CELLULAR AUTOMATA:

OPTIMAL ANALYSIS, CODING, AND TESTING FOR ENCRYPTION

By

Stephen J. Faulkenberry

A Thesis Submitted in Partial Fulfillment
Of the Requirements for the Degree of
Master of Science in Engineering Technology

Middle Tennessee State University
May 2016

We all want to help one another. Human beings are like that. We want to live by each other's happiness - not by each other's misery.

We don't want to hate and despise one another. In this world there is room for everyone. And the good earth is rich and can provide for everyone.

The way of life can be free and beautiful, but we have lost the way.

Greed has poisoned men's souls, has barricaded the world with hate, has goose-stepped us into misery and bloodshed.

We have developed speed, but we have shut ourselves in. Machinery that gives abundance has left us in want.

Our knowledge has made us cynical. Our cleverness, hard and unkind. We think too much and feel too little.

More than machinery we need humanity. More than cleverness we need kindness and gentleness.

Without these qualities, life will be violent and all will be lost.

-- Charlie Chaplin,
*from **The Great Dictator (1940)***

ABSTRACT

Cellular automata are a set of discrete structures generated and manipulated by predetermined rules, in which each state (or evolution) is influenced by the previous. Utilizing the simplicity of this fundamental structure, a number of configurations have been organized and derived from elementary (single dimensional) cellular automata. By harvesting the evolution of these structures as output, they lend greatly to random number generation and by extension, encryption. Analyzing, testing, and programming these methods has led to observations on optimal approaches to each. Utilizing the Diehard testing suite and the National Institute of Standards and Technology (NIST) Statistical Testing Suite (STS), configurations can be judged against each other as well as external systems. Optimal methods for generating configurations, visual observation and data analysis are compiled in a workbook program. A complete analysis for the state diagrams of k [1, 27] in the 3-bit rule space is included and a Cellular Automata Standard of Encryption (CASE) is suggested for real world use.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

AES =	Advanced Encryption Standard
c =	CA cell
CA =	Cellular Automata
CASE =	Cellular Automata Standard of Encryption
ECA =	Elementary Cellular Automata
GoE =	Garden of Eden state
k =	Length of CA seed
LUT =	Lookup Table
Ø =	Rule symbol
PRNG =	Pseudo Random Number Generator
RNG =	Random Number Generator
t =	CA time step

CHAPTER ONE: CELLULAR AUTOMATA

SECTION 1.1 – Elementary Cellular Automata

All structures operate on rules, whether it is the natural world relying on the laws of physics or an artificial environment relying on compiled logic, which itself is arguably an extension of physics. An elementary cellular automata (ECA), shown in Figure 1, consists of a single dimension of cells, each set in an active (binary 1, logic high) or inactive (binary 0, logic low) state. Each ECA begins with a seed, a string of active or inactive cells, serving as the initial state before the start of automation, in which a rule produces the next state of a cell depending on its neighborhood and fundamental look up table.

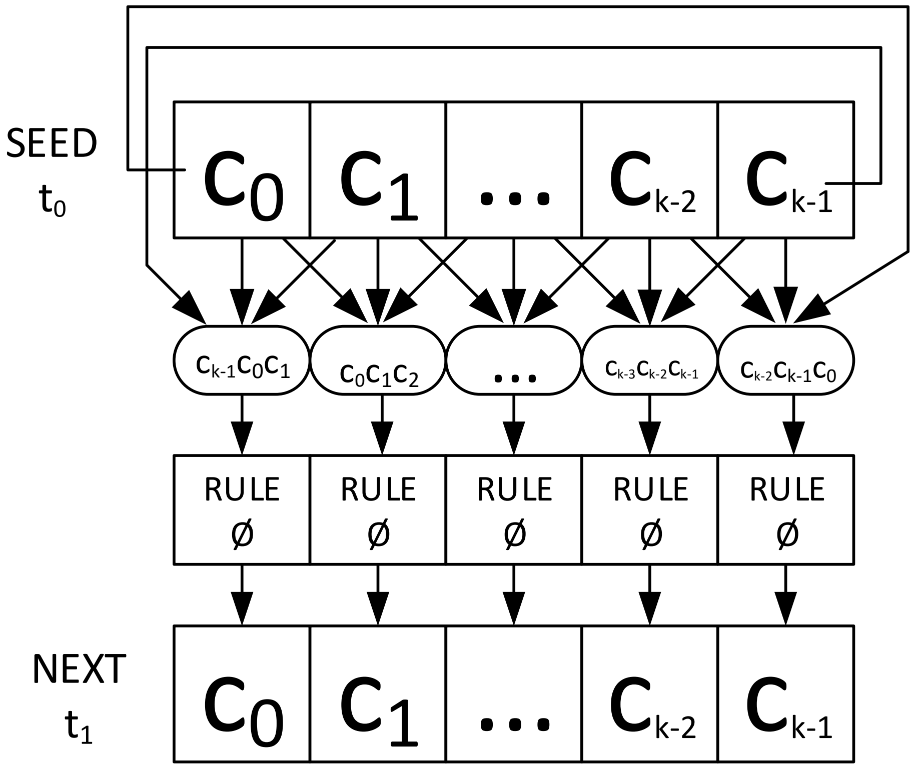
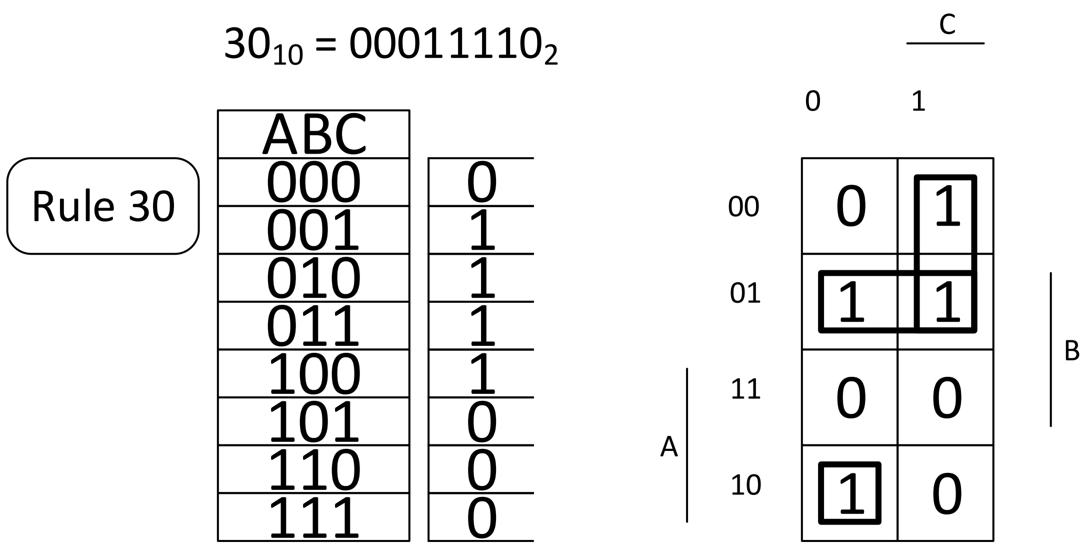


Figure 1 – Elementary Cellular Automata of Size k Using a 3-cell Neighborhood

SECTION 1.2 – Rules and Neighborhoods

A rule is built from a fundamental function or polynomial expression of a basic binary neighborhood – in the case of ECA the neighborhood consists of the cell and its two adjacent cells, or 3 bits. As a result, there are only 8 (or 2^3) states that this neighborhood can exhibit, each producing a different output based on their position in that rule's lookup table (LUT), shown in Figure 2. Expanding the possible rules for a neighborhood of 3 bits creates a complete neighborhood of 256 (or 2 ^ [2^3] = 2^8) fundamental functions, denoted by their decimal value, i.e. Rule 0 through Rule 255. To accommodate the left and right extremes' lack of a third adjacency, the ECA is looped around as shown in Figure 1, making the extremes adjacencies of each other.

$30_{10} = 00011110_2$



ABC	
000	0
001	1
010	1
011	1
100	1
101	0
110	0
111	0

$$A'B + A'C + AB'C' = A \text{ XOR } (B \text{ OR } C)$$

Figure 2 – Tri-Cell Neighborhood Diagram with k-Map Reduced Function - Rule 30

Applying the fundamental rules across the ECA produces its next evolution, or state. Following the initial state, each successive state is a temporal iteration, or timestep, which can be used to describe a specific state of the structure when in combination with the ECA's seed (timestep 0, or t_0) and governing rule. Figure 3 shows Rules 30, 45, 60, and 90 for the 32-bit seed $1111\ 1101\ 1111\ 0111\ 1110\ 0000\ 1110\ 1011_2$.

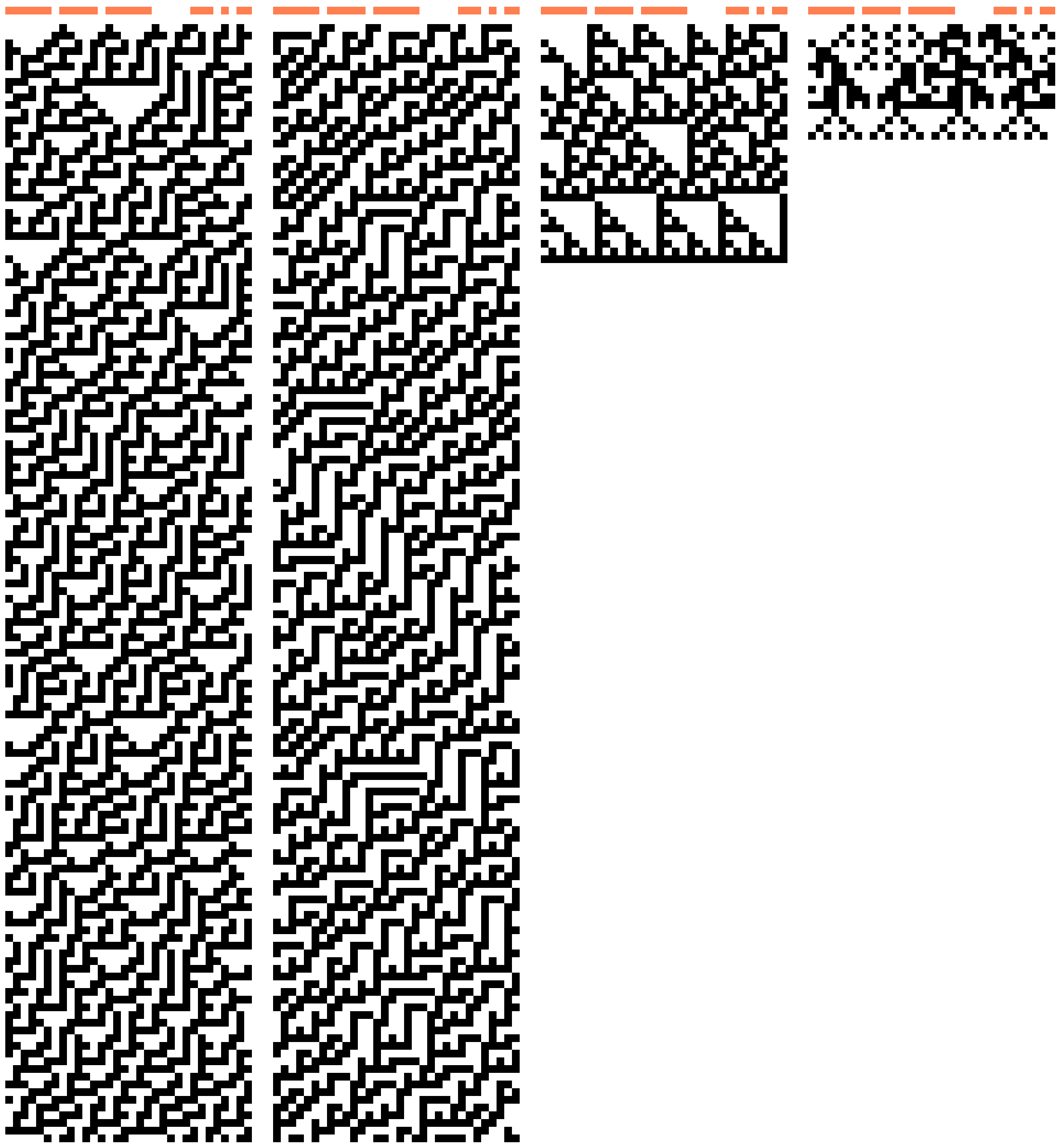


Figure 3 – Automation of ECA with Rules 30, 45, 60, and 90 (left to right)

SECTION 1.3 – Adjacencies

Varying these rules and the way the neighborhoods are represented leads to a number of basic configurations. Extending the neighborhood to 5 bits produces (2 ^ [2 ^ 5] = 2^{32}), or 4,294,967,296 rules, and further extending the neighborhood to 7 bits produces (2 ^ [2 ^ 7] = 2^{128}), or 340,282,366,920,938,463,463,374,607,431,768,211,456 rules. CA rapidly becomes exponential as the configurations and basic neighborhoods evolve beyond their ECA limits. In addition, each cell's adjacencies are not static, allowing for even more variation in the basic representation of single dimension CA by moving the physical adjacency to cells that do not directly connect, shown in Figure 4.

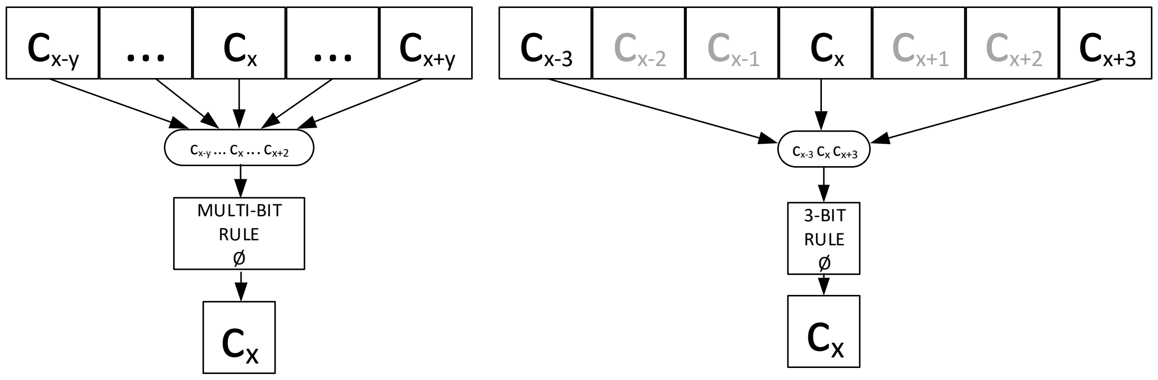


Figure 4 – Extending ECA Neighborhoods and Changing Adjacencies

Representations of the CA lattice also lend to variation. Expanding into multiple dimensions and including additional generators on the boundaries are prime examples. In those cases, the use of fundamental rules changes to use more than one neighborhood and rule results are combined. John von Neumann's two dimensional neighborhood of four adjacent cells is one of the first examples of this configuration. See chapter two for more detail on configurations.

SECTION 1.4 – Complete Description of a k-Space

CA are finite objects – the k-Space of any particular rule can be completely defined for study and analysis. For example, an ECA of length 8 has a complete k-Space of (2^8 = 256) potential states. In this space, there are three components – Cycles, Transients, and Garden of Eden States (GoES). Any given state is considered to fall into one of these three groups. When analyzed together, they show the CA's "basin of attraction".

SECTION 1.4.1 – Garden of Eden States (GoES)

A state that cannot be entered from any other state in the k-Space is considered a Garden of Eden state. Without the reverse algorithm (see 5.2), determining if a state is a GoE would require analyzing the next states of *every* state, putting them into a two column list and determining what entry on the left does not occur in the right, as shown in Figure 5.

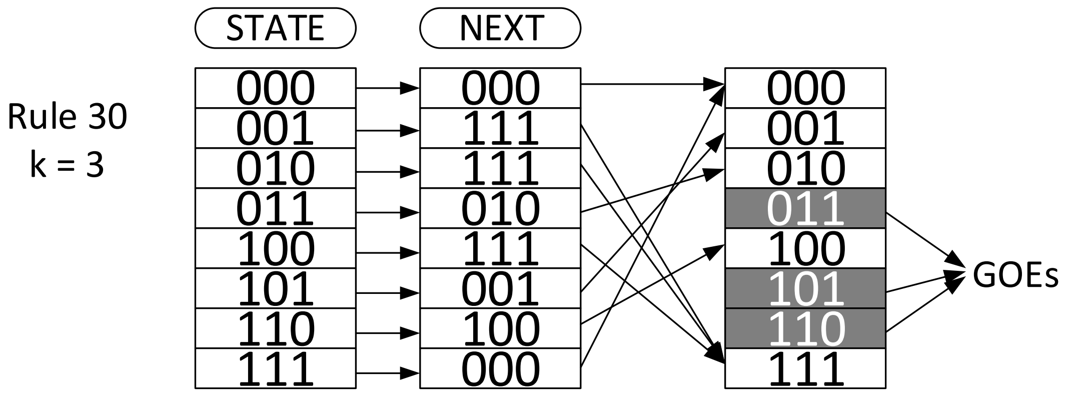


Figure 5 – GoE Discovery, Two Column Approach

SECTION 1.4.2 – Transients

A state that is only entered once in automation is considered a transient. All transients originate from GoEs, since they will never be entered. By process of elimination, any state that does not fall into a cycle and is not a GoE must be a transient. Transient length is determined by the number of cycles between the state and cycle entry. In the complete analysis, the longest transient refers to the GoE state with the longest length before cycle entry. Figure 6 shows an example of two transients.

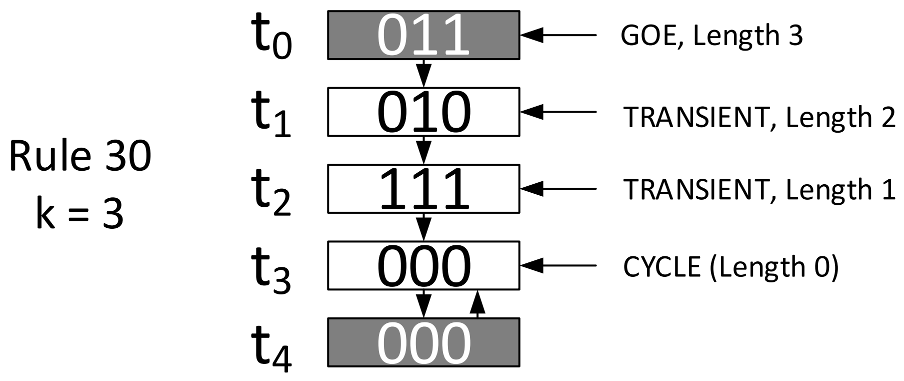


Figure 6 – Transient Diagram Using Rule 30, Size k = 3, GoE 011

SECTION 1.4.3 – Cycles

Eventually, a state will be entered which will iterate into other states until it repeats, forming a cycle due to the natural law of the rule. There are three types of cycles. A regular cycle occurs when a transient or GoE enters a state that iterates for a number of states before the original entry is repeated. A single cycle occurs when the state cycles to itself – for Rule 30 this happens with a seed of all zeroes, since the next state is (000) = 0, as seen in Figure 6. An orphan cycle occurs when no entry points are defined. For Rule 105, this happens with even-length seeds of alternating 0s and 1s (ex. 01010101, k = 8), which forms the next state of 10101010, then back to 01010101. Figure 7 shows the differences between types. For Rule 105, k = 4, there are no transients or GoEs, making all the cycle states orphans.

