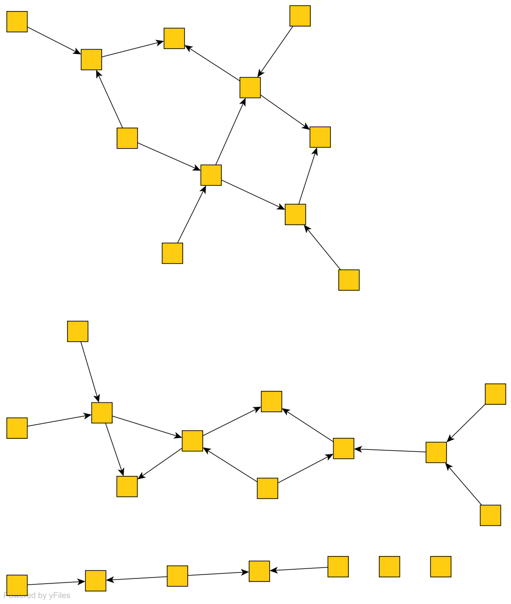


Figure B.26: Reduced Circuit Variant Graph after applying 20 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $R17' + C17 + R17''$. (14 gates, 3 levels)

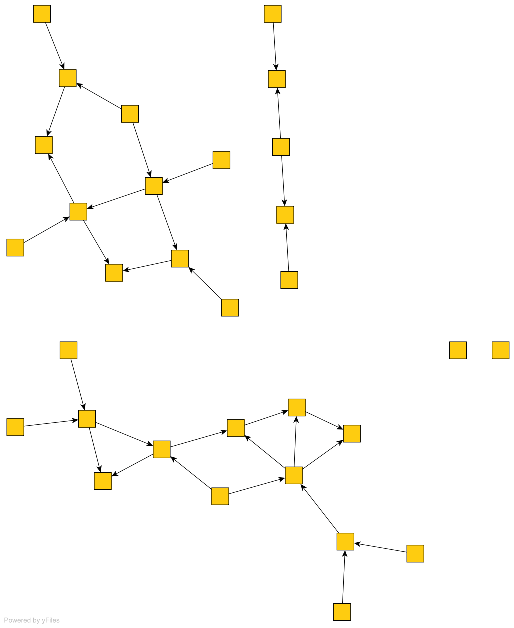


Figure B.27: Reduced Circuit Variant Graph after applying 50 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $R17' + C17 + R17''$. (16 gates, 5 levels)

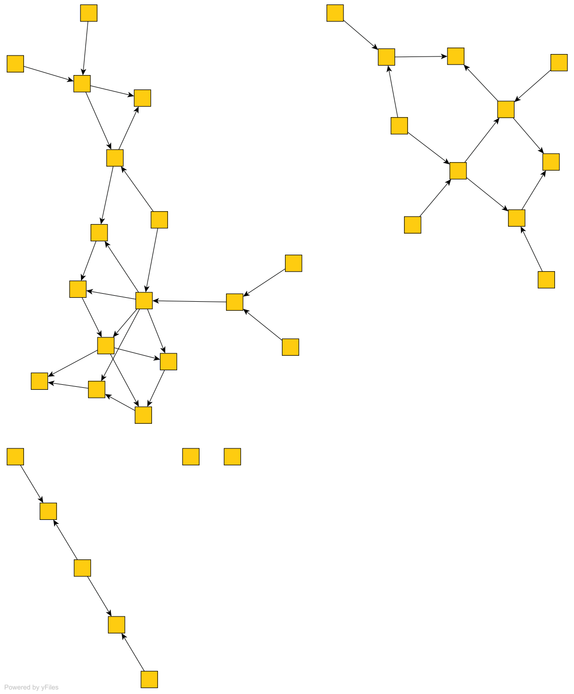


Figure B.28: Reduced Circuit Variant Graph after applying 100 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $R17'+C17+R17''$. (20 gates, 9 levels)