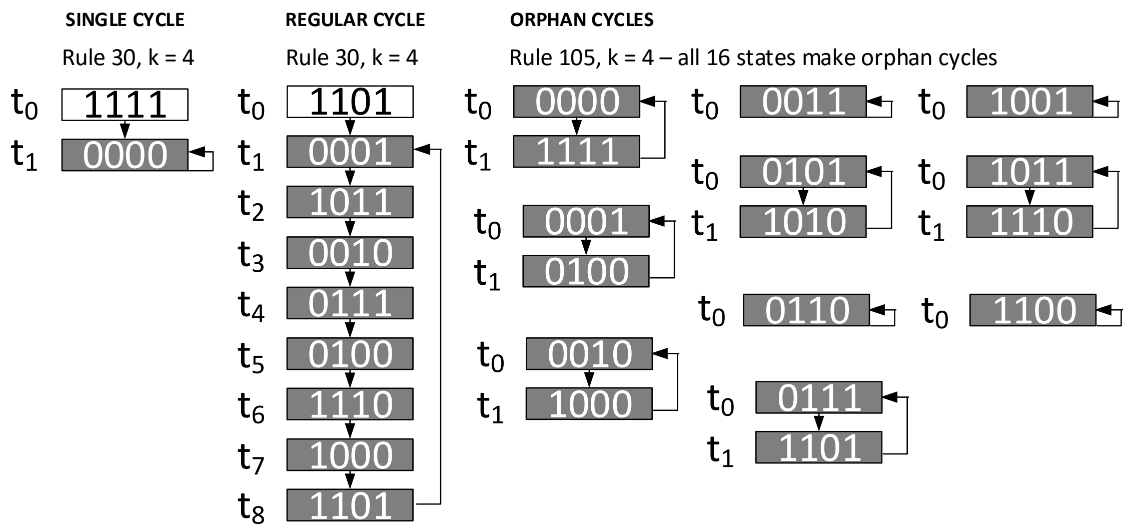


Figure 7 – Cycle Types: Single, Regular, and Orphan

SECTION 1.4.4 – Basin of Attraction

Combining GoEs, transients and cycles will show a complete picture of the k-Space of a CA in the form of basins of attraction, where each non-orphan cycle is considered an attractor. Since there can be many unique cycles, there can be many attractors. The very outmost leaves are GoEs and their inner branches are transients. Figure 8 shows the basin of attraction for rule 30, k = 4 in decimal form. There are no transients since each GoE immediately enters a cycle. States 5 and 10 (decimal values) are orphan cycles. The main attractor is made up of 1, 11, 2, 7, 4, 14, 8, and 13. The 0 state is a minor attractor.

STATE		NEXT	
0000	0	0000	0
0001	1	1011	11
0010	2	0111	7
0011	3	1110	14
0100	4	1110	14
0101	5	0101	5
0110	6	1101	13
0111	7	0100	4
1000	8	1101	13
1001	9	0111	7
1010	10	1010	10
1011	11	0010	2
1100	12	1011	11
1101	13	0001	1
1110	14	1000	8
1111	15	0000	0

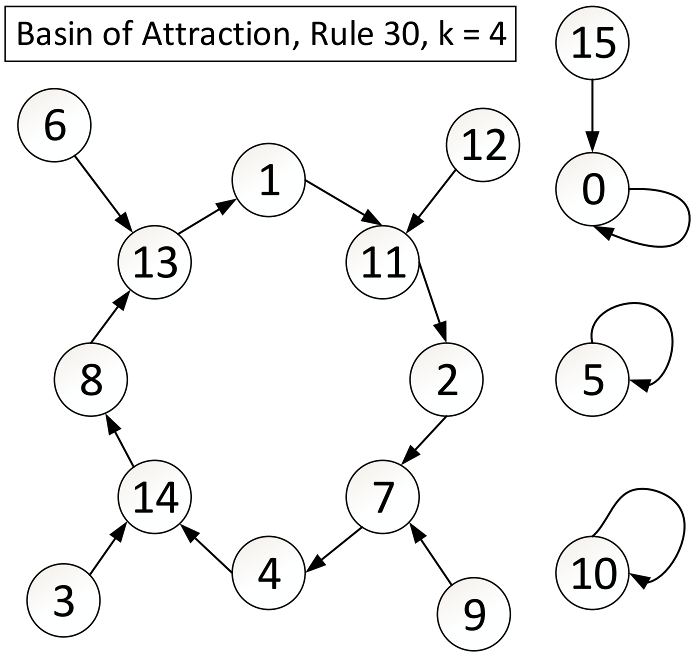


Figure 8 – Basin of Attraction for Rule 30, k = 4

SECTION 1.5 – Research Problem

Cellular automata were first formally described by John von Neumann in 1966 [1] and their use in random number generation described later by Stephen Wolfram in 1986 [2]. A basic element of this research is intended to provide background on specific configurations for the purpose of RNG creation and to form a more complete picture of the practical uses of ECA, in addition to the theoretical concepts. Rather than assume the conclusions of past research as fact, the focus of this research was to reaffirm the core knowledge of ECA and develop a model for testing and automation.

SECTION 1.5.1 – Focus 1: ECA Chaotic Rules and Configurations

1. What rules and configurations are conducive to chaotic structure?

Of the 256 basic functions, which have the longest cycles and are thus chaotic in nature, allowing for pseudo-random variation in output? Research plans include the creation of software that can model and generate ECA rules and configurations, then complete analysis and random number testing on the software's output.

SECTION 1.5.2 – Focus 2: ECA Random Number Generators

2. Can ECA be used for practical random number generation?

While proving certain rules and configurations are chaotic and capable of passing random number tests, at what point is ECA capable of producing random numbers at a practical level? Research plans include rigorous testing of all ECA rules at increasing complexity (rising seed/key lengths) to determine practical RNG applications.

SECTION 1.5.3 – Focus 3: ECA Encryption

3. If RNG creation is possible, how much complexity is necessary for encryption?

Random number generation on its own is not enough to satisfy the growing needs of encryption in the modern world. What kind of complexity is necessary to actualize ECA for use in encryption? Research plans include initial evaluation of various configurations to judge which would be best suited for parallelized encryption, then concentrated testing to determine what rules are best suited for practical encryption.

CHAPTER TWO: CONFIGURATIONS

SECTION 2.1 – Variations

Various CA configurations were designed/tested for use as Pseudo Random Number Generators (PRNGs), differing by their adjacencies, boundaries, ruling combinations, and representations. Each has its own distinct advantages and disadvantages depending on rules and fundamental design. Among them are the 1D (3-bit, 5-bit), 2D (von Neumann, Moore, Arrow, Hexagonal), Cascade, and Inverted Pyramid.

SECTION 2.2 – Spatial and Temporal Rules (1D)

As shown in section 1.1, ECA form the fundamental 1D configuration, but they can be augmented using a variety of tools – including spatial and temporal rules. A spatial rule is a rule applied to a given cell. A temporal rule is a rule applied to an entire timestep. Under a single rule, both the spatial and temporal rules match. When using more than one rule, a choice must be made as to how they are represented. Under a spatial configuration, every alternating cell in an iteration is computed with a separate rule. Under a temporal configuration, every alternating iteration is computed with a separate rule. It is possible to combine the two with differing rules, resulting in a spatial-temporal hybrid, though effects of rule combination and potential collision/cancellation need to be taken into consideration. Figure 9 displays the differences between the two.

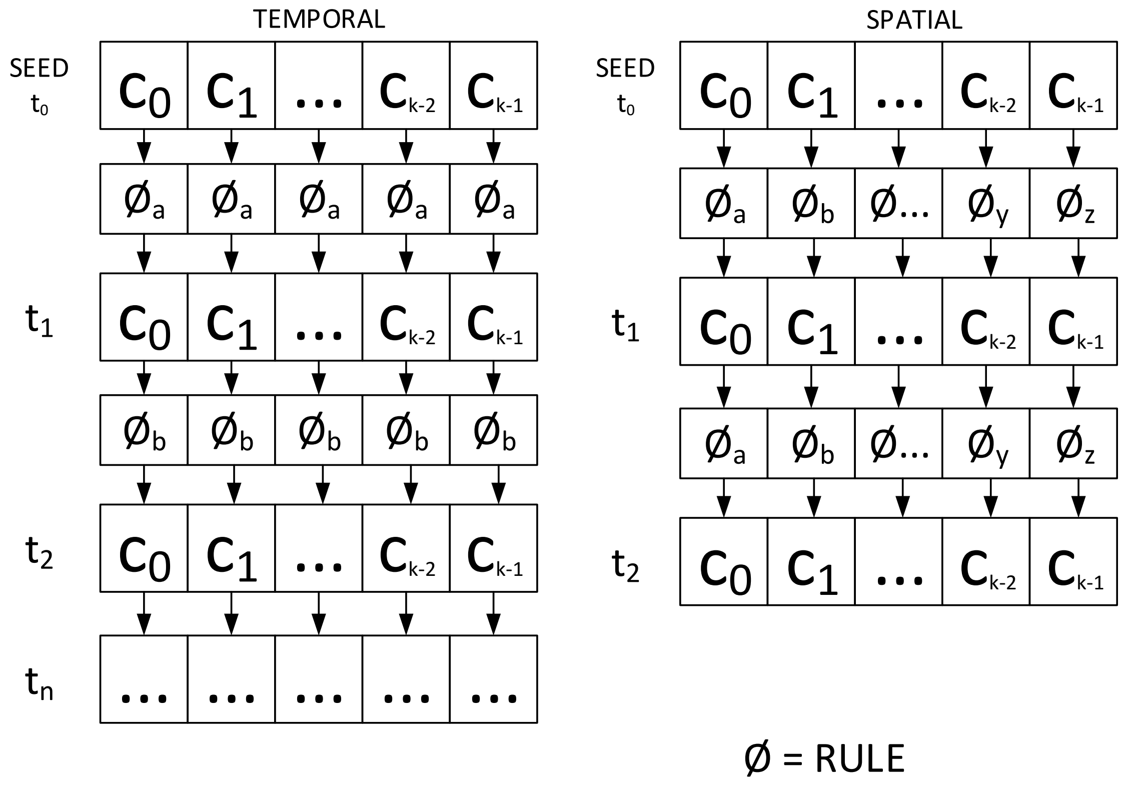


Figure 9 – Temporal (left) and Spatial (right) Ruling

SECTION 2.3 – Von Neumann Neighborhood (2D)

Expanding the 1D ECA to a lattice or matrix of cells allows the representation of CA in two dimensions. Instead of two adjacencies, the lattice provides up to 8 direct connections. John von Neumann originally limited this neighborhood to the four immediate adjacencies on the cell perimeter. Just as in ECA, 2D grids are continuous in nature, analogous to a torus. To continue using the 3 bit rules, three steps are involved for computing the next state of a cell, as shown in Figure 10. A cell is computed horizontally, vertically, then ruled with a combination of the two results and the value of the original cell. The same or different rules can be specified for each calculation.

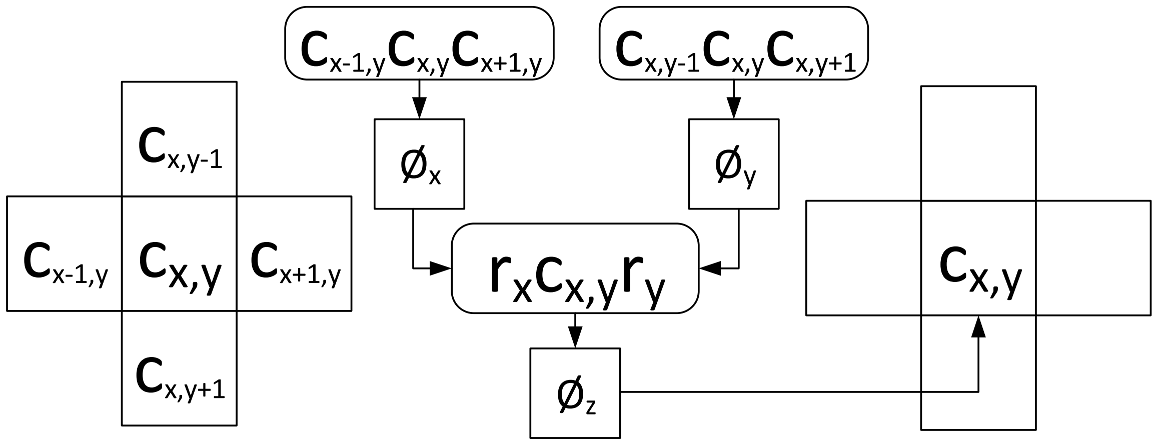


Figure 10 – 2D Von Neumann Neighborhood

SECTION 2.4 – Moore Neighborhood (2D)

Edward F. Moore described his neighborhood using all 8 surrounding cells, leaving a total of 9 values for computation. Four steps are required to compute the next rule, generated horizontally or vertically, as shown in Figure 11. Three results are computed from rows (horizontal) or columns (vertical) then ruled together for the final result.

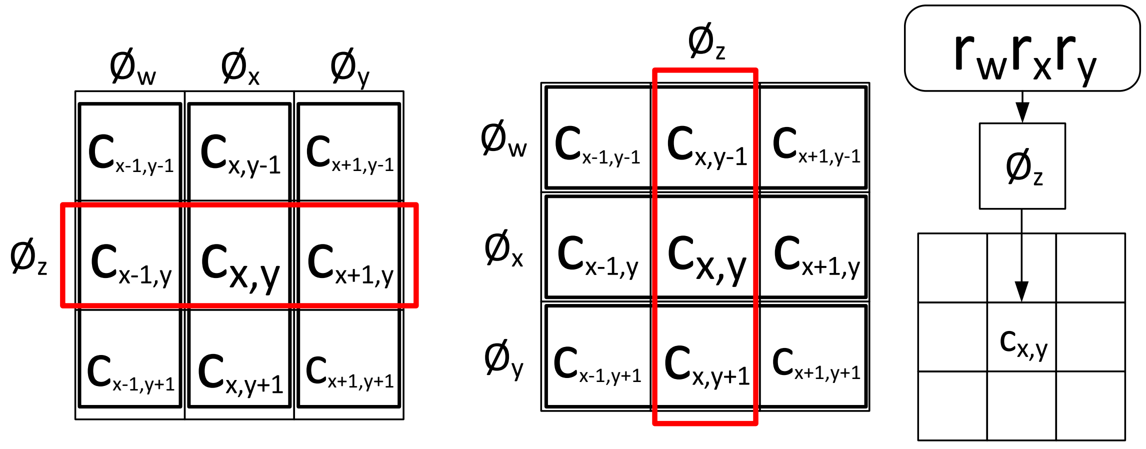
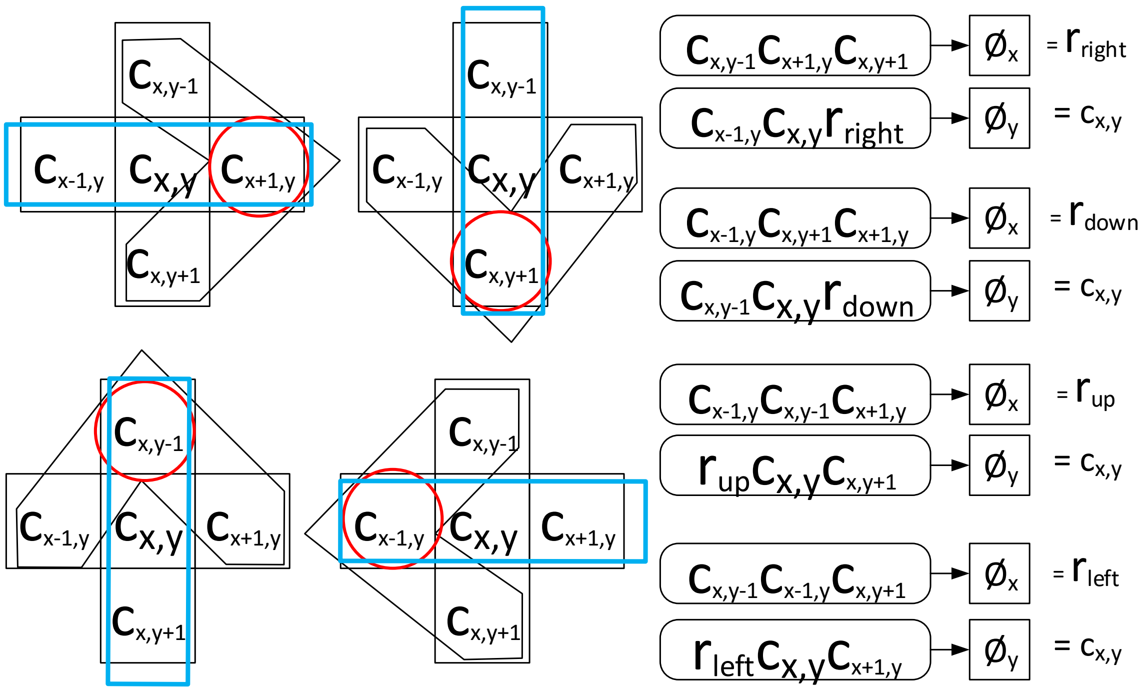


Figure 11 – Moore Neighborhood, Horizontal (left) and Vertical (right)

SECTION 2.5 – Arrow Neighborhoods (2D)

Arrow configurations are an augmentation of von Neumann neighborhoods, using a direction to pull an initial rule. Only two steps are involved for calculation. First, the directional bit (up, down, left, right) is ruled in conjunction with the two nearest adjacencies then ruled in the order they appear on the grid, shown in Figure 12.



$c_{x,y-1}$
$c_{x-1,y}$
$c_{x,y}$
$c_{x+1,y}$
$c_{x,y+1}$
$c_{x,y-1}c_{x+1,y}c_{x,y+1} \rightarrow \varnothing_x = r_{right}$
$c_{x-1,y}c_{x,y}r_{right} \rightarrow \varnothing_y = c_{x,y}$
$c_{x-1,y}c_{x,y+1}c_{x+1,y} \rightarrow \varnothing_x = r_{down}$
$c_{x,y-1}c_{x,y}r_{down} \rightarrow \varnothing_y = c_{x,y}$
$c_{x-1,y}c_{x,y-1}c_{x+1,y} \rightarrow \varnothing_x = r_{up}$
$r_{up}c_{x,y}c_{x,y+1} \rightarrow \varnothing_y = c_{x,y}$
$c_{x,y-1}c_{x-1,y}c_{x,y+1} \rightarrow \varnothing_x = r_{left}$
$r_{left}c_{x,y}c_{x+1,y} \rightarrow \varnothing_y = c_{x,y}$

Figure 12 – Arrow Neighborhoods (2D)

SECTION 2.6 – Hexagonal Neighborhoods (2D)

A hexagonal grid contains cells with six adjacencies, leaving multiple interpretations for rules and operations. Three results can be obtained by gathering rules from the hexagonal axes (x, y, z) then combined for a final result, similar to the Moore neighborhood. Alternatively, two results can be obtained from the outer shell (3 bit halves) then ruled together with the internal bit. Adjacency wrapping needs to be taken into consideration since some grid sizes do not symmetrically mirror boundaries. Figure 13 shows axis ruling with 4 calculations and shell ruling with 3 calculations.

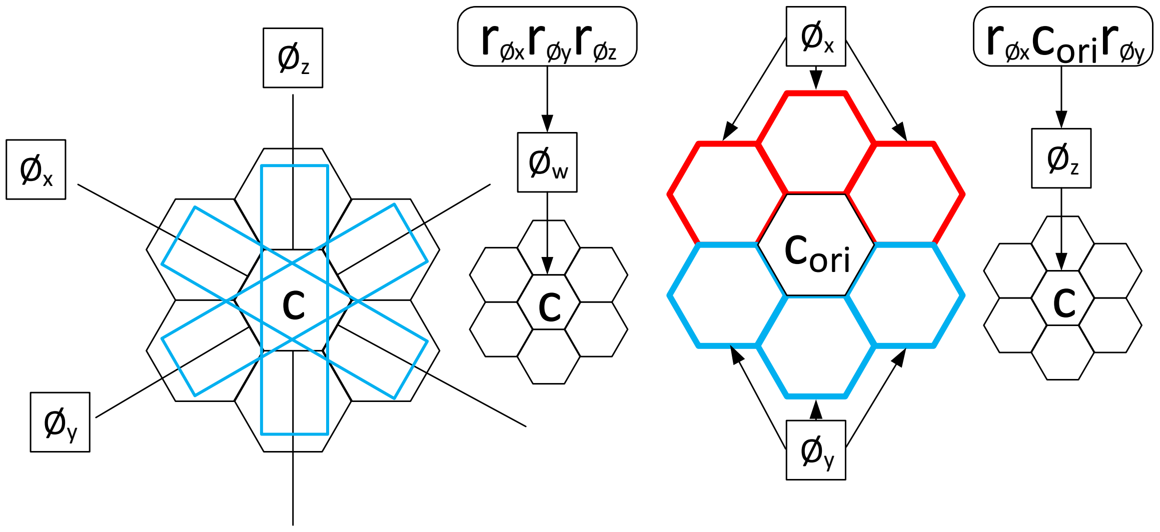


Figure 13 – Hexagonal Neighborhoods (2D), Axis Ruling (left) and Shell Ruling (right)

SECTION 2.7 – Cascade (1.5D)

Cascade configurations generate sub-blocks of CA based off an initial seed block. An arbitrary length (K_1) is used to generate an arbitrary number of timesteps (T_1) with a starting rule. A number of sub-blocks (equal to K_1) are seeded from the vertical columns in the original seed block, meaning ($T_1 = K_2$). The sub-blocks can then be independently iterated and the data collected for use in a PRNG. T_2 is decided by how much data is requested, using the following formula and adding one to compensate for estimation:

$$T_2 = \frac{data}{(K_1 * T_1)} + 1$$

Figure 14 shows a diagram of the cascade configuration using a seed block of k = 8, 7 timesteps, and Rule 30. Karim Salman describes this structure in his ECA research platform and refers to it as “Twister”. [3]

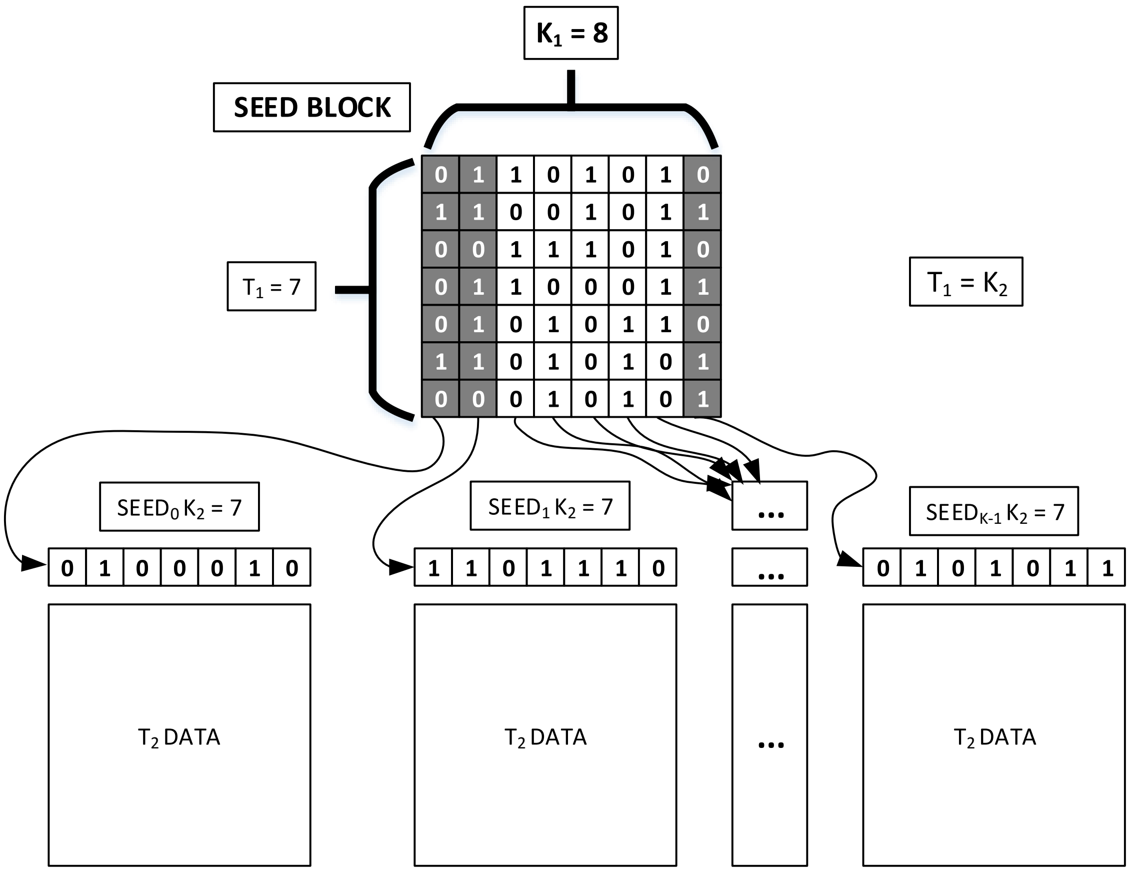


Figure 14 – Cascade Configuration, Using a Starting Block of Size 8 with 7 Timesteps

SECTION 2.8 – Inverted Pyramid (1D)

A fundamental problem with ECA is that the boundaries wrap and influence the entire lattice within a set number of timesteps. To avoid this influence, a CA analogous to an inverted pyramid is generated, starting with a large odd-numbered k and losing two bits per timestep until it reaches the final step, with a single cell remaining, shown in Figure 15. Salman describes this as the unbounded ECA [3]. By not using the extremes of each timestep, the noise influence from wrapping is negated. For purposes of random number generation, the data size must be used to determine the initial k. The next whole integer (ceiling) of the data length's square root is taken, incremented to account for the row that will be used by the seed (so it does not mix with the end results), multiplied by 2, then subtracted by 1 to gain an odd-numbered k that will produce at or greater than the data length requested, according to the following formula:

$$k = \left(ceiling\left(\sqrt{data}\right) + 1\right) * 2 - 1$$

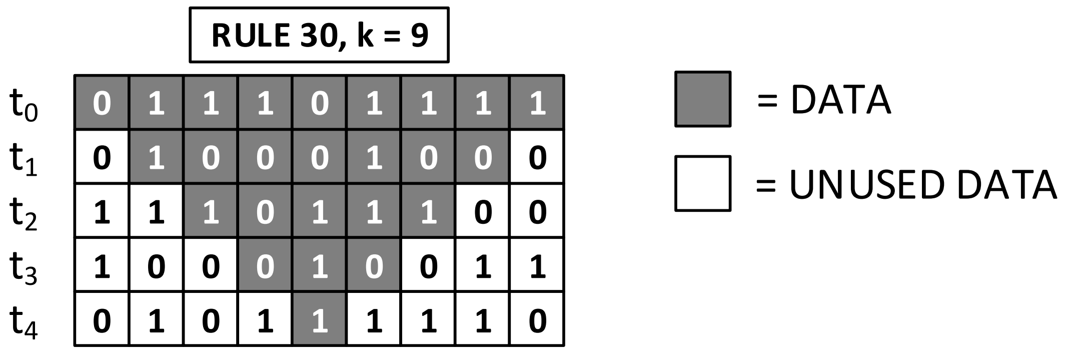


Figure 15 – Inverted Pyramid Configuration with Rule 30, k = 9

CHAPTER THREE: CELLULAR AUTOMATA STANDARD OF ENCRYPTION

SECTION 3.1 – Principle Theory

Encryption standards are tricky systems to create, being only as strong as the principle and math behind them. Cellular Automata's biggest strength lies in the guarantee of computation – there is no known way to skip computation and figure out the exact data from a seed's timestep without actually calculating all the intermediate steps. In that respect, introducing a "critical time" factor can also increase complexity. A cellular automata standard of encryption, hereby referred to as CASE, can be used to encrypt data in a desired key configuration, similar to the way AES (the U.S. approved Advanced Encryption Standard) has 128, 192, and 256 blocks. There is no real limit on the key size, so long as the same encryption method is utilized on the decryption end. For the system described in CASE, the Cascade configuration is modified to accommodate certain key sizes. The seed block and cascade rules used in this system will need to be ones that produce the most reliable passing pseudo-random results in all tests – in other words, the combined result of the three focal points for the research problem.

SECTION 3.2 – CASE Structure

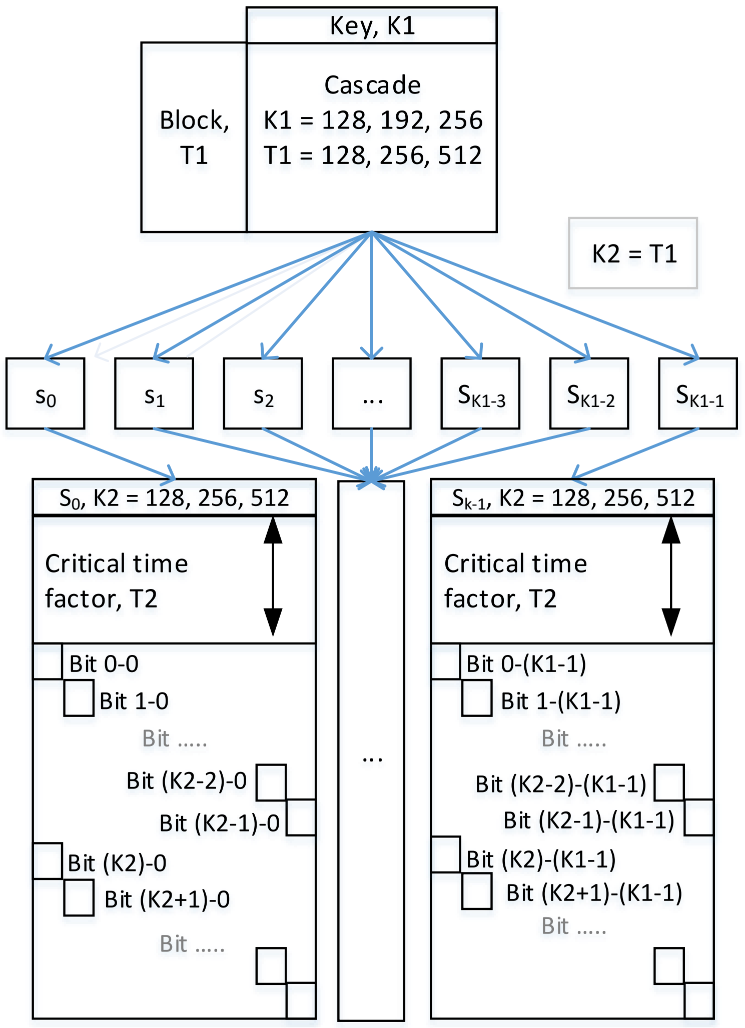


Figure 16 – CASE Structure and Breakdown

Figure 16 shows the CASE structure, with 9 basic configurations suggested. Further configurations are possible, but for simplicity only three choices are shown for the key and internal complexity sizes. The key can be a 128, 192, or 256-bit number, which is generated by hashing or encrypting an ASCII-input character into a 16, 24, or 32-byte array. The block size is the required amount of data to encrypt – i.e. a file or array of data will be segmented into 128, 192, or 256 bit blocks, then encrypted using this method, with an internal complexity generated by the T1 value. 128-bit CA will not cycle for the critical period and by using bits from each seed, there should be more than enough entropy to produce pseudo-random results. To increase the amount of time required to decrypt, a critical time factor (of a very high magnitude) can be added, but since such information would not be stored in a key, it would more likely become a CASE configuration. Such factors could be thousands, millions, or billions of timesteps – there is no imposed limit. The decryption end would need to run through the calculations before arriving at usable stream data. This could be used to combat key interception for ultra-critical time periods and also to eliminate correlation between the key block and the stream data.

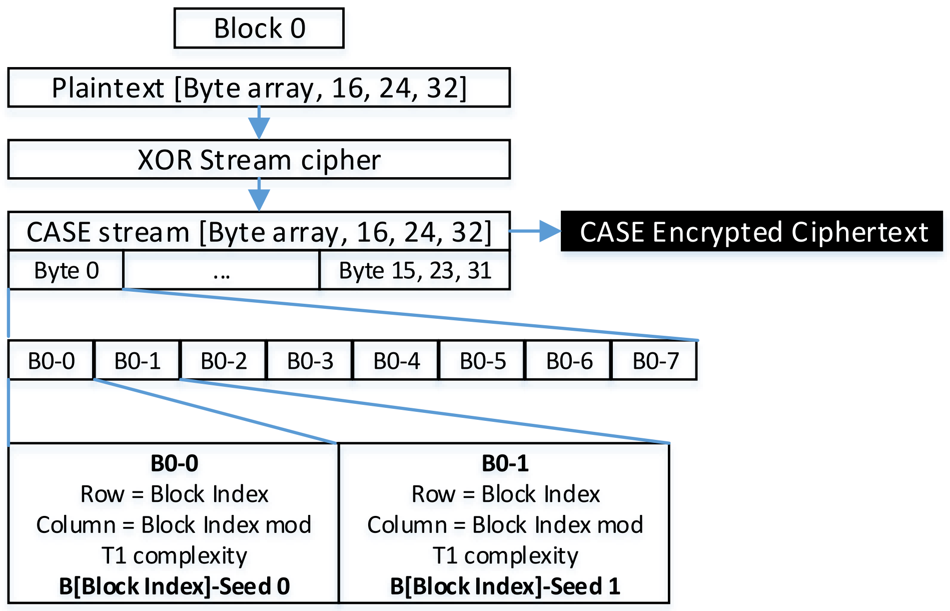

Block 0
Plaintext [Byte array, 16, 24, 32]
XOR Stream cipher
CASE stream [Byte array, 16, 24, 32]
CASE Encrypted Ciphertext
Byte 0
...
Byte 15, 23, 31
B0-0
B0-1
B0-2
B0-3
B0-4
B0-5
B0-6
B0-7
B0-0
Row = Block Index
Column = Block Index mod T1 complexity
B[Block Index]-Seed 0
B0-1
Row = Block Index
Column = Block Index mod T1 complexity
B[Block Index]-Seed 1

Figure 17 – CASE Block Breakdown

Figure 17 shows the block breakdown as CASE moves through its paces. The row index will change if a critical time factor is added.

SECTION 3.3 – CASE Configurations

Similar to how current encryption systems have levels of use for classified or critical data, basic configurations for CASE and their suggested standard names are summarized in Table 1. T1 complexity will need to be investigated on a case by case basis, and will not be represented here.

Table 1 – CASE Standards Summary

CASE Standard	Usage	Breakdown
CASE-128	General encryption	128-bit key/block size
CASE-192	General encryption	192-bit key/block size
CASE-256	General encryption	256-bit key/block size
CASE-128T1 CASE-192T1 CASE-256T1	Low-level time-sensitive hardware	CASE-XXX plus T1 time factor of 1 million timesteps
CASE-128T2 CASE-192T2 CASE-256T2	Mid-level time-sensitive hardware	CASE-XXX plus T2 time factor of 1 billion timesteps
CASE-128T3 CASE-192T3 CASE-256T3	High-level time-sensitive hardware	CASE-XXX plus T3 time factor of 100 billion timesteps

What this breaks down in to is the possibility of a computing system having mapped out all the possible 128, 192, and 256-bit CA diagrams. As demonstrated in the state diagram data in Section 5.5, the data necessary to hold these diagrams is too cumbersome for any real world theoretical storage. As computing power progresses, CASE standards can be scaled to match with higher time critical factors, leaving the general scheme intact. As such, provided no attacks on the system are extremely severe, it should be able to withstand the test of time, which is why it is very important to choose rules that have the highest possibility of maintaining pseudo-random streams.

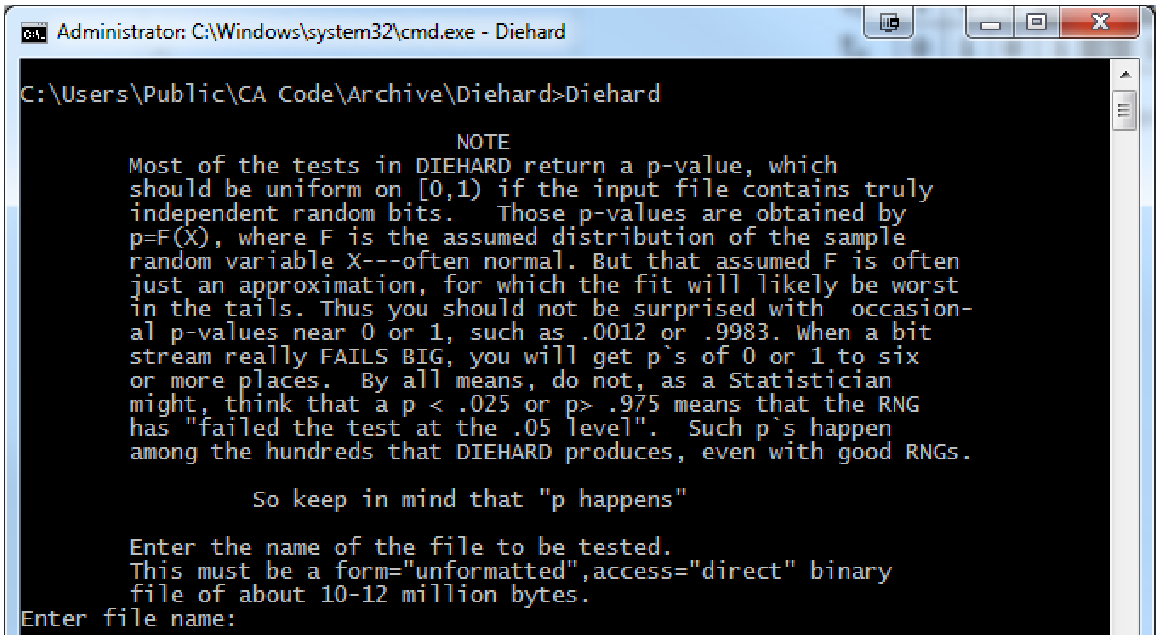
CHAPTER FOUR: RANDOM NUMBER TESTING

SECTION 4.1 – Fallacy of Random Number Generation

Contrary to first impressions, the generation of random numbers means that by definition they are no longer "random", in that they are not unexpected. In respect to an arbitrary viewpoint, they may *appear* random, which is why this process is always referred to as *pseudo* random number generation. CA are completely predictable provided you know the starting state and acting rules. In the same manner, one could in theory totally predict Earth's weather with exact pressure simulations, measurements, and all applicable knowledge needed for the current state. Since this is not technically feasible as of this writing, the weather remains somewhat "random".

SECTION 4.2 – Diehard Battery of Randomness Tests

Originally written in Fortran, the Diehard battery of tests is a common utility used to measure the statistical randomness of a given set of data, parsed into 32-bit numbers [4]. There are 17 tests that can be run, provided there is enough binary data for 68 million random 32-bit integers (67,108,889 * 4 bytes = 256 MB). Three tests can be omitted for smaller data sets with a minimum of 2.5 million 32-bit integers. 229 statistical p-values are generated from these tests, along with an overall p-value. The importance of these values is explained in section 3.4. Figure 18 shows the starting prompt for the Diehard program, with a short caveat on p-values.

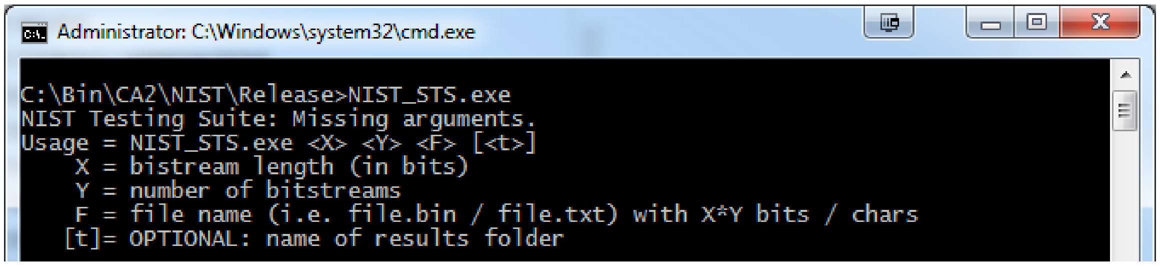

Administrator: C:\Windows\system32\cmd.exe - Diehard

```
C:\Users\Public\CA Code\Archive\Diehard>Diehard

                                NOTE
        Most of the tests in DIEHARD return a p-value, which
        should be uniform on [0,1) if the input file contains truly
        independent random bits.   Those p-values are obtained by
        p=F(X), where F is the assumed distribution of the sample
        random variable X---often normal. But that assumed F is often
        just an approximation, for which the fit will likely be worst
        in the tails. Thus you should not be surprised with  occasion-
        al p-values near 0 or 1, such as .0012 or .9983. When a bit
        stream really FAILS BIG, you will get p`s of 0 or 1 to six
        or more places.  By all means, do not, as a Statistician
        might, think that a p < .025 or p> .975 means that the RNG
        has "failed the test at the .05 level".  Such p`s happen
        among the hundreds that DIEHARD produces, even with good RNGs.

                 So keep in mind that "p happens"

        Enter the name of the file to be tested.
        This must be a form="unformatted",access="direct" binary
        file of about 10-12 million bytes.
Enter file name:
```

Figure 18 – Diehard Command Prompt with Brief Explanation of P-Value Results

SECTION 4.3 – NIST Statistical Testing Suite (STS)

The U.S. National Institute of Standards and Technology (NIST) provides a statistical testing suite, shown in Figure 19, written in C for the use of testing random number generators [5]. Similar to Diehard, it provides 15 tests for randomness, however it uses several independent streams of data to perform many tests at once. Using 100 streams of 400,000 bits each (40,000,000 bits total), every test in the suite can be run for a rounded result. The testing suite documentation gives details on the acceptance of any particular test and passing results of over 90% can be safely called "random".