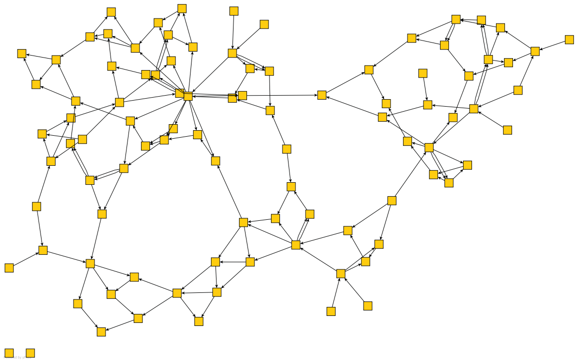


Figure B.29: Reduced Circuit Variant Graph after applying 500 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $R17'+C17+R17''$. (80 gates, 23 levels)

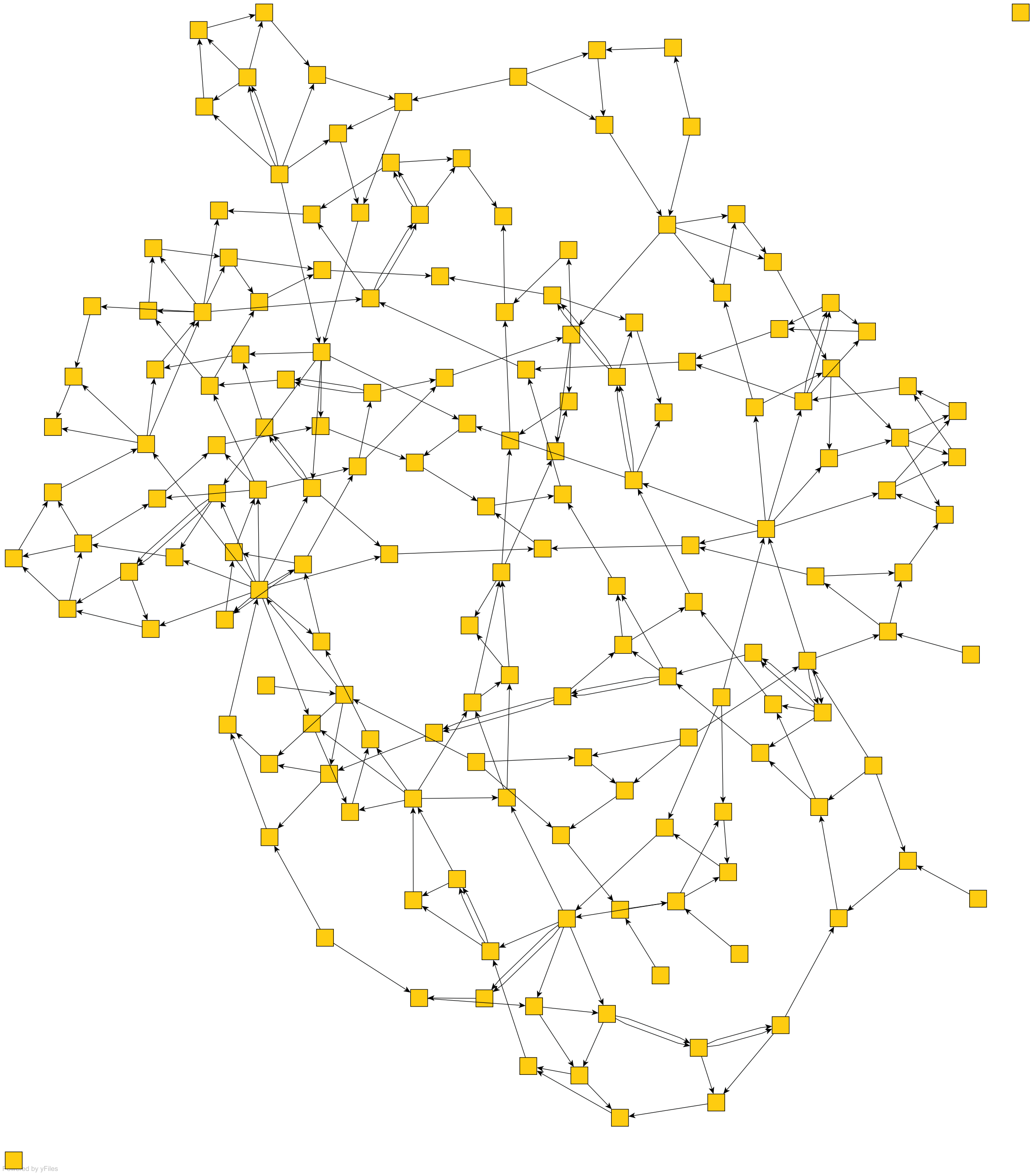


Figure B.30: Reduced Circuit Variant Graph after applying 1000 iterations, 3 replacement sizes, and TFTTTT library generation option to circuit $R17'+C17+R17''$. (147 gates, 45 levels)