Administrator: C:\Windows\system32\cmd.exe

```
C:\Bin\CA2\NIST\Release>NIST_STS.exe
NIST Testing Suite: Missing arguments.
Usage = NIST_STS.exe <X> <Y> <F> [<t>]
    X = bistream length (in bits)
    Y = number of bitstreams
    F = file name (i.e. file.bin / file.txt) with X*Y bits / chars
   [t]= OPTIONAL: name of results folder
```

Figure 19 – NIST STS, Recoded to Pass in File Names with Internalized Options

SECTION 4.4 – Probability and p-Values

In respect to random number generation, a p-value (ranging from 0.0 to 1.0) is a measure of probability that the given set of data is "random", according to a threshold specified by α (alpha) [6]. Five steps in evaluating random p-value results are:

1. Set null hypothesis – "This set of data is random."
2. Set alternative hypothesis – "This set of data is not random."
3. Calculate the p-value for the random test (in this case, Diehard or NIST is responsible for this step), using the known distributions for the given tests.
4. Choose a significance level (alpha). An alpha that is too small will invariably judge many sets random and an alpha that is too high will invariably judge sets to be non-random.
    a. NOTE: The documentation for NIST uses an alpha of 0.01. For most early Diehard testing, an alpha of 0.0005 was used, which was later considered too lenient. For the final tests, an alpha of 0.01 was used for both testing suites.
5. Judge p-value based on alpha. If inside the threshold (0 + alpha < p-value < 1 - alpha), accept the null hypothesis. If at or outside the threshold, reject the null hypothesis in favor of the alternative.

SECTION 4.5 – Normal Distribution

A single p-value is not usually considered enough to form a decision, which is why Diehard and the NIST STS provide so many tests. When many p-values are calculated for a given set of data, it is important that they follow a normal distribution. Deviations from this distribution indicate that there are patterns within the data that came to light in a particular group of tests with similar pressure points. Figure 20 shows two results from Diehard testing – on the left is a result that is likely random since it follows a normal distribution. On the right, the p-value graph is skewed upward, meaning it is likely to be non-random.

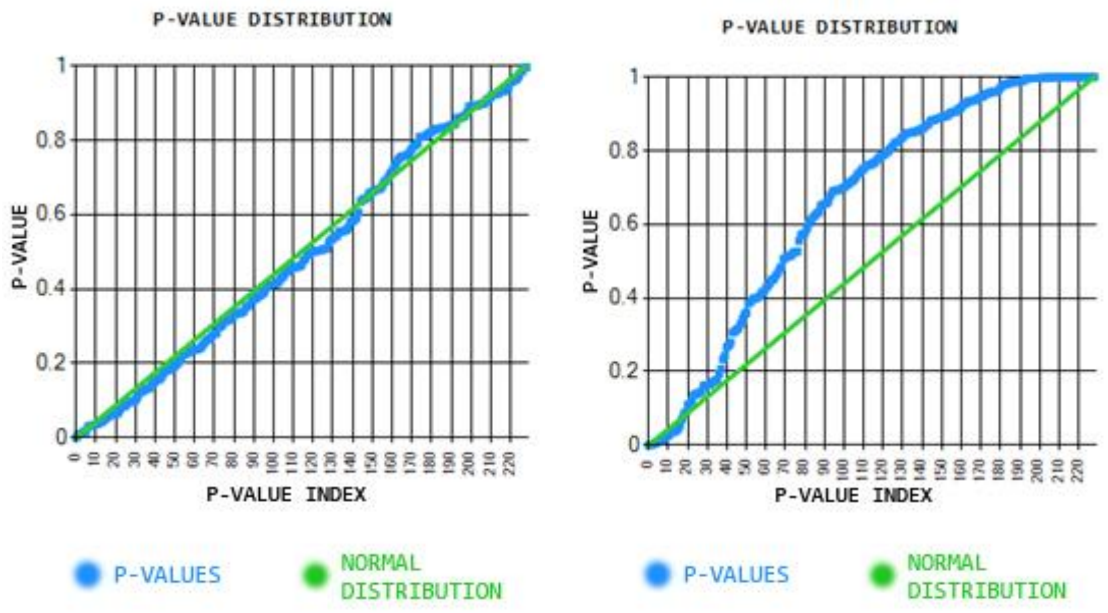


Figure 20 – P-value Distributions: Likely Random (left), Non-random (right)

CHAPTER FIVE: CODING AND TESTING CA

SECTION 5.1 – Optimization and Best Practices

ECA are simple structures which can be replicated in virtually any programming language built with an array data structure. Important considerations are calculation efficiency, representation, and ease of coding. Writing elementary calculators in Python 2.7 may only take a handful of lines, but the calculations may take 1000x longer than the equivalent in C, C++, or other languages closer to pure assembly. The following best practices were devised to handle CA programming from scratch:

1. Learn CA first. Repeating old methodology is an easy trap to fall in.
2. Choose a comfortable language. Preferably "fast" and well documented.
3. Avoid excessive use of expensive data types.
    a. For example, storing 0000 1101 1100 1100 1011 1011 1100 1100 (32 cells) as a string takes up to 32 bytes in languages where characters are stored as an 8-bit data type.
    b. Instead, if a 32-bit integer data type or array of 32 Boolean (true/false, 1/0) variables is used, the data size is simply 4 bytes (32 bits). As a result, any actions to handle those bits become much simpler than manipulating characters in a string, which normally have a high data overhead.
4. Separate the CA calculator program from any custom User Interfaces (UIs).
5. Plan for data consumption – CA is by its nature exponential, and any data gathered will grow at an accelerated pace.
6. Use source control and backups – data loss and power outages are unavoidable.

SECTION 5.2 – Object Oriented Approach

Early program designs were focused primarily on proof of concept CA calculators and on-demand functionality, which while useful for gaining a technical understanding of CA, valuable time could have been lent more effectively toward automating testing and analysis. To that end, major efforts were made to consolidate the programming constructs into a cohesive entity as well as to implement source control and maintenance to aid in continued development. A framework for these operations is shown in Figure 21, with the workbook generating and analyzing tests via automation.

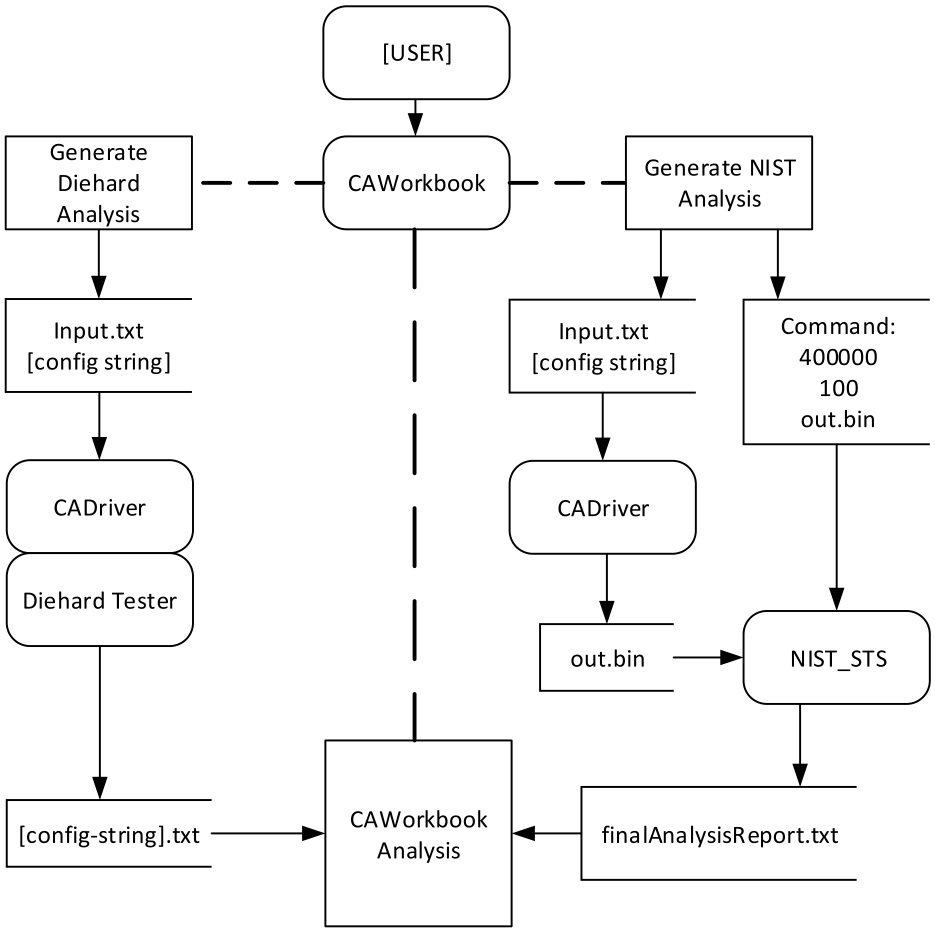
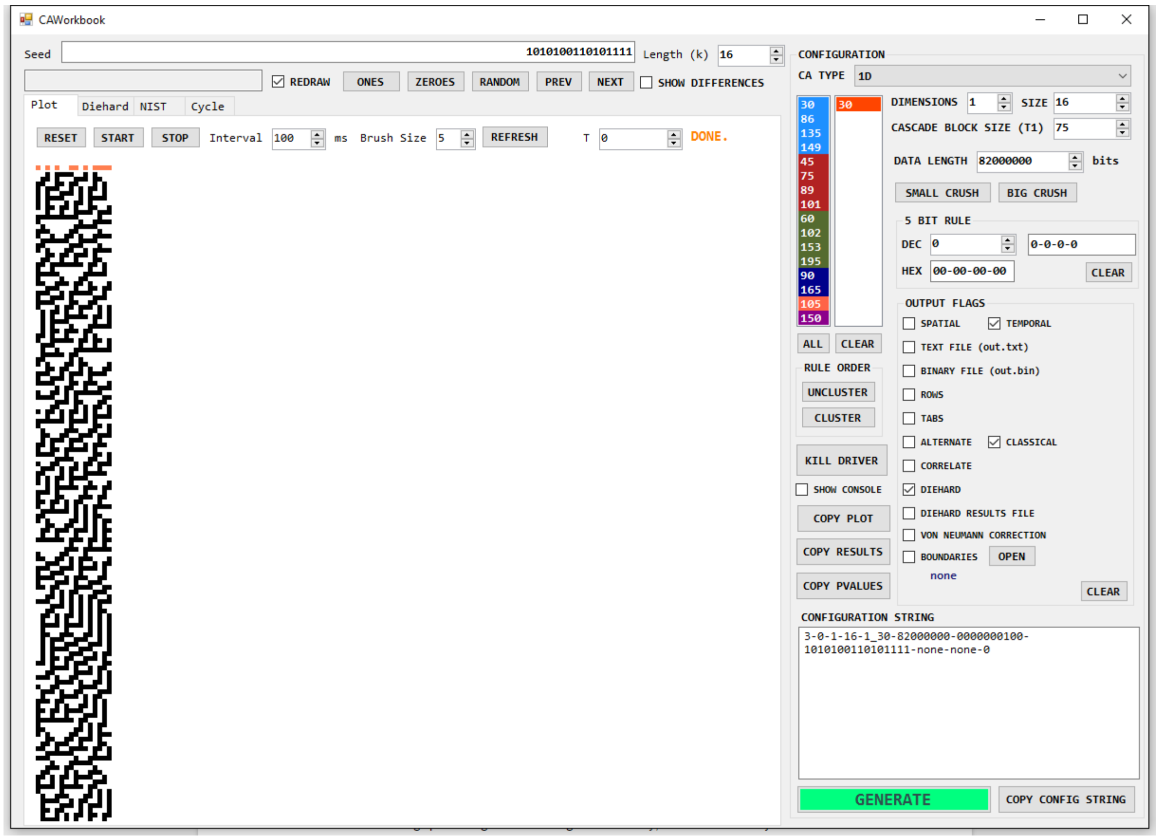


Figure 21 – CA Testing Framework

SECTION 5.3 – CA Workbook



Seed 1010100110101111 Length (k) 16

Interval 100 ms Brush Size 5 T 0

CA TYPE 1D

DIMENSIONS 1 SIZE 16

CASCADE BLOCK SIZE (T1) 75

DATA LENGTH 82000000 bits

DEC 0 0-0-0-0

HEX 00-00-00-00

Rules: 30, 86, 135, 149, 45, 75, 89, 101, 60, 102, 153, 195, 90, 165, 105, 150; selected: 30

Output flags checked: TEMPORAL, CLASSICAL, DIEHARD

CONFIGURATION STRING

```
3-0-1-16-1_30-82000000-0000000100-
1010100110101111-none-none-0
```

Figure 22 – CA Workbook, Visual C# (.NET 4.6)

To more effectively visualize the CA generation and aid in testing, the CA Workbook shown as a screenshot in Figure 22 was created to act as a front end interface for accessing the CA Driver and generating random data for testing. Options for Diehard and NIST STS testing are interfaced by passing CA configuration strings to the driver program, sending their output to the testers then analyzing their results. Since research is ongoing, the software grows as more CA configurations and analysis options are added. See Appendix B for a brief breakdown of all the screens as of this writing.

SECTION 5.3.1 – Diehard Analysis

To generate a Diehard test, the form creates a configuration string based on the CA options, then feeds it into the driver program, which dissects the options and generates enough data to fill the requested length. Diehard is integrated into the driver, and it runs its tests as soon as data generation is completed, then outputs a results file with 230-270 p-values, depending on the data length. If there were not enough random numbers for testing, the results file is not generated. Once alerted that testing is complete, the CA Workbook program analyzes the result.

SECTION 5.3.2 – NIST STS Analysis

To generate a NIST test, the configuration string is passed to the driver requesting enough data for 40 million bits while a command is passed to the NIST STS requesting 100 stream tests of 400,000 bits each, taken from a binary file called "out.bin". Once the driver generates the data, it stores it in a binary file then the NIST suite begins testing on the file. Once completed, a final analysis report is generated and then analyzed by the CA Workbook program.

SECTION 5.4 – Configuration Strings

Four revisions of the configuration string system were created in the course of testing, each adding functionality or correcting past oversights, making them incompatible between each other. Revision 0 sets the basic format, containing all information for the CA configuration and the seed to fill the starting state. Revision 1 added an option for external data to be fed into the boundaries of CA (undoing the ECA wrap-around) as well as a test number feature for automation. Revision 2 added an option for specifying output files. Revision 3 added an extra output flag for von Neumann correction [7]. Figure 23 shows the way the configuration strings are connected – each arrow represents a dash. For example, 0-0-1-8-4_30_45_86_90-82000000-00001000-00110101 breaks down into Revision 0, Configuration 0 (1D), 1 dimension, size 8, 4 rules (30, 45, 86, 90), 82 million bits of data, output flags, and 00110101 as the seed.

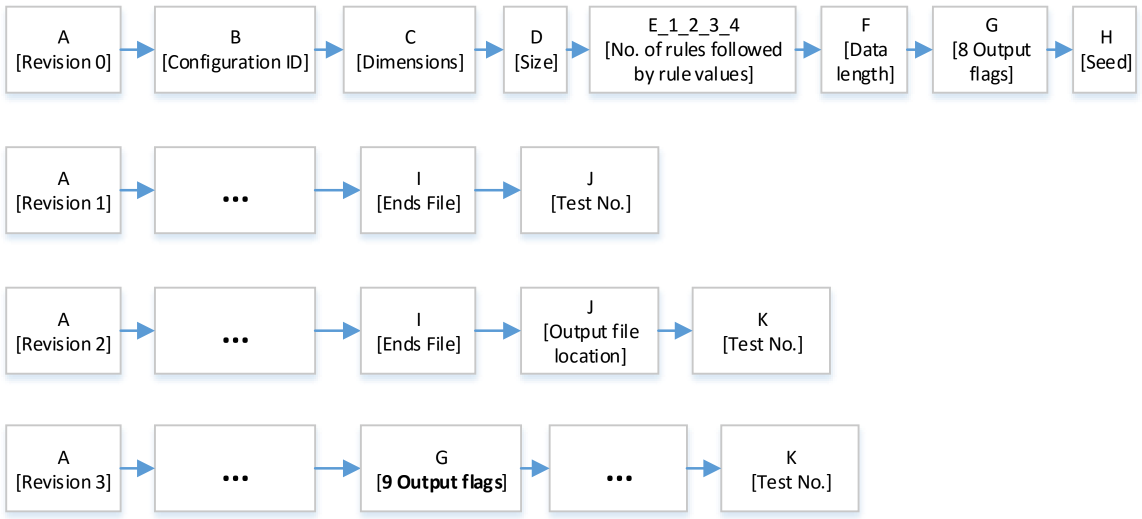


Figure 23 – CA Configuration Strings

SECTION 5.5 – CA Driver

Each section of the driver corresponds to options in the CA configurations (see chapter 2). Two primary data structures were created for the driver – Calculator and DiehardTest, as shown in Figure 24. The Collector (which handles most CA configurations), Cascade, and Pyramid classes all inherit from Calculator, which itself inherits from DiehardTest – as such, every CA calculator is also a tester. Diehard's code was originally packaged in a single file and was relatively easy to integrate into the driver to facilitate testing. Due to its intricate organizational structure, the NIST testing suite was kept separate as a standalone executable. Further development would be needed to integrate NIST directly into the CA driver, mostly for organizational gains rather than functional.

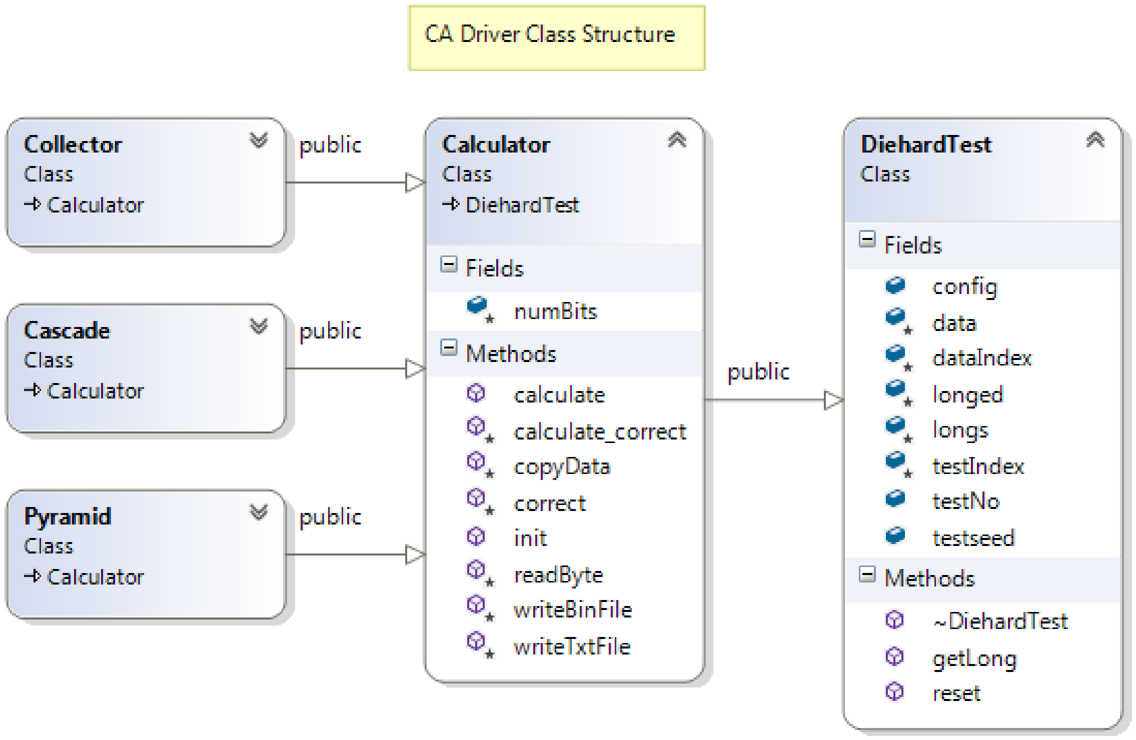


Figure 24 – CA Driver Class Diagram

SECTION 5.6 – Automation

Automated CA testing required careful planning to ensure data integrity and meaningful results. Drawing on options from the original form (renamed CA1), a new automation form (CA2) was created to address the C# to C++ interfacing problem faced with the first iteration – mostly due to string interpretation since characters in C# are 16-bit types and characters in C++ are 8-bit types. The DLL functionality was removed and replaced with direct use of the driver executable. Each instance of the driver can run on a separate thread (virtual core), allowing the automation system to scale with more powerful computing hardware. Figure 25 shows the program flow for multithreading.