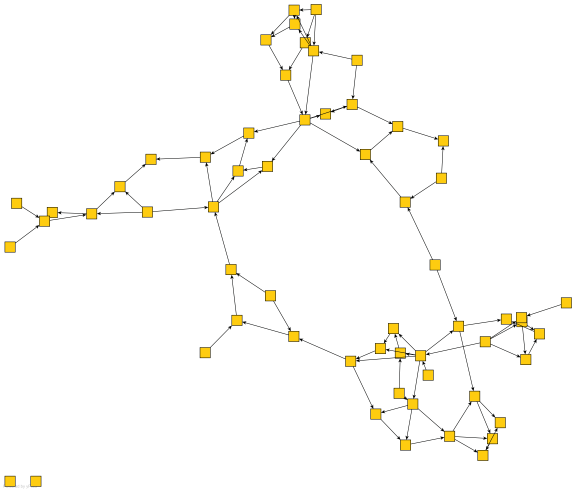


Figure B.31: Reduced Circuit Variant Graph after applying 20 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $R17' + C17 + R17''$. (45 gates, 14 levels)

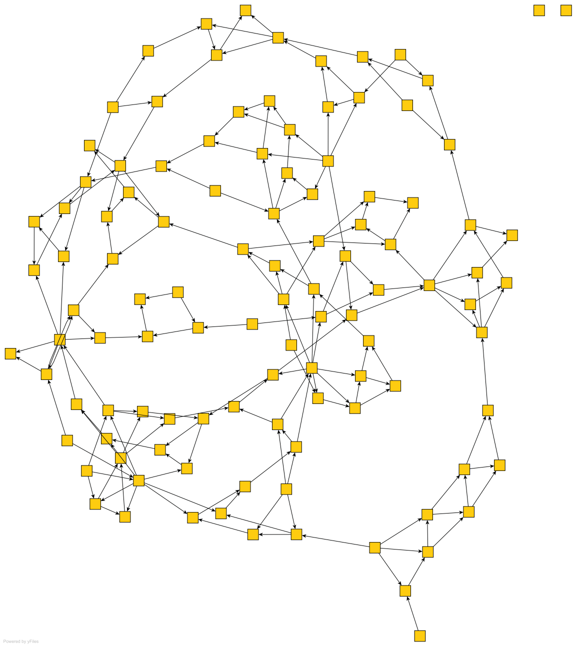


Figure B.32: Reduced Circuit Variant Graph after applying 50 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $R17' + C17 + R17''$. (94 gates, 32 levels)

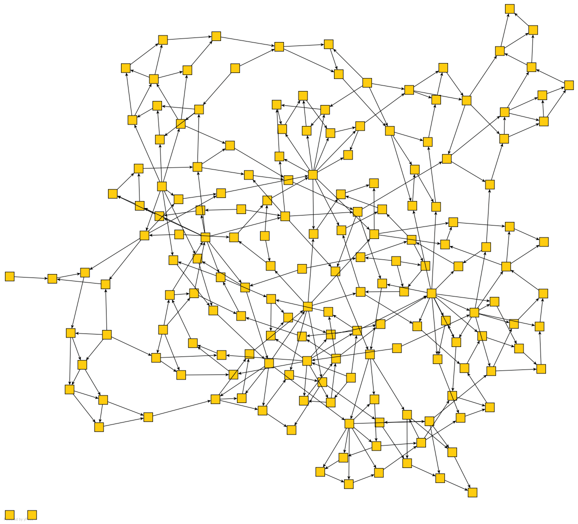


Figure B.33: Reduced Circuit Variant Graph after applying 100 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $R17' + C17 + R17''$. (158 gates, 43 levels)