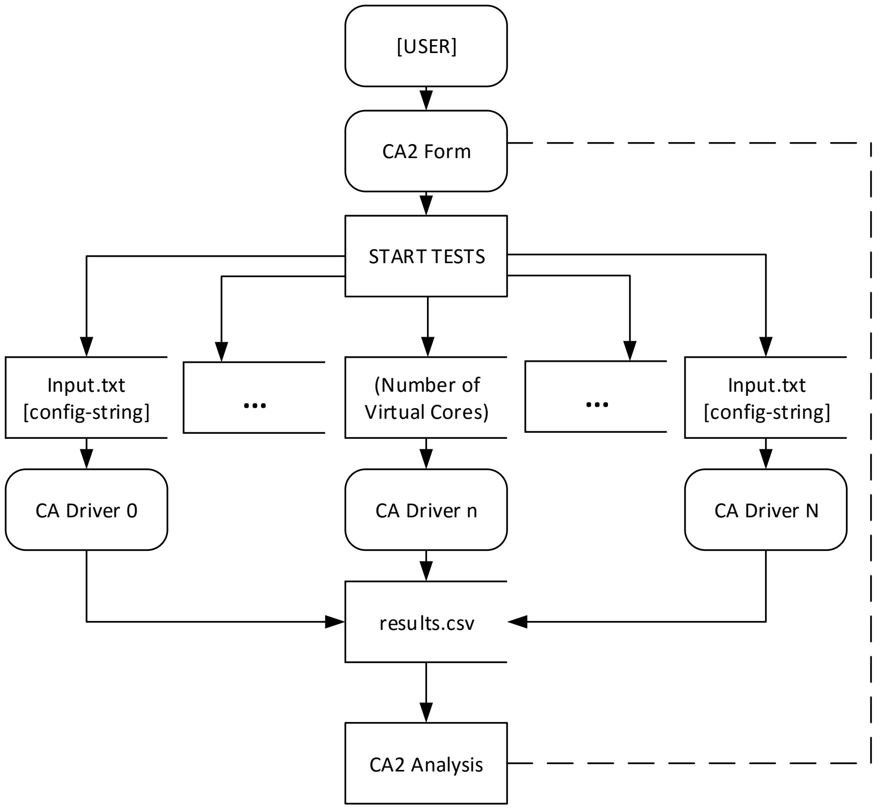


Figure 25 – CA Driver Automation with Multiple Threads (Virtual Cores)

CHAPTER SIX: ECA COMPLETE ANALYSIS

SECTION 6.1 – Observation Criteria

The complete analysis of the ECA spectrum includes all k-space descriptions for each rule, meaning 256 k-space descriptions for each seed size (k). Since these are absolute observations (rather than statistical observations like the p-value results), judgments can be made on the scalability, complexity, feasibility and potential of a complete ruleset. Since all seeds in a k-space end up in the three fundamental categories – GoEs, transients, or cycles – eight observations were made for each k-space description.

- Rule (0 to 255)
- Size k (1 to ∞)
- Number of GoE States
- Number of Transient States
- Number of Cycle States
- Longest Transient Length
- Longest Cycle Length
- Number of Unique Cycles

An important thing to note is that these are based on the interpretation that though the two boundaries are connected, seeds cannot be rotated and remain the same – i.e. 001, 010, and 100 are all treated as totally different states (1, 2, and 4 in decimal, respectively). Under the rule that such seeds are treated as the same state, these observations would change dramatically. See Appendix A for Rule 30, 45, and 106.

SECTION 6.2 – Reverse Algorithm

Since all ECA states are attracted to cycles, the GoE states are likely to be the fastest entry point for classifying a k-space description. Determining if a state is a GoE through conventional brute force is effective but very slow and requires all next states to be known. Rather than calculate an entire block of seeds, the reverse algorithm (see Figure 26) can be used to construct pre-images for each cell and see if it is possible that another state could produce the current state. If not, then the state in question *must* be a GoE. Pre-images for the given rule are created from a lookup table then stored for each cell. To trace a path, the right two cells of each pre-image are compared to the left two cells of the next potential pre-images. If a path can be constructed from start to finish, then a previous state was possible and thus the state in question is not a GoE.

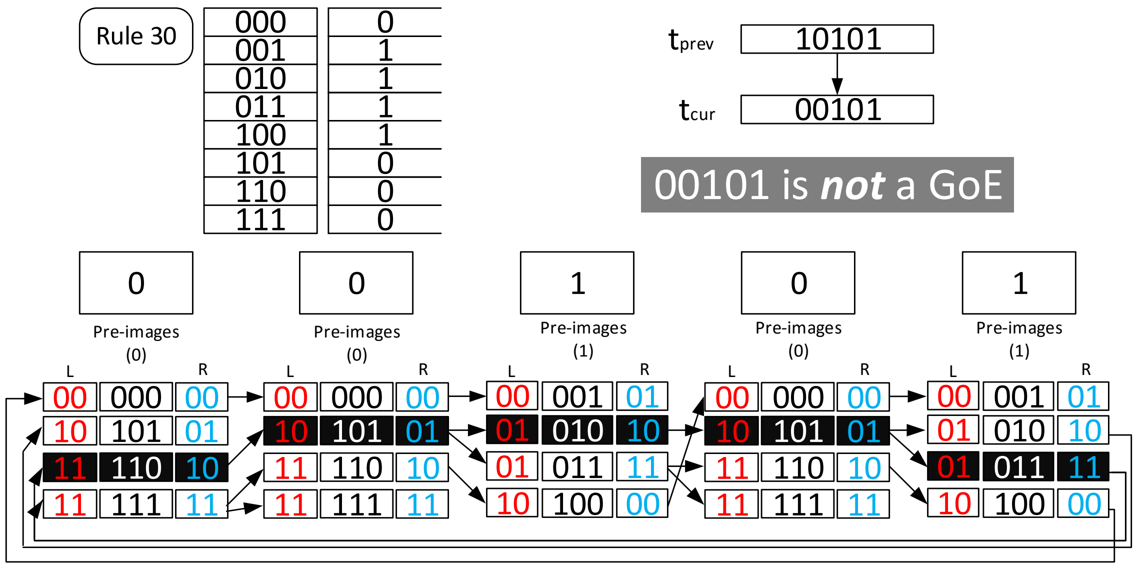


Figure 26 – Reverse Algorithm for ECA

The reverse algorithm is also proof that given enough permutations and computational power, it is possible to reconstruct data that was cycled if the rule and/or rules are known – though the branching will quickly become exponential for larger seed lengths and without an initial timestep (length to current state), ambiguity remains. As such, this potential represents an NP-complete problem for certain rules, in that only by simulating all possible outcomes can the solution be known, and the number of simulations grows dynamically with the length of the CA, meaning no efficient system for narrowing the possibilities is easily derived [8].

SECTION 6.3 – Fundamental Functions

Certain rules exhibit fundamental functions and their outcomes can be easily predicted. Rule 0 forces all next states to be 0 and Rule 255 forces all next states to be 1. Logically, this means the number of GoE states is 2^k – 1 for both rules since only one state (all 0s or all 1s) will cycle to itself – the rest will never fall into the "next state" list. Another example is Rule 204, which is essentially an "identity" rule. Its next state is always equal to B, the current state of the center cell, i.e. itself. Rule 51 is the flipped bit version of 204, meaning the next state is equivalent to the negation of the last. Shift left is Rule 170 and shift right is Rule 240, which essentially take cells C or A in place of the center cell B for the next state. These are summarized in Table 2.

Table 2 – Fundamental Functions Implemented as Rules

Rule	Action	Logic (ABC neighborhood)
240_{10} = $1111\ 0000_2$	Shift right	A
204_{10} = $1100\ 1100_2$	Identity	B
170_{10} = $1010\ 1010_2$	Shift left	C
15_{10} = $0000\ 1111_2$	Negate A, Shift Right	‘A
51_{10} = $0011\ 0011_2$	Negate B	‘B
85_{10} = $0101\ 0101_2$	Negate C, Shift Left	‘C
0_{10} = $0000\ 0000_2$	Logic 0	0
255_{10} = $1111\ 1111_2$	Logic 1	1

SECTION 6.4 – ECA State Diagram Algorithm

With the reverse algorithm, optimizing the ECA generation algorithm is a far less daunting task than brute force interpretation. By generating all the next states and determining GoEs, the search time can be restricted only by the size k. The final algorithm in use is represented in the pseudo-code of Figure 27.

```
Find Cycle Data
   While Cycle Not Found
       Get Next State
       If Next State is Known Transient or Cycle
           Break
       End If
       If Cycle Found
           Record Cycle entry
           Break
       End If
   End While
   Record Transients
   Record Cycle

Generate State Diagram
   For All States in 2^k
       Generate Next State
       Determine if Garden of Eden State (Reverse Algorithm)
           If GoE Found
               Rotate binary string and insert all representations
           End If
   End For
   For All GoE States
       Find Cycle Data
   End For
   For All Non-GoE States
       Find Cycle Data
   End For
   Record Diagram Observations
       Record Number of GoEs
       Record Number of Transients
       Record Number of States that are in Cycles
       Record Longest Cycle
       Record Longest Transient
       Record Number of Unique Cycles
```

Figure 27 – ECA Algorithm in Pseudo Code

SECTION 6.5 – State Diagram Data Storage

To facilitate more compartmentalized operations, the next state generation (i.e. running a given seed through a CA calculator) can be saved to a binary file. For sizes k = 1 to 32, this can be done by storing consecutive 32-bit integers rather than a text file with a string of bytes. So instead of 32 bytes to store a state, it would take 4 bytes. For sizes k = 33 to 64, a 64-bit integer could be used, leaving the size at 8 bytes instead of 64. Assuming the states are stored in the correct order, there is no reason to store the starting state, meaning the size of the file is equal to 2 bytes (one for rule and one for size) + 2^k * 4 bytes. For sizes 1 to 32, to hold all the states of a given rule will take up 32 gigabytes (GB) a piece. So for all 256 rules, the required raw binary storage is 8 terabytes. After they are analyzed, they would no longer be needed and can be deleted. A generator program was written to conform to the structure in Figure 28.

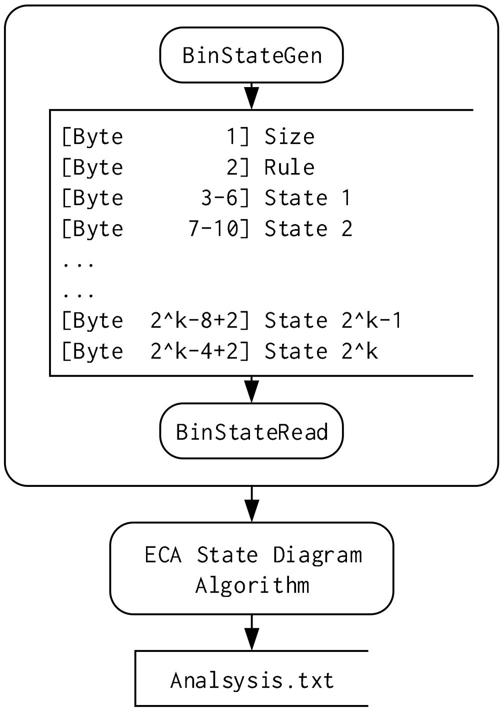


Figure 28 – Binary State Diagram Storage

As the size k grows larger, it becomes increasingly difficult to contain all the working data for the diagram in RAM, and resorting to file storage is a necessary evil as the size k approaches 32. To just hold a working list of all possible 32-bit integers requires 16 gigabytes of RAM. The necessary data sizes are shown in Table 3 and 4. As the algorithm completes, the working list will grow smaller and before the final phase when the GoEs are removed from the transient list, the transient and cycles lists will contain all 2^k states between them, meaning that the minimum target data size is 2^k * 20 bytes. Programming overhead for the data structures will magnify these values even further, and in reality at around k = 27, the C++ implementation begins to take up a majority of 32 GB of RAM. To hold all k = 32 state data would take 80 gigabytes and to hold all k = 64 state data (40 bytes instead of 20) would take 640 exabytes, which actually supersedes the addressing limits of a 64-bit processor (16 EB).

Table 3 – Minimum Data Sizes for State Diagram Storage

Item	Data Types	Data Size (base)
Transients	32-bit state, 32-bit length	8 bytes
Cycles	32-bit state, 32-bit cycle ID	8 bytes
Working List	32-bit state	4 bytes
GoE States	32-bit state	4 bytes

Table 4 – Minimum Data Size for k = 26 to 32, and 64

Size k	Bytes (B)	Kilobytes (KB)	Megabytes (MB)	Gigabytes (GB)
26	1342177280	1310720	1280	1.25
27	2684354560	2621440	2560	2.5
28	5368709120	5242880	5120	5
29	10737418240	10485760	10240	10
30	21474836480	20971520	20480	20
31	42949672960	41943040	40960	40
32	85899345920	83886080	81920	80
64	7.4 * 10^20	7.2 * 10^17	703687441776640	687194767360

SECTION 6.6 – GoE Trends

Globally, the number of GoE states for each rule follows a visible trend as the size (k) is incremented. Shown in Figure 29 is the GoE graph for k = 16. The traits become compressed into the two extremes as the k-space increases, leaving a noticeable gap in the bottom half – a rule will either have very few GoEs or more than half of all potential states will be GoEs.

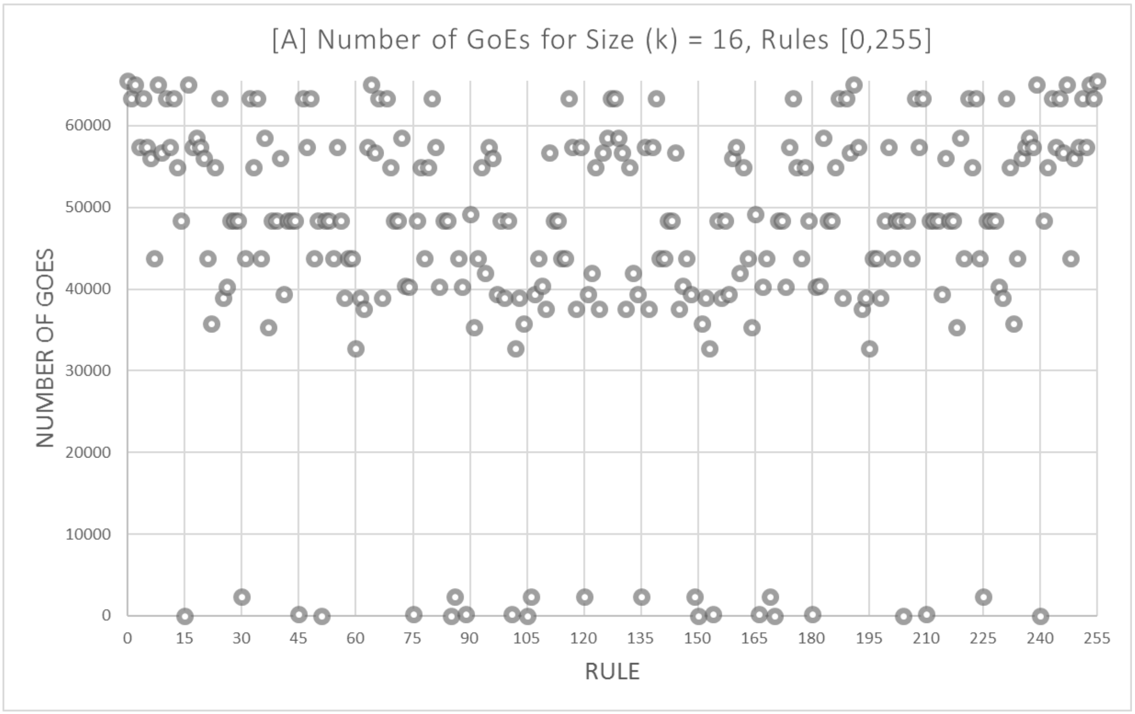


Figure 29 – Graph of GoE Counts, k = 16, All 256 ECA Rules

SECTION 6.7 – Unique Cycle Trends

In part due to the fundamental functions listed in section 5.3, the number of unique cycles usually only has two critical points (identity, 204 and negative identity, 51), as shown in Figure 30, which are always 2^k and $2^k/2$ respectively.

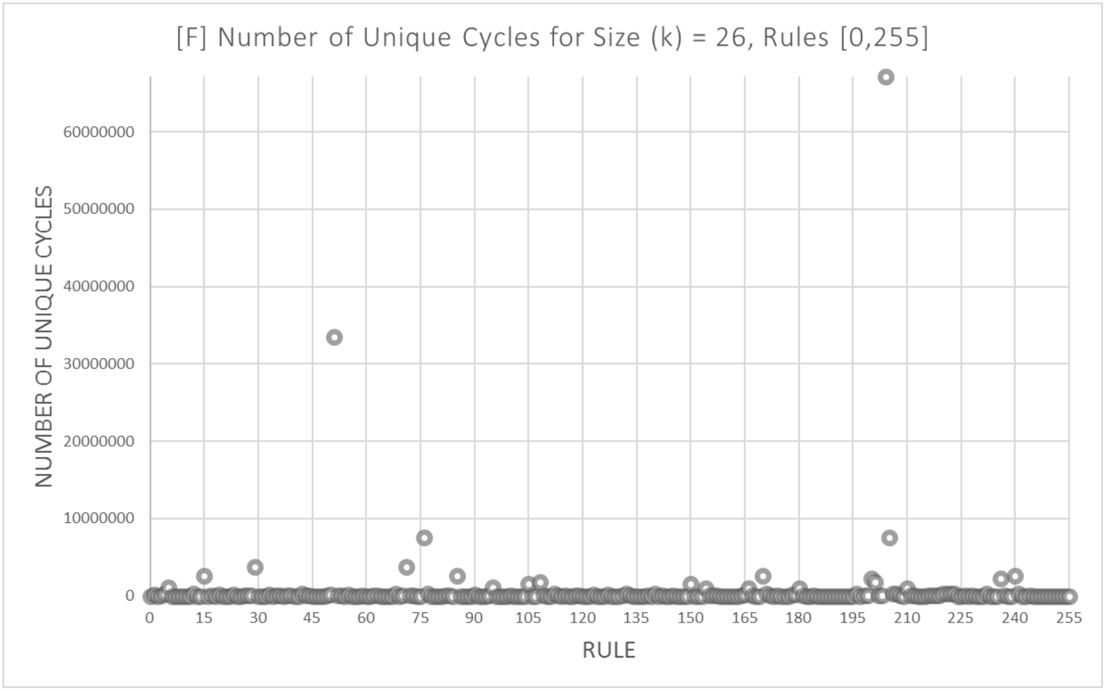


Figure 30 – Graph of Unique Cycles, k = 26, All 256 ECA Rules

SECTION 6.8 – Longest Cycle Length

Rule 45's group (45, 75, 89, and 101) maintain the longest cycle for most k-spaces following 4, with the notable exception of 8 and 16, where Rule 30's group (30, 86, 135, 149) has the longest cycle length. Table 5 shows the longest cycles for k = 1 to 27.