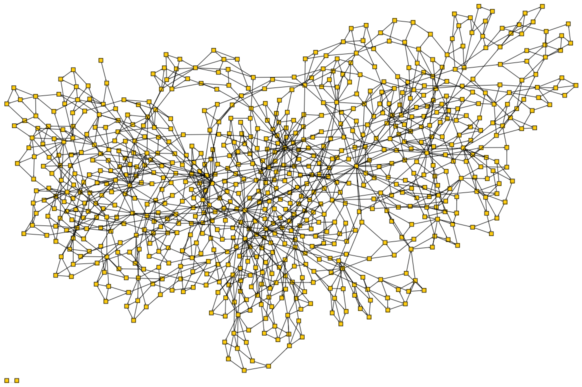


Figure B.34: Reduced Circuit Variant Graph after applying 500 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $R17' + C17 + R17''$. (761 gates, 205 levels)

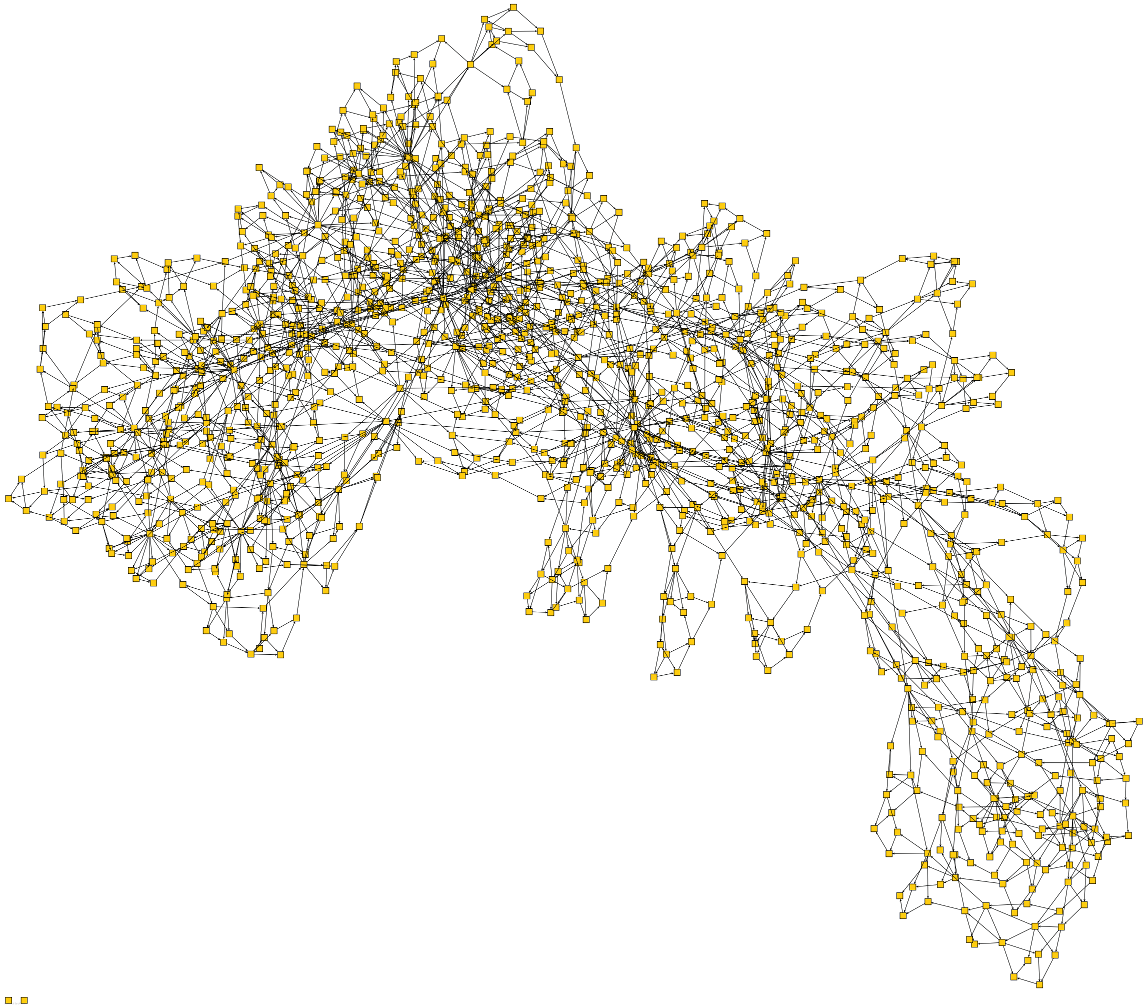


Figure B.35: Reduced Circuit Variant Graph after applying 1000 iterations, 4 replacement sizes, and FFFTTT library generation option to circuit $R17' + C17 + R17''$. (1471 gates, 429 levels)