Table 5 – Longest Cycle Length by Rule Group

k	Longest Cycle Group (Rules)	Length	Seeds (2^k)	Length/Seeds
1	Odd 1-127	2	2	100.00%
2	Odd 1-127, 160-163, 168-171, 176-179, 184-187, 224-227, 232-235, 240-243, 248-251	2	4	50.00%
3	14-15, 26-27, 38-39, 52-53, 82-85, 142-143, 154-155, 166-167, 180-181, 210-213	6	8	75.00%
4	3, 17, 27, 30, 35, 39, 49, 53, 58-59, 62-63, 83, 86, 114-115, 118-119, 131, 135, 145, 149, 163, 177	8	16	50.00%
5	45, 75, 89, 101	30	32	93.75%
6	45, 75, 89, 101	18	64	28.13%
7	45, 75, 89, 101	126	128	98.44%
8	30, 86, 135, 149	40	256	15.63%
9	45, 75, 89, 101	504	512	98.44%
10	45, 75, 89, 101	430	1024	41.99%
11	45, 75, 89, 101	979	2048	47.80%
12	45, 75, 89, 101	240	4096	5.86%
13	45, 75, 89, 101	1105	8192	13.49%
14	45, 75, 89, 101	2198	16384	13.42%
15	45, 75, 89, 101	6820	32768	20.81%
16	30, 86, 135, 149	6016	65536	9.18%
17	45, 75, 89, 101	78812	131072	60.13%
18	45, 75, 89, 101	7812	262144	2.98%
19	45, 75, 89, 101	183920	524288	35.08%
20	45, 75, 89, 101	142580	1048576	13.60%
21	45, 75, 89, 101	352884	2097152	16.83%
22	45, 75, 89, 101	122870	4194304	2.93%
23	45, 75, 89, 101	3459591	8388608	41.24%
24	45, 75, 89, 101	421188	16777216	2.51%
25	45, 75, 89, 101	10828525	33554432	32.27%
26	45, 75, 89, 101	334308	67108864	0.50%
27	45, 75, 89, 101	81688176	134217728	60.86%

Despite Rule 45's group being the longest cycle on most even sized k's, the length to total number of states ratio is very low. Figure 31 shows the change between sizes on a logarithmic scale, showing the dips between even and odd while maintaining the obvious trend toward exponential growth.

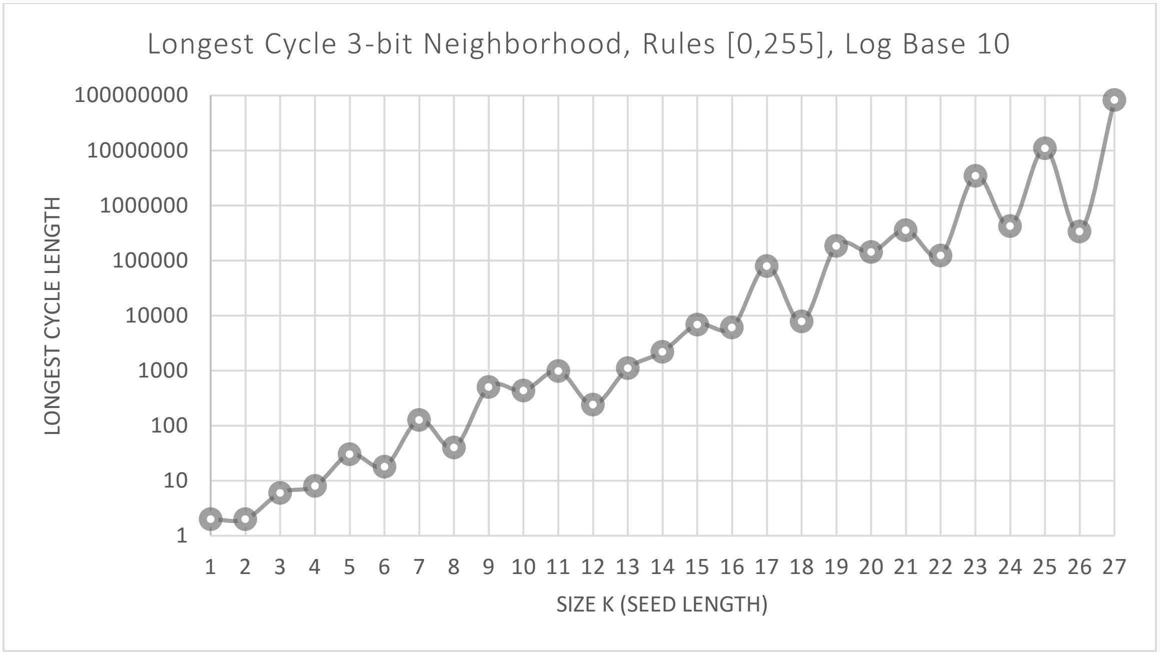


Figure 31 – Longest Cycle Length Graph, Logarithmic Scale, Base 10

Long cycles are important to consider since they will not repeat data in a critical period of reference, allowing for more random number generation through chaotic propogation. Rule 30 and Rule 45 become important players due primarily to the fact that they hold the longest cycles.

CHAPTER SEVEN: TESTING RESULTS

SECTION 7.1 – Balanced ECA Rules

A rule is balanced when it has an equivalent number of preimages for both 0 and 1, but not every balanced rule lends to random number generation (chaotic propogation). Figure 32 shows 70 rules in the basic 256, which have an equal number of 0s and 1s. Diehard tests were ran for 100 random seeds on 128, 256, 512, 1024, 2048, and 4096-bit seed lengths to test against what rules are viable. In total, this comprises 153,600 tests (7 seconds per test). Some have obvious results (0 and 255, for example). For comparison, there are 601,080,390 balanced rules in the 4,294,967,296 rules of the 5-bit neighborhood, and to test all rules in the 5-bit space would require a large computing network to finish in a reasonable period of time.

```
15 23 27 29 30 39 43 45 46 51 53 54 57 58 60 71 75 77 78 83 85 86

89 90 92 99 101 102 105 106 108 113 114 116 120 135 139 141 142 147

149 150 153 154 156 163 165 166 169 170 172 177 178 180 184 195 197

198 201 202 204 209 210 212 216 225 226 228 232 240
```

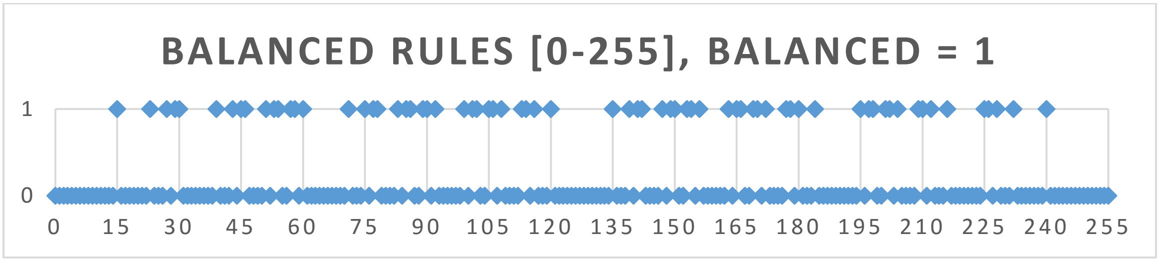


Figure 32 – Balanced Rules, ECA 256

SECTION 7.2 – ECA Tests

The following sections show the results analysis of the ECA tests ran for all rules with a size k of 128, 256, 512, 1024, 2048, and 4096 (see Figure 33). One hundred random seeds were used, with the same set of seeds used in each size category. The y-axis data shows 100 points for each rule (corresponding to a test with a seed), with a percentage value of how many p-values were within the alpha of 0.01 out of the 230 Diehard results. For the purposes of evaluation, one could say that Rule 0's test results all had a 0% chance of being random. As the passing results draw closer to 100%, the results must be taken with the understanding that no result is 100% random, but for testing purposes appears random. Good candidates for random number generation are those that approach or exceed 90% passing. Data analysis seems to follow the hypothesis that a subset of balanced rules is most likely to produce more chaotic (random) behavior when analyzing the raw data from continuous CA calculations for PRNGs.

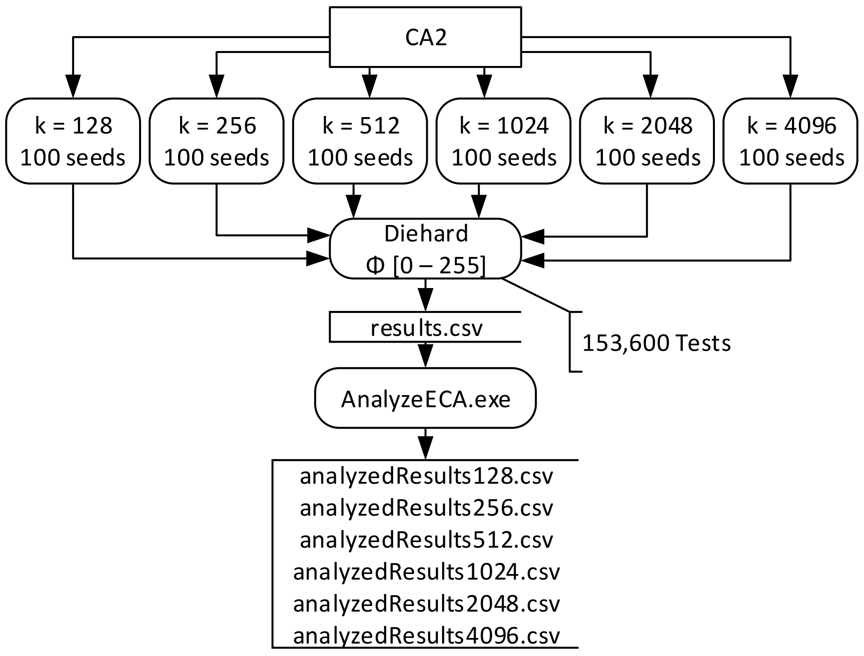


Figure 33 – ECA Testing and Analysis

SECTION 7.2.1 – k = 128, Diehard Results

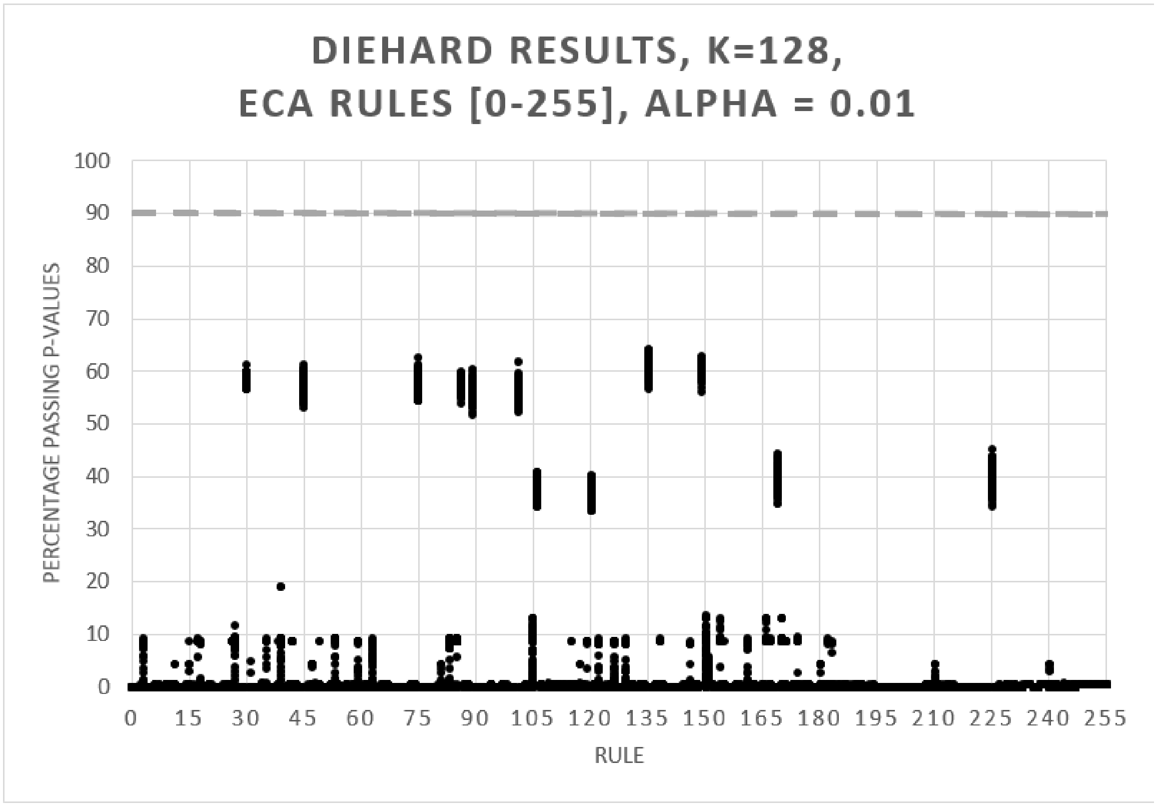


Figure 34 – Diehard Results, ECA k = 128, Alpha = 0.01, All Rules, 25600 Tests

Figure 34 shows no result for k = 128 exceeds 90% passing when an alpha of 0.01 is used in the evaluation phase of Diehard testing. A simple conclusion that can be made on this data is that strictly harvesting the ECA data from a random 128 length starting seed (aka, a 128-bit key) is not enough to pass random testing, especially for the purposes of cryptography. However, there are 12 rules that stand out for further examination. From left to right, they are 30, 45, 75, 86, 89, 101, 106, 120, 135, 149, 169, and 225. This behavior is expected, since they fall into three rules clusters, two of which have the longest cycles at the k < 32 level. These are the [30, 86, 135, 149], [45, 75, 89, 101], and [106, 120, 169, 225] groups. Rule 30 and Rule 45's clusters lead the k = 128 results.

SECTION 7.2.2 – k = 256, Diehard Results

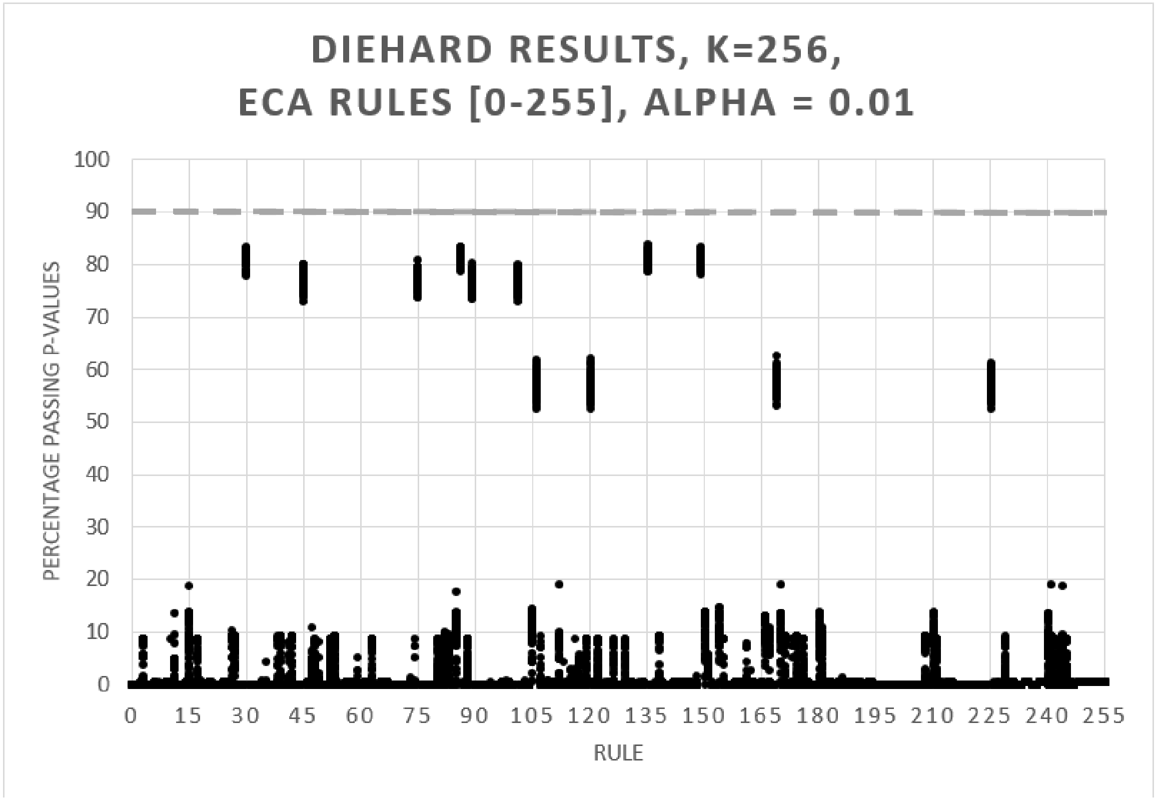


Figure 35 – Diehard Results, ECA k = 256, Alpha = 0.01, All Rules, 25600 Tests

Figure 35 shows that for k = 256, the same groups from k = 128 lead the results, but still fall short of the 90% mark. Rule 30's group [30, 86, 135, 149] takes the lead, averaging at the 80% mark while Rule 45's group [45, 75, 89, 101] averages at the 75% mark. Rule 106's group [106, 120, 169, 225] has risen to the 55% mark. Compared to AES-256 and other 256-bit secure algorithms, raw CA harvesting with these groups is not a viable option for cryptography at this level.

SECTION 7.2.3 – k = 512, Diehard Results

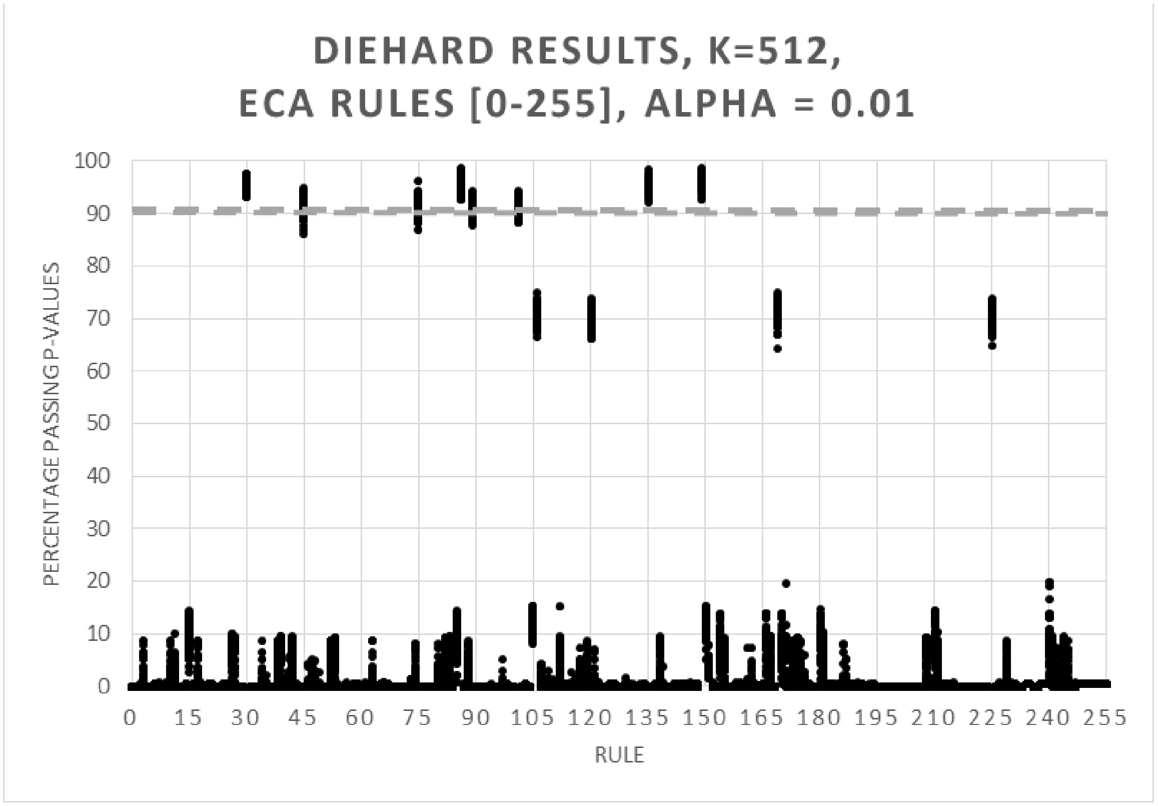


Figure 36 – Diehard Results, ECA k = 512, Alpha = 0.01, All Rules, 25600 Tests

Figure 36 shows at k = 512, the core groups for Rule 30 [30, 86, 135, 149] and Rule 45 [45, 75, 89, 101] could be safely called random and potentially used in a 512-bit cryptographic system when using their raw output. It is likely that the third group, Rule 106 [106, 120, 169, 225] will perform at greater bit levels since it averaged at 70% at this level. A clear lead is still given to Rule 30 over Rule 45, but this is likely because Rule 30 has a greater cycle than Rule 45's group at size k with powers of 2.