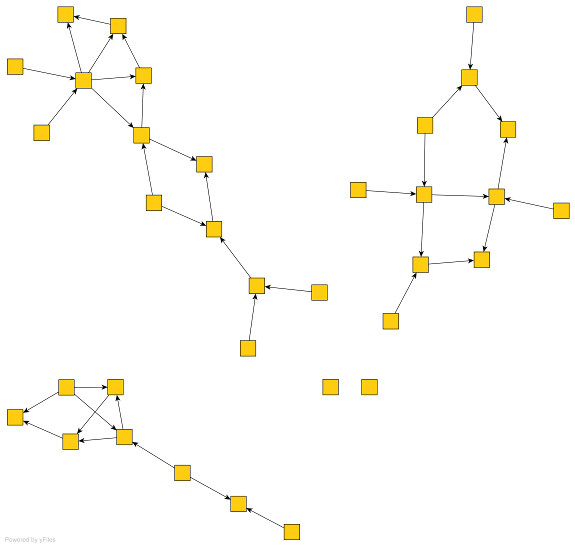


Figure B.36: Reduced Circuit Variant Graph after applying 20 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $R17' + C17 + R17''$. (19 gates, 5 levels)

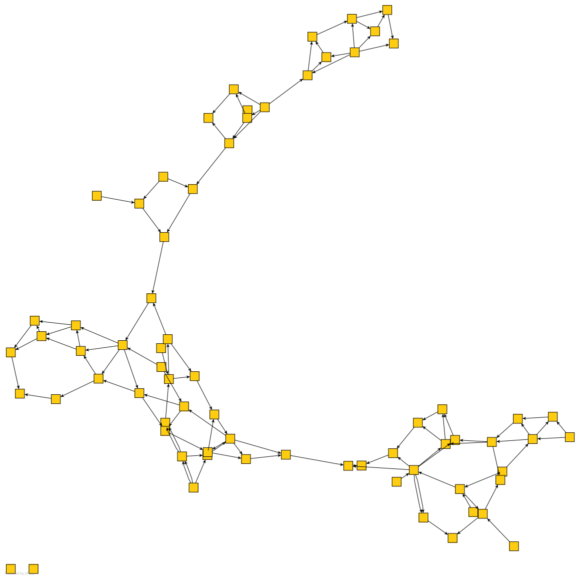


Figure B.37: Reduced Circuit Variant Graph after applying 50 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $R17' + C17 + R17''$. (55 gates, 16 levels)

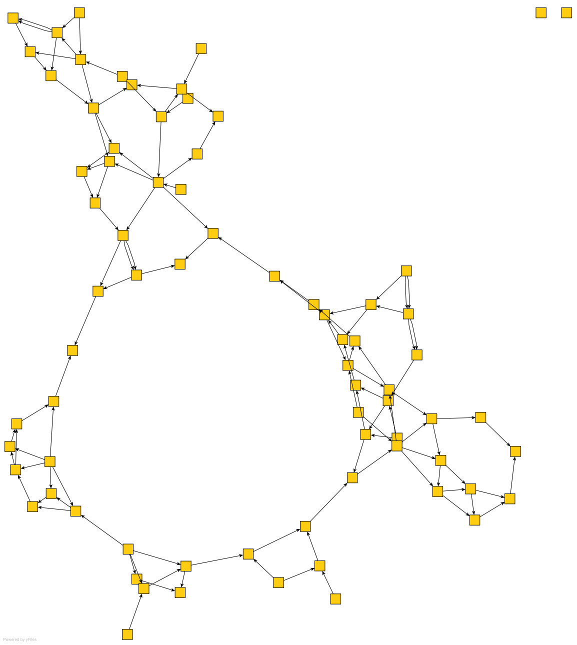


Figure B.38: Reduced Circuit Variant Graph after applying 100 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $R17'+C17+R17''$. (59 gates, 16 levels)