SECTION 7.2.4 – k = 1024, Diehard Results

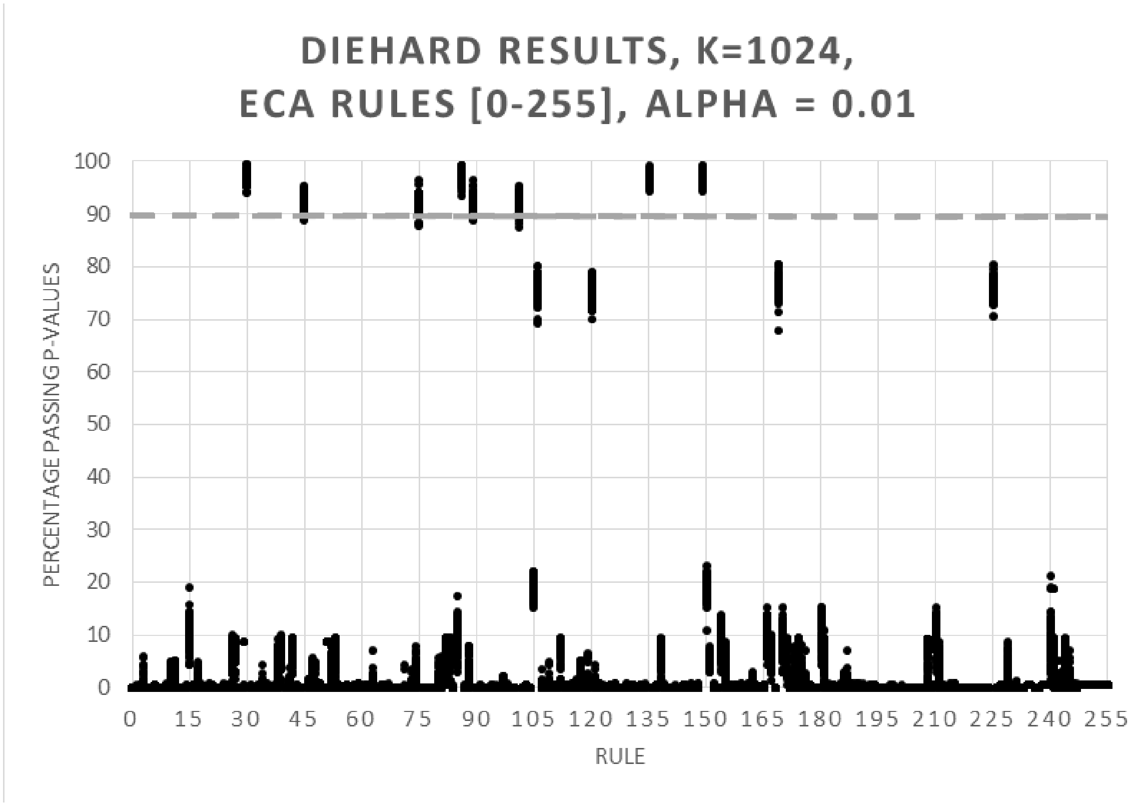


Figure 37 – Diehard Results, ECA k = 1024, Alpha = 0.01, All Rules, 25600 Tests

Figure 37 shows that Rule 30 and Rule 45's groups maintain their status while Rule 106's group pulls up to 75% average passing at k = 1024, still not suitable for random number generation at this level. Rule 105 and 150 (each are negations of each other) begin to show a break from their low numbers, pushing up to 20% average passing.

SECTION 7.2.5 – k = 2048, Diehard Results

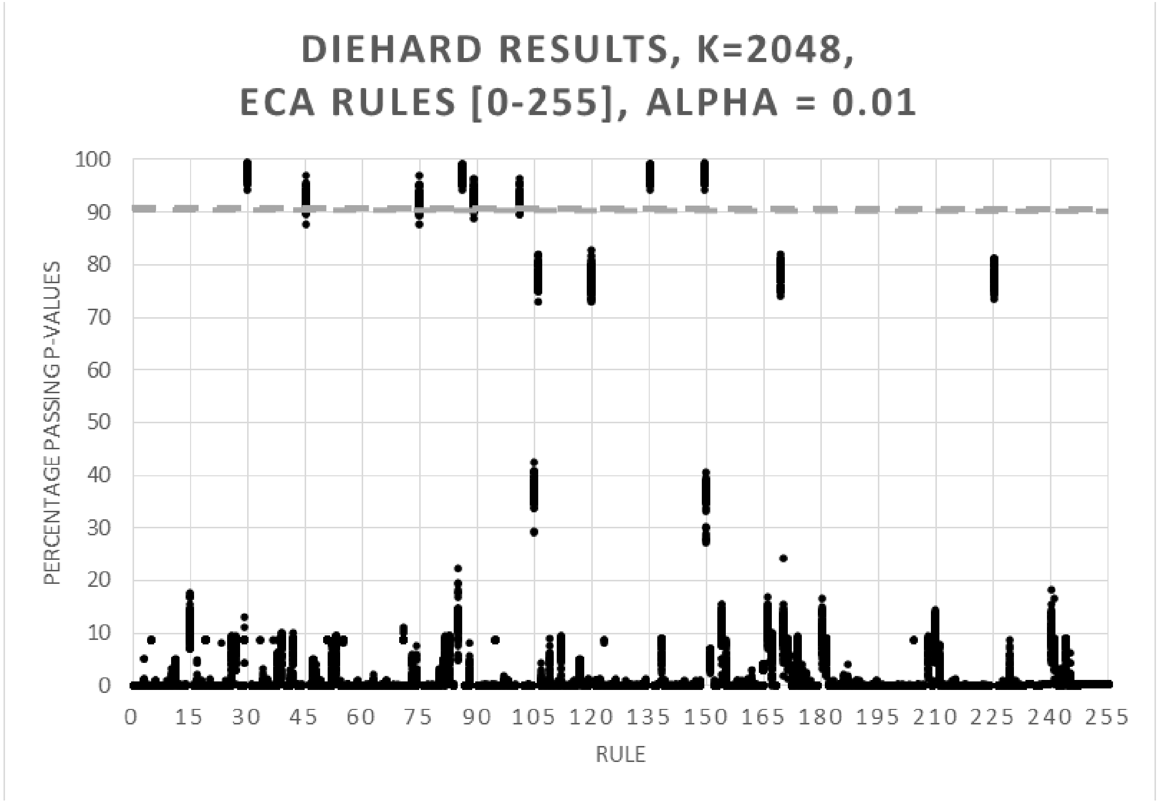


Figure 38 – Diehard Results, ECA k = 2048, Alpha = 0.01, All Rules, 25600 Tests

Figure 38 shows that Rule 30 and 45 seem to have plateaued at their respective levels, while Rule 106 inches up by 1-2% on average. Further extensions of the key size (k) will likely push the 106 group into the 90% level. Rule 105 and 150 have moved up to 35% average from their small lead at 20% in k = 1024. Tests for the remaining rules are still necessary since they provide context for the relative randomness among rules. In addition, it provides proof that simply extending the k-length is not the only catalyst for randomization, otherwise all rules would be increasing at a regular rate.

SECTION 7.2.6 – k = 4096, Diehard Results

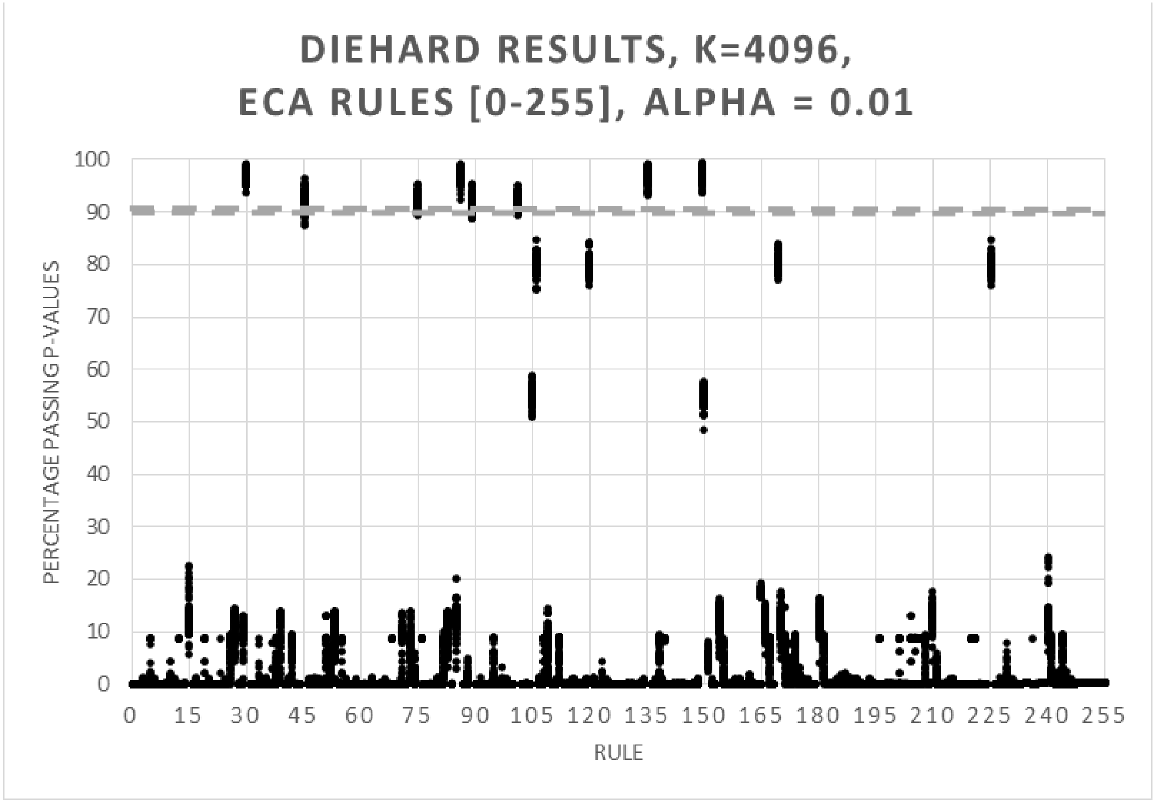


Figure 39 – Diehard Results, ECA k = 4096, Alpha = 0.01, All Rules, 25600 Tests

Figure 39 shows the results of the final round of tests, where Rule 30 and 45 maintain their plateau while the Rule 106 group continues to crawl toward the 90% mark. Rule 105 and 150 have pushed into the 55% average passing range, an increase of 20% over the 2048 range. 4096 bit keys are a common upper limit and therefore any system utilizing raw CA calculations should only be used with Rule 30 and 45 at this level.

SECTION 7.3 – Cascade Tests

A large batch of Diehard tests were ran using a seed block of k = 16, t = 32, with 10 random seeds on all combinations of two rules (i.e. 256 x 256 x 10 = 655,360 tests, at an average of roughly 10 seconds each), the results flow shown in Figure 40. Testing took ten days on an 8-thread processor running non-stop. Two graphs are represented in the following sections – one with the number of passing p-values (a scalar representing randomness density) and the second with Diehard tests that had a total number of p-values above the 90% threshold. Each chart is setup as a bubble chart, with the area of each bubble relative to its total at that x-y point pair. There are 65,536 points on each graph, though many have zero passing values and thus no bubble is shown. Cascade's configuration is proof that even with a 16-bit key, random number generation is still a decent possibility with CA.

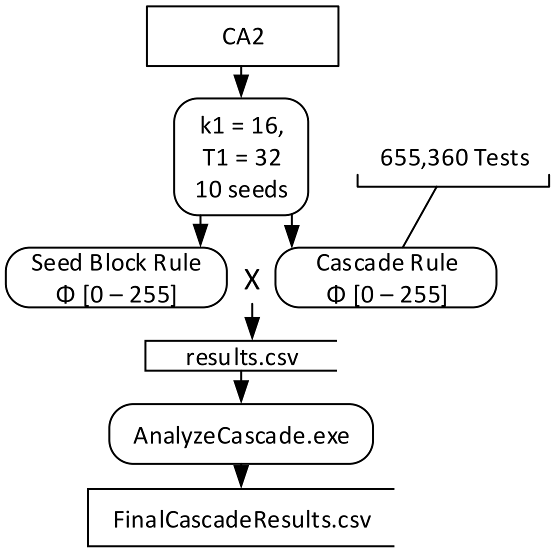


Figure 40 – CA2 Process for Testing Cascade Combinations

SECTION 7.3.1 – k1=16, t1=32, Passing p-values Density

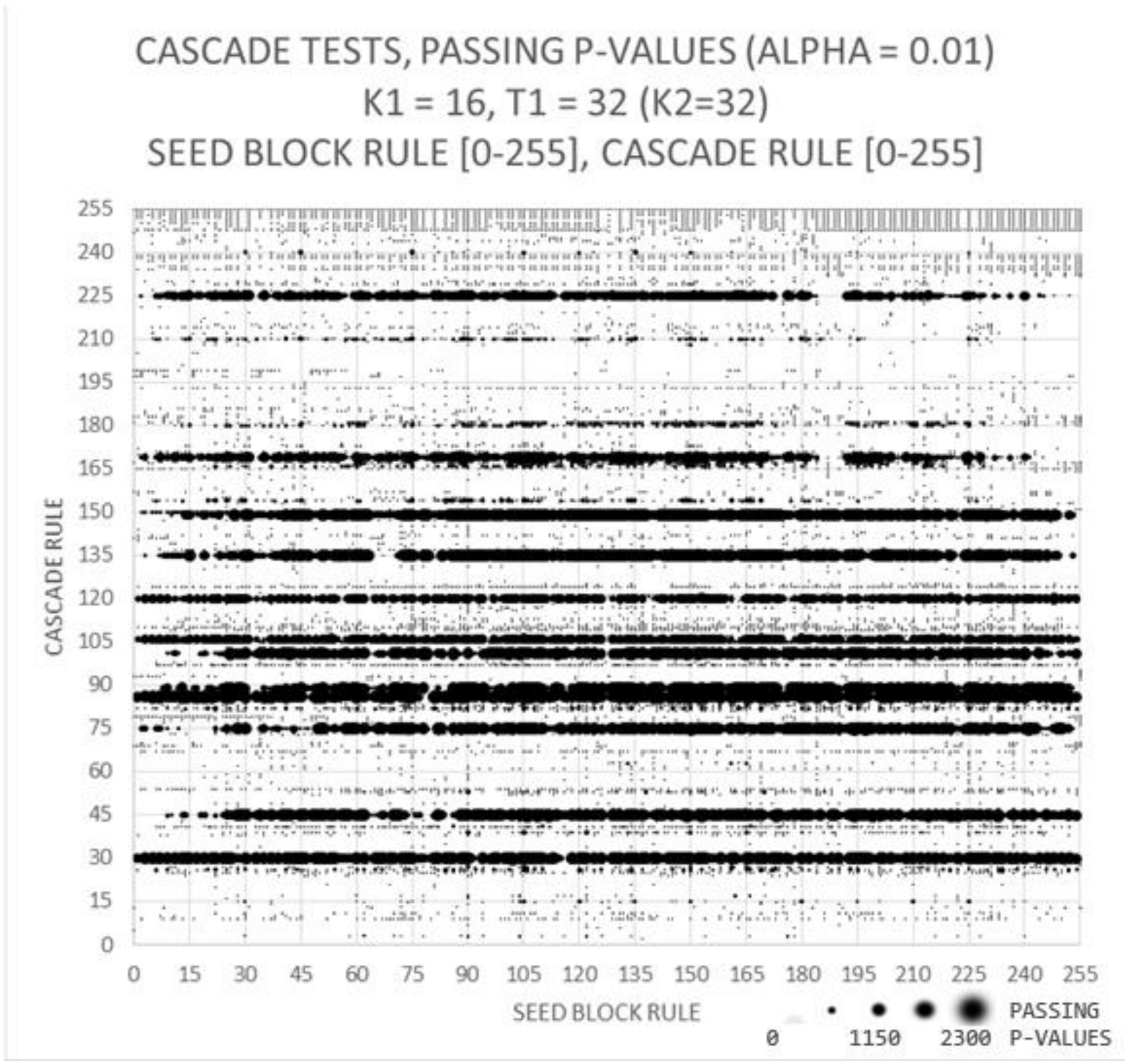


Figure 41 – Cascade Tests, Passing P-values, k1 = 16, t1 = 32

While the passing p-value density does not show the number of passing tests, it gives clear evidence that a weak seed block rule can be overrun by a stronger cascade rule. Rule 30 and 86 seem to perform the strongest regardless of their starting seed. Rule 45 and its usual gamut of rules also fair very well. Figure 41's graph correlates with the results from the ECA testing, representing the major trends.

SECTION 7.3.2 – k1=16, t1=32, Passing Tests Density

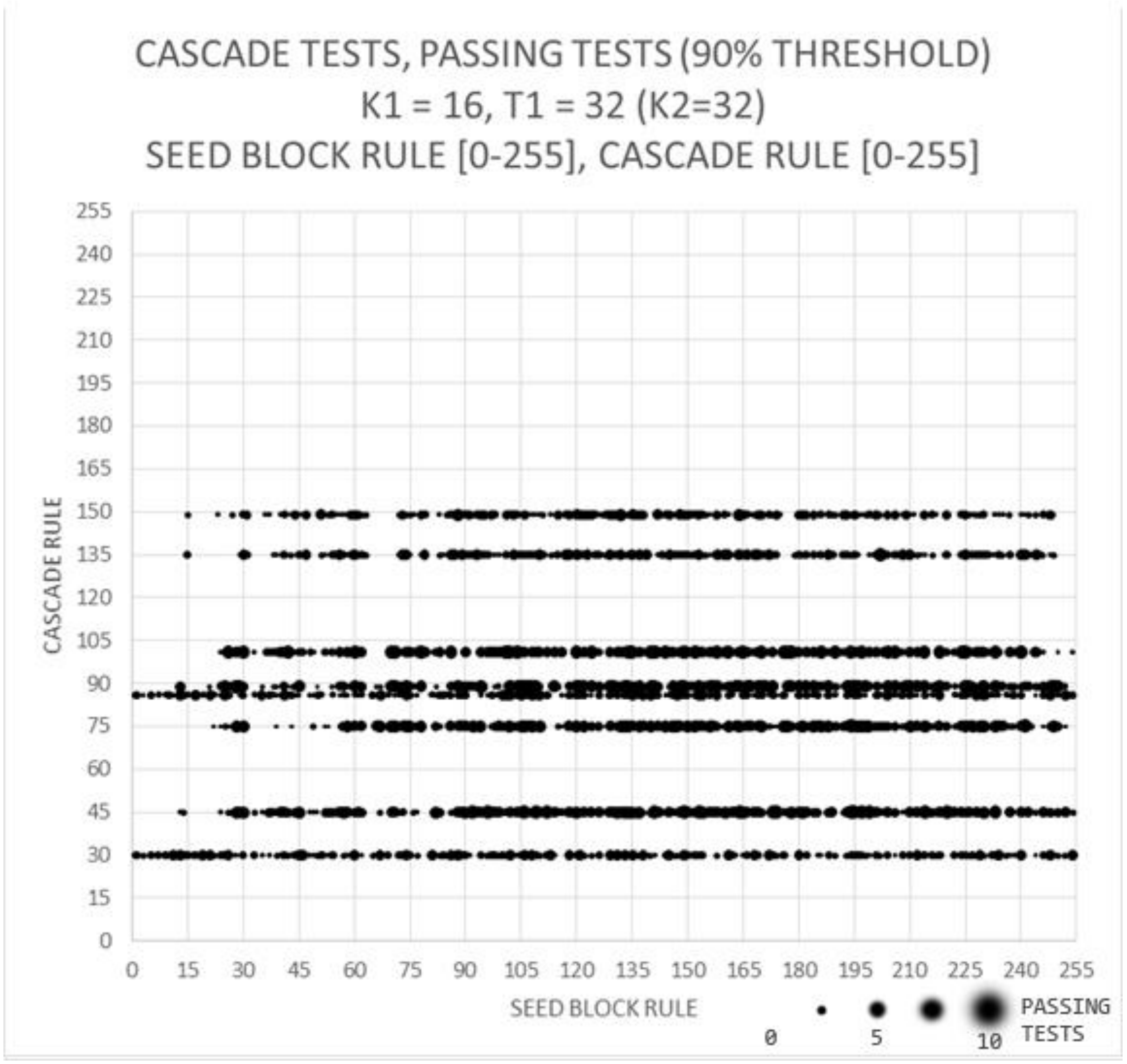


Figure 42 – Cascade Tests, Passing Diehard Tests, k1 = 16, t1 = 32

Figure 42 shows the density of Diehard tests that actually passed, ranging from 0 to 10 at each point, with larger bubbles carrying greater weight. Despite its large density, Rule 30 does not do as well as 45 here. In the raw data, Rule 45's group [45,75,89,101] are the only ones with 8 to 10 total passes of the 10 seeds tested. Despite 45's advantage, Rule 30 and 86 both have relatively equal distribution of passing tests across all seed block rules, and will likely pass at higher key lengths.