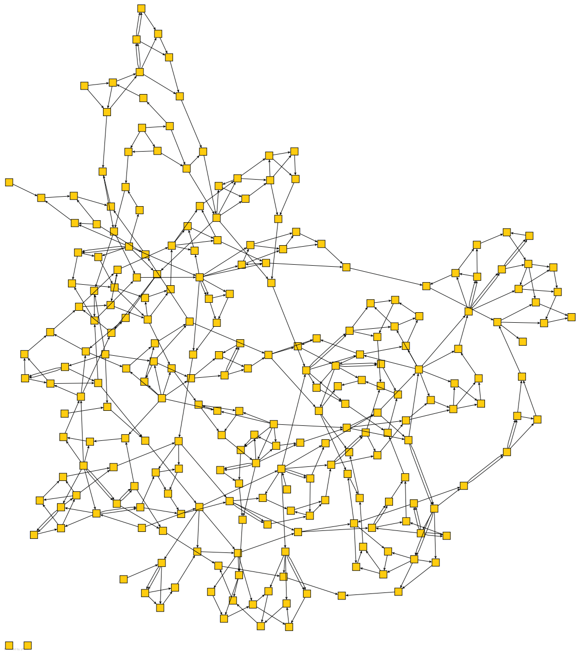


Figure B.39: Reduced Circuit Variant Graph after applying 500 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $R17'+C17+R17''$. (227 gates, 62 levels)

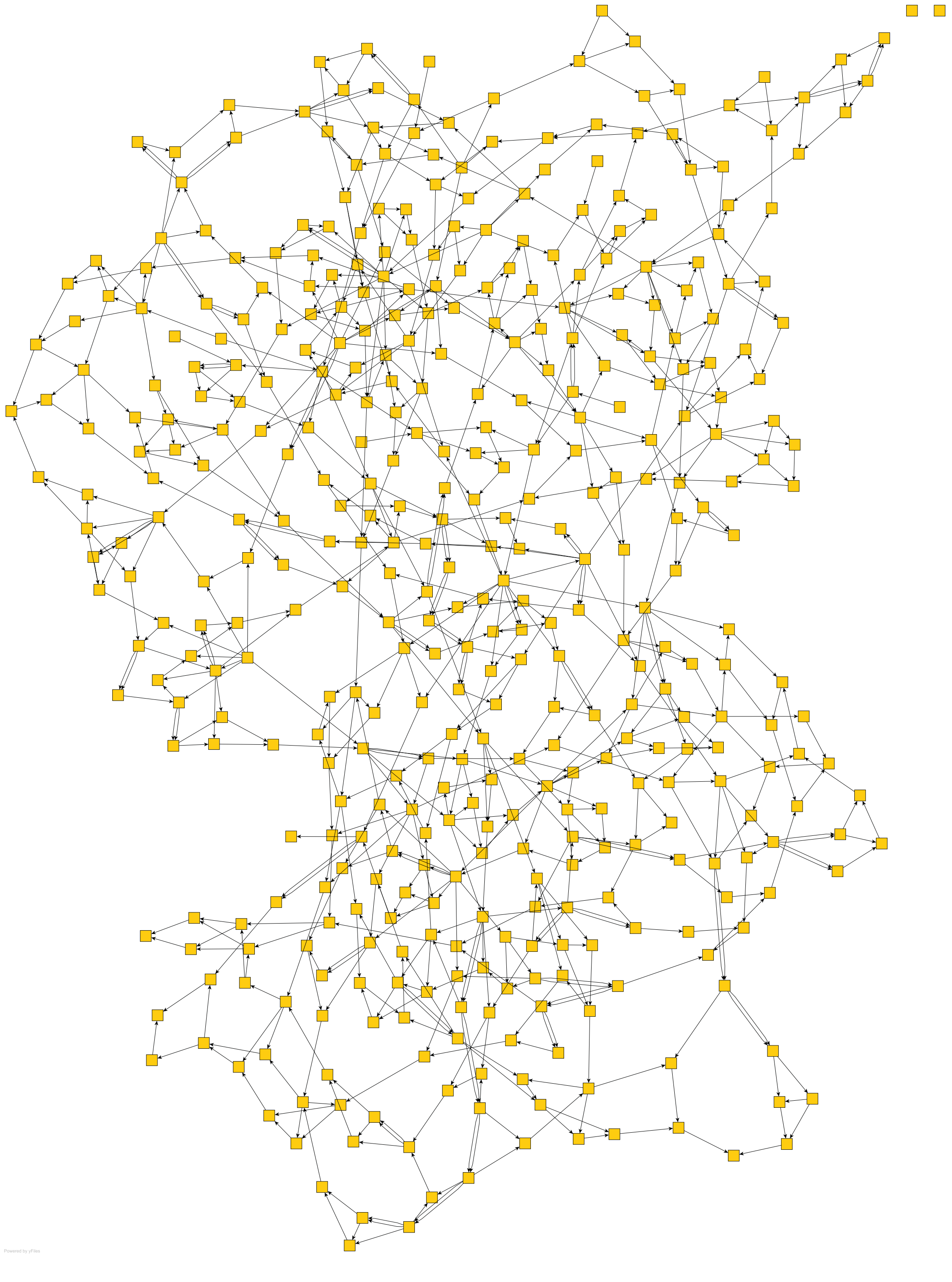


Figure B.40: Reduced Circuit Variant Graph after applying 1000 iterations, 4 replacement sizes, and TFTTTT library generation option to circuit $R17'+C17+R17''$. (419 gates, 127 levels)

Bibliography

1. Aronson, R.B. “Forward Thinkers Take to Reverse Engineering”. *Manufacturing Engineering*, 117:34–44, November 1996.

2. Associates, DRM. “What Is Reverse Engineering?” Internet: `http://www.npd-solutions.com/reoverview.html`.

3. Barak, Boaz, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. “On the (Im)possibility of obfuscating programs”. *Electronic Colloquium on Computational Complextiy*, 8(57):1–41, 2001.

4. “Benchmark circuits”. Internet: `http://www.fm.vslib.cz/ kes/asic/iscas/`, Jan 2007.

5. Chikofsky, Elliot J. and James H. Cross II. “Reverse Engineering and Design Recovery: A Taxonomy”. *Software, IEEE*, 7:13–17, January 1990.

6. Chow, Stanley, Yuan Gu, Harold Johnson, and Vladimir A. Zakharov. “An Approach to the Obfuscation of Control-Flow of Sequential Computer Programs”. *ISC ’01: Proceedings of the 4th International Conference on Information Security*, 144–155. Springer-Verlag, London, UK, 2001. ISBN 3-540-42662-0.

7. Collberg, Christian, Clark Thomborson, and Douglas Low. “Manufacturing cheap, resilient, and stealthy opaque constructs”. *POPL ’98: Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 184–196. ACM, New York, NY, USA, 1998. ISBN 0-89791-979-3.

8. Collberg, Christian S. and Clark Thomborson. “Watermarking, tamper-proofing, and obfuscation: tools for software protection”. *IEEE Trans. Softw. Eng.*, 28(8):735–746, 2002. ISSN 0098-5589.

9. Goldwasser, Shafi and Guy N. Rothblum. “On Best-Possible Obfuscation”. *4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, 194–213. Springer, 21-24 February 2007. ISBN 3-540-70935-5.

10. Hansen, Mark C., Hakan Yalcin, and John P. Hayes. “Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering”. *IEEE Des. Test*, 16(3):72–80, 1999. ISSN 0740-7475.

11. Huth, Michael and Mark Ryan. *Logic in computer science: Modelling and Reasoning about Systems*. Cambridge University Press, 2004.

12. Jeffrey T. McDonald, Yong C. Kim, Kenneth Norman. “Introducing CORGI: A Framework for Whitebox Circuit Obfuscation”, November 2008.

13. Linn, C. and S. Debray. “Obfuscation of Executable Code to Improve Resistance to Static Disassembly”, 2003. URL `citeseer.ist.psu.edu/linn03obfuscation.html`.

14. McDonald, J. Todd, Yong C. Kim, and Alec Yasinsac. “Software Issues in Digital Forensics”. *ACM Operating Systems Review*, 42(3), April 2008.

15. McDonald, J. Todd and Alec Yasinsac. “Program intent protection using circuit encryption”. *Proceedings of the 8th International Symposium on System and Information Security*. IEEE Computer Society, 8-10 Nov 2006.

16. McDonald, Jeffrey T. *Enhanced Security for Mobile Agent Systems*. Ph.D. thesis, Florida State University, 2006.

17. Mish, Frederick C. (editor). *Merriam-Webster’s collegiate dictionary*. Merriam-Webster, Incorporated, Springfield, MA, 10 edition, 2001. ISBN 0-87779-710-2.

18. Norman, Ken. *Architecture for White-box Obfuscation Using Randomized Subcircuit Selection And Replacement*. Master’s thesis, Air Force Institute of Technology, 2008.

19. “Wiktionary”. Internet: `http://en.wiktionary.org/`, Oct 2007.

20. Williams, Jason A. *Characterizing Component Hiding Using Anestral Entropy*. Master’s thesis, Air Force Institute of Technology, 2009.

21. Yasinsac, Alec and J. Todd McDonald. “Tamper Resistant Software Through Intent Protection”. *The International Journal of Network Security*, 7(3):370–382.

Vita

Captain Hanseok Kim graduated from Pucheon High School in Pucheon-si, Gyeonggi, Korea. He entered undergraduate studies at Soongsil University in Dongjak-gu, Seoul, Korea where he graduated with a Bachelor degree in Computer Engineering in 2003. He was commissioned through Officer Training School in Yeongcheon-si, Gyeongbuk, Korea in 2003.

Captain Kim was first assigned to the Army Signal School in Yuseong-gu, Daejeon, Korea for the Officer Basic Course Training. Before leaving the Army Signal School, he was honored for the most exceptional academic achievement by the Korean Army Chief of Staff. Capt Kim was next assigned to the 5th Logistic Support Command in Suseong-gu, Daegu, Korea in September 2003 as an officer in charge of program development. In January 2005, he was assigned to the ROKA Computation Center in Gyeryong-si, Chungnam, Korea where he served as a quality control officer. He was next assigned to the ROKA Personnel Command in Gyeryong-si, Chungnam, Korea in September 2006 as an education preparatory officer. Next, Capt Kim entered the Graduate School of Engineering and Management, Air Force Institute of Technology in August 2007. Upon graduation, he will be assigned to the Army Signal School in Yuseong-gu, Daejeon, Korea for the Officer Advanced Course Training.

Permanent address: 5-202, Buseong Jutaek
287-11, Wonjong 1(il)-dong, Ojeong-gu
Bucheon-si, Gyeonggi-do, South Korea, 421-820

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704–0188*

1. REPORT DATE *(DD–MM–YYYY)* 26-03-2009	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Sep 2007–Mar 2009

4. TITLE AND SUBTITLE Removing Redundant Logic Pathways in Polymorphic Circuits	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Kim, Hanseok, Capt, ROKA	**5d. PROJECT NUMBER** 08-183
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCS/ENG/09-03
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Office of Scientific Research Attn: Dr. Robert L. Herklotz Suite 325, Room 3112 875 N. Randolph Street Arlington, VA 22203-1768 703–696–6565 (DSN: 426) robert.herklotz@afosr.af.mil	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFOSR/NL
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Evaluating the quality of software and circuit obfuscators is a research goal of great interest. However, there exists little research about evaluation of obfuscation effectiveness through analyzing and investigating redundancies found in the obfuscated variants.

In this research, we consider programs represented as structural combinational circuits and then analyze obfuscated variants of those circuits through a tool that produces functionally equivalent variants based on subcircuit selection and replacement. We then consider how Boolean logic and reduction affects the size and levelization of circuit variants, giving us a concrete metric by which to consider obfuscation effectiveness. To accomplish these goals, we create an experimental environment based on a set of predefined circuits, a set of predefined algorithms which produce variants of those circuits, and a collection of logic reduction techniques and tools. We build logic reduction techniques using predefined patterns and predefined functions expressed as truth tables.

As a contribution, we characterize and evaluate the effectiveness of obfuscating algorithms based on these reduction techniques. We show, for the circuits we observe, optimization on size is affected by ordering of the specific reduction patterns and functions. We also show, for the circuits we observe, reduction is affected by the specific obfuscating algorithm used to produce the variant. Based on these results, we provide a promising measurement of interest to compare both circuit variants and obfuscating algorithms.

15. SUBJECT TERMS

software tools, software engineering, computer programs, cryptography, obscuration, software obfuscation, randomization, pseudo random sequences, random functions, circuit reduction, circuit optimization, circuit minimization

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Lt Col J. Todd McDonald
U	U	U	UU	129	**19b. TELEPHONE NUMBER** *(include area code)* 937–255–3636 x4639, jmcdonal@afit.edu