CHAPTER EIGHT: CONCLUSION

SECTION 8.1 – Summary

Elementary Cellular Automata are deterministic structures created by iterating through each cell and calculating its next state from predetermined rules. Configurations and augmentations of ECA are wide open interpretations of basic ECA rules and applications, leading to different lattices, rules, and circumstantial use. An encryption methodology for CA such as CASE can be modeled for practical use on modern and future equipment. Random number testing on ECA is accomplished via the Diehard utility or the statistical testing suite provided by the U.S. National Institute of Standards and Technology. P-values generated from those tests are statistical indicators of pseudo-randomness, since nothing can "truly" be random. Coding and testing CA is accomplished through the use of a driver and workbook program combination, allowing for automation, code reuse and flexibility. Complete analysis of ECA can help with logistically correlating testing results with their respective attributes. Rigorous testing allows systematic evaluation of certain rules and configurations, leading to judgments on practicality of pseudo-random number generation and ultimately encryption.

SECTION 8.2 – Conclusions

In conjunction with ECA complete analysis and random number testing, results clearly show that pseudo random number generation is possible at high levels for basic ECA and that some rules are categorically more chaotic than others. Evaluation and concentrated testing on the Cascade configuration shows that it lends greatly to parallel encryption, since it is capable of producing random results with minimal input.

SECTION 8.2.1 – Focus 1: ECA Chaotic Rules and Configurations

1. What rules and configurations are conducive to chaotic structure?

Rules 30 and 45 are the most likely candidates for chaotic propogation in ECA, regardless of configuration. Rules 105 and 106 show promise at higher complexity, warranting further analysis. Dr. Salman's twister (or cascade) structure provides a minimal augmentation of ECA that is able to generate chaotic results with small seed lengths, lending greatly to practical use.

SECTION 8.2.2 – Focus 2: ECA Random Number Generators

2. Can ECA be used for practical random number generation?

Cellular Automata are capable of producing pseudo-random results with minimal input. Concentrated research and extended proofs would be necessary to further this use of CA without making too many assumptions on the infallibility of its simplicity. Rigorous testing shows the practical level of RNG for each rule, correlating the state diagram analysis and affirming that the Rule 30 and 45 groups are most likely to be random.

SECTION 8.2.3 – Focus 3: ECA Encryption

3. If RNG creation is possible, how much complexity is necessary for encryption?

Cascade configurations allow for parallelized encryption by providing a source block of seeds (or keys) for child vectors. The CASE methodology should be flexible enough to provide a basis for a true encryption standard with CA at its core – preferably rule groups 30 or 45. Since basic ECA is NP-complete it requires all intermediate steps, meaning that as computing power increases, the critical time for the child vector propogation can be adapted depending on use. As with AES and other symmetric encryption, hardware and software implementations can coexist, provided rigorous validation and standardization are regulated.

SECTION 8.3 – Recommendations

The sheer simplicity of cellular automata is reason enough not to ignore its use, either for encryption or any other purpose. Its exponential scalability allows it to adapt as computing power advances, provided no proof is found to compute timesteps without intermediate results. Matthew Cook has provided a proof that Rule 110 is Turing complete, meaning it is useful for universal computation [9]. As process architecture shrinks, this could allow for a simplified deterministic computing structure without the possibility of clock failure, with the potential of encryption (by switching the base rule) built naturally into the circuits. Whether or not CA comes to be used in the real world, it seems to be an untapped resource and its future cannot be ignored.

SECTION 8.3.1 – Future Work on Analysis

Complete analysis of rules and increasing seed lengths requires exponentially greater amounts of memory for diminishing gains in data. Preliminary investigation seems to indicate that the Rule 30 and 45 groups are the most viable sets. Suggested avenues would include investigating the analysis of spatial/temporal rules to see if there is merit in combining certain selections over using flat rules for all timestep calculations. A new batch of tests could then be run to determine if PNRGs are possible at lower key lengths than 512, making ECA more viable.

SECTION 8.3.2 – Future Work on Encryption

Expanding the CASE model is the next logical step in utilizing CA for encryption, which would include investigating its weaknesses and potentially rewriting the premise entirely if such weaknesses could not be overcome. An alternative would be to develop another method of parallelizing the data output. Serial encryption is not necessarily efficient when large amounts of data is involved, but on small scale systems it may not be as much of a concern, therefore any configuration can do, provided it gives enough randomization with more minimal inputs than generic ECA.

SECTION 8.3.3 – Self-Replication and Validation with CA

John von Neumann’s original research into CA was for the purpose of creating machines that continuously replicate themselves. Nanotechnology is rapidly advancing in all fields of industry via microarchitecture shifts, bioscience procedures, or more mundane mechanics. While not a perfect fit, combining CA rules into a nano-environment could easily allow for self-validation in testing and a multitude of other potentials, if not one day the creation of von Neumann's machines.

SECTION 8.3.4 – Bioscience Research with CA

Natural rules govern cell replication, mitosis, creation, and general repair. CA rules are not perfectly analogous to chemistry, but in some cases can be close enough to warrant model creations to aid in reducing computer simulation time. Protein folding is one avenue of distributed computer research that could benefit if it is not utilizing rough models already. [10] DNA networks are also a potential for CA, since genomes and their nitrogen bases have many similarities to the way ECA form around attractor cycles.

REFERENCES

[1] J. V. Neumann, The Theory of Self-Reproducing Automata, Urbana and London: University of Illinois Press, 1966.

[2] S. Wolfram, "Random Sequence Generation by Cellular Automata," in *Advances in Applied Mathematics 7 No. 2*, 1986, pp. 123-169.

[3] K. Salman, "Elementary Cellular Automata (ECA) Research platform," *Journal of Selected Areas in Software Engineering (JSSE)*, vol. 3, no. 6, 2013.

[4] G. Marsaglia, "The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness," Florida State University, 1995. [Online]. Available: http://stat.fsu.edu/pub/diehard/.

[5] National Institute of Standards and Technology, "NIST Statistical Test Suite," 16 July 2014. [Online]. Available: http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html.

[6] "Hypothesis Testing (P-value approach)," 2015. [Online]. Available: https://onlinecourses.science.psu.edu/statprogram/node/138.

[7] D. Naccache, "von Neumann Correction," in *Encyclopedia of Cryptogtraphy and Security*, Springer US, 2011, p. 1364.

[8] F. Green, "NP-Complete Problems in Cellular Automata," *Complex Systems,* pp. 453-474, 1987.

[9] M. Cook, "Universality in Elementary Cellular Automata," *Complex Systems,* vol. 15, pp. 1-40, 2004.

[10] "Folding@Home," Stanford University, [Online]. Available: https://folding.stanford.edu/home/the-science/. [Accessed 23 1 2015].

APPENDICES

APPENDIX A: STATE DIAGRAM CYCLE DATA

SECTION A.1 – Complete ECA Analysis, Rules 30, 45, 106

Table 6 – Rule 30 [30, 86, 135, 149] State Diagram Data, k = 1 to 27

Size	GoEs	Transient States	Cycle States	Longest Transient	Longest Cycle	Unique Cycles
1	1	0	1	1	1	1
2	1	0	3	1	1	3
3	3	4	1	3	1	1
4	5	0	11	1	8	4
5	6	20	6	5	5	2
6	12	49	3	10	1	3
7	22	14	92	2	63	9
8	33	172	51	16	40	5
9	57	211	244	13	171	3
10	101	885	38	46	15	6
11	166	1540	342	55	154	13
12	280	3385	431	126	102	12
13	482	6279	1431	66	832	5
14	813	13489	2082	127	1428	18
15	1373	29619	1776	321	1455	31
16	2337	52908	10291	287	6016	9
17	3962	113305	13805	362	10846	7
18	6708	251086	4350	1137	2844	18
19	11382	508782	4124	1234	3705	5
20	19305	995345	33926	1736	6150	27
21	32721	2047812	16619	4308	2793	60
22	55485	4123075	15744	5477	3553	24
23	94094	8251342	43172	6051	38249	4
24	159536	16422689	194991	9568	185040	49
25	270506	32600370	683556	9252	588425	16
26	458693	66264744	385427	18758	312156	33
27	777765	132913219	526744	34054	240300	40

Table 7 – Rule 45 [45, 75, 89, 101] State Diagram Data, k = 1 to 27

Size	GoEs	Transient States	Cycle States	Longest Transient	Longest Cycle	Unique Cycles
1	0	0	2	0	2	1
2	2	0	2	1	2	1
3	0	0	8	0	3	5
4	4	10	2	4	2	1
5	0	0	32	0	30	2
6	8	30	26	6	18	6
7	0	0	128	0	126	2
8	16	134	106	18	32	7
9	0	0	512	0	504	6
10	32	260	732	16	430	9
11	0	0	2048	0	979	16
12	64	3094	938	167	240	33
13	0	0	8192	0	1105	24
14	128	8988	7268	312	2198	16
15	0	0	32768	0	6820	54
16	256	53014	12266	1776	2816	42
17	0	0	131072	0	78812	22
18	512	220728	40904	3533	7812	201
19	0	0	524288	0	183920	36
20	1024	771640	275912	3678	142580	282
21	0	0	2097152	0	352884	262
22	2048	3865994	326262	21950	122870	272
23	0	0	8388608	0	3459591	224
24	4096	14904662	1868458	53104	421188	4411
25	0	0	33554432	0	10828525	514
26	8192	66213056	887616	352642	334308	1353
27	0	0	134217728	0	81688176	3134

Table 8 – Rule 106 [106, 120, 169, 225] State Diagram Data, k = 1 to 27

Size	GoEs	Transient States	Cycle States	Longest Transient	Longest Cycle	Unique Cycles
1	1	0	1	1	1	1
2	1	0	3	1	2	2
3	3	1	4	2	3	2
4	5	0	11	1	4	5
5	6	0	26	1	15	4
6	12	34	18	8	6	5
7	22	28	78	5	49	6
8	33	52	171	6	15	18
9	57	208	247	12	54	16
10	101	375	548	17	205	18
11	166	1507	375	73	176	20
12	280	3094	722	85	168	37
13	482	6071	1639	76	416	46
14	813	12089	3482	153	448	72
15	1373	23806	7589	191	1095	108
16	2337	55492	7707	457	2688	155
17	3962	116756	10354	938	3230	214
18	6708	234871	20565	1155	2664	357
19	11382	480586	32320	1233	13471	501
20	19305	960275	68996	1063	21240	782
21	32721	1995596	68835	3506	14658	1184
22	55485	4049210	89609	5030	32428	1818
23	94094	8200190	94324	10024	14306	2792
24	159536	16061462	556218	6997	80544	4728
25	270506	32549375	734551	11269	309150	6729
26	458693	66315704	334467	26587	26858	10482
27	777765	132053401	ERROR*	30637	242352	16317

*A 32-bit rollover error occurred when computing the number of unique cycles for this group, resulting in loss of data for that category.

SECTION A.2 – Longest Cycle Graphs, k = 8, 16, 24, 27

Figures 43 through 46 show the longest cycles for all rules in sizes k = 8, 16, 24, and 27. For 8 and 16, Rule 30 holds the longest cycle, and will likely continue this trend as the number doubles, since Rule 45 is still greater at k = 24. Even-numbered sizes produce lower cycle lengths for Rule 45, but with odd-numbered k's it dominates the graph.

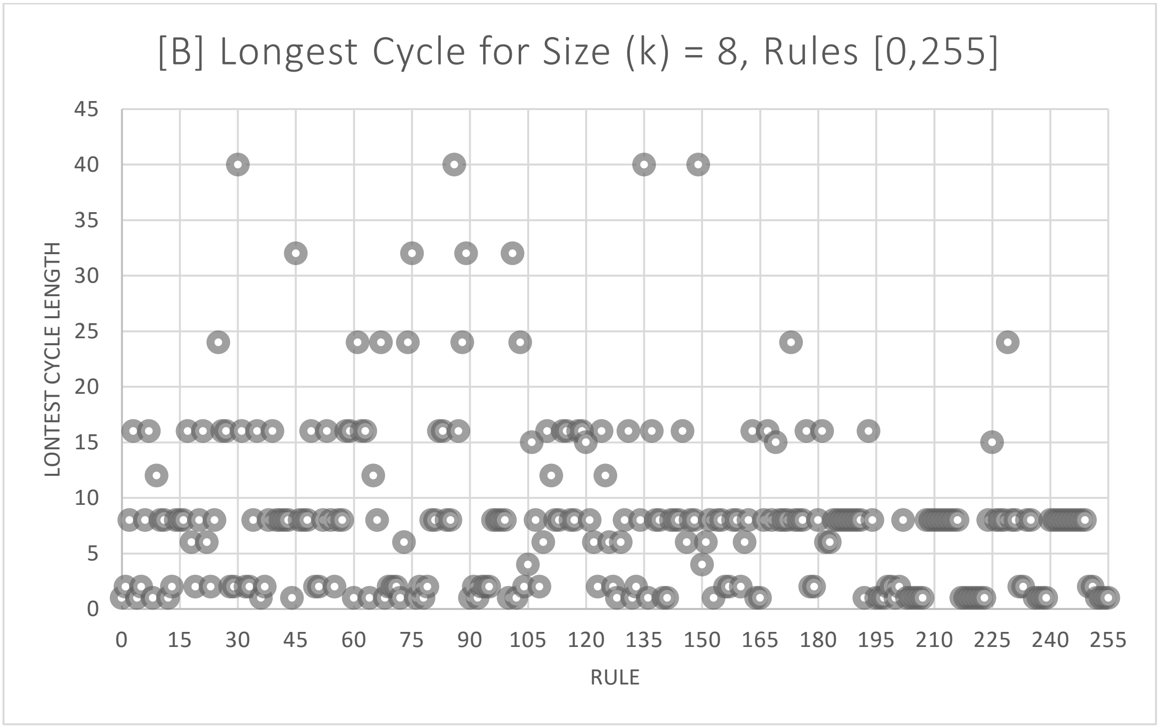


Figure 43 – Longest Cycle Diagram, All Rules, k = 8

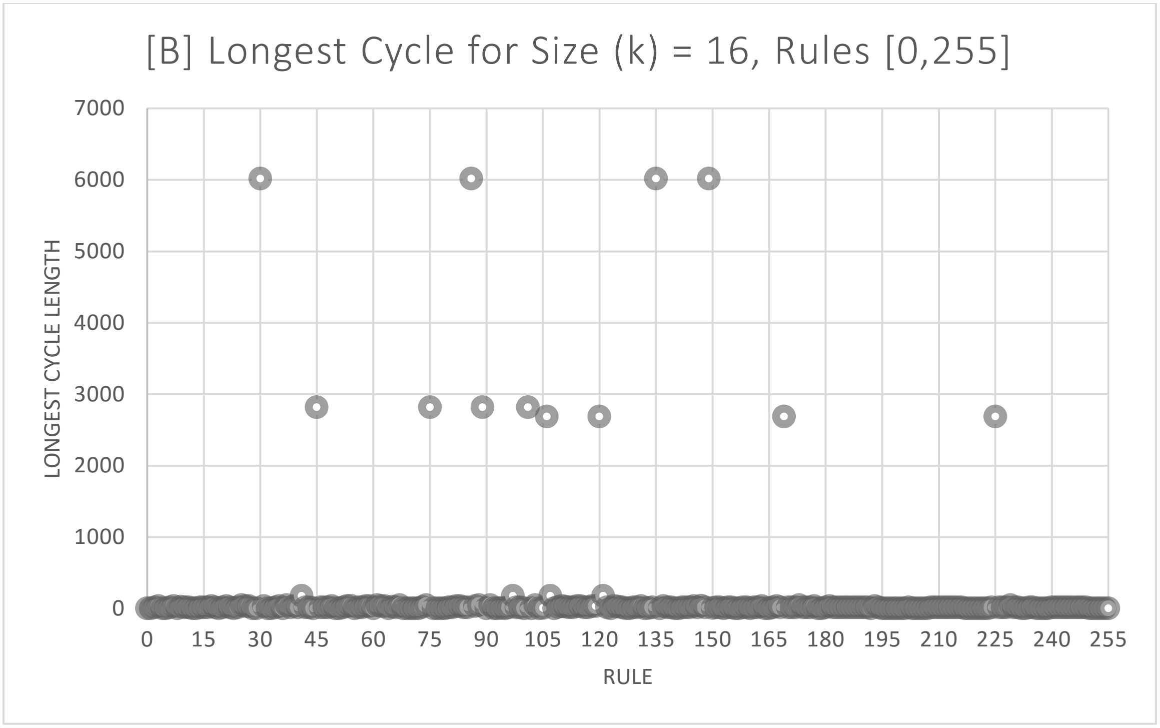


Figure 44 – Longest Cycle Diagram, All Rules, k = 16

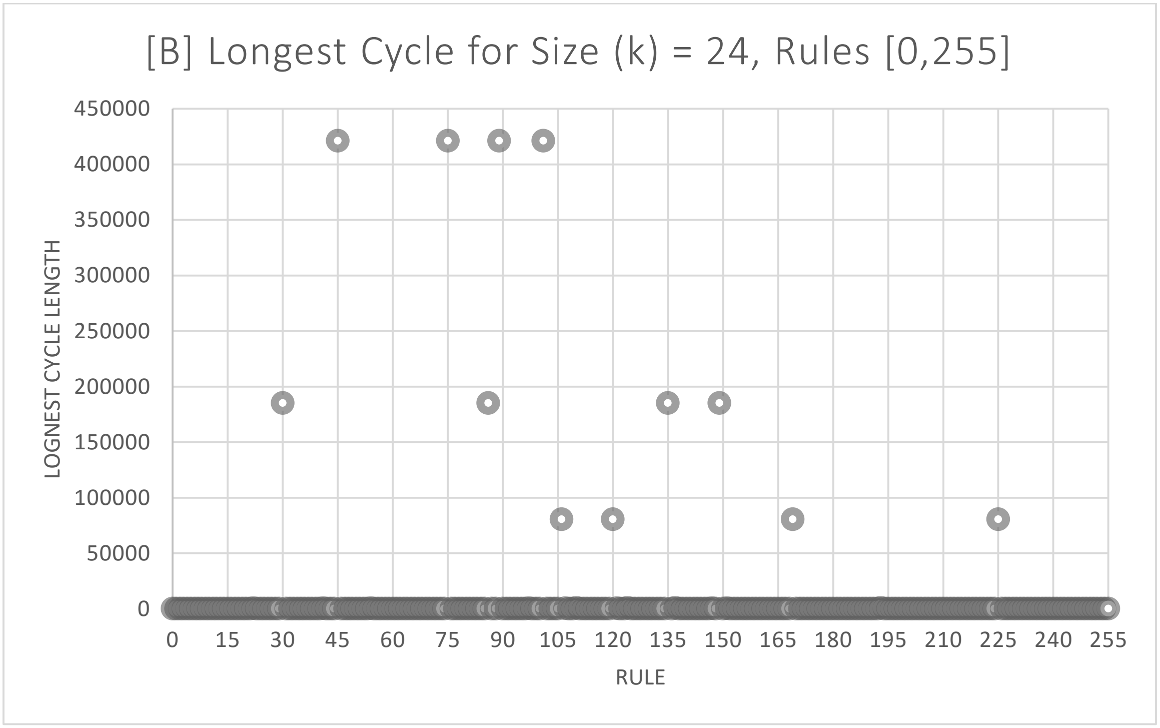


Figure 45 – Longest Cycle Diagram, All Rules, k = 24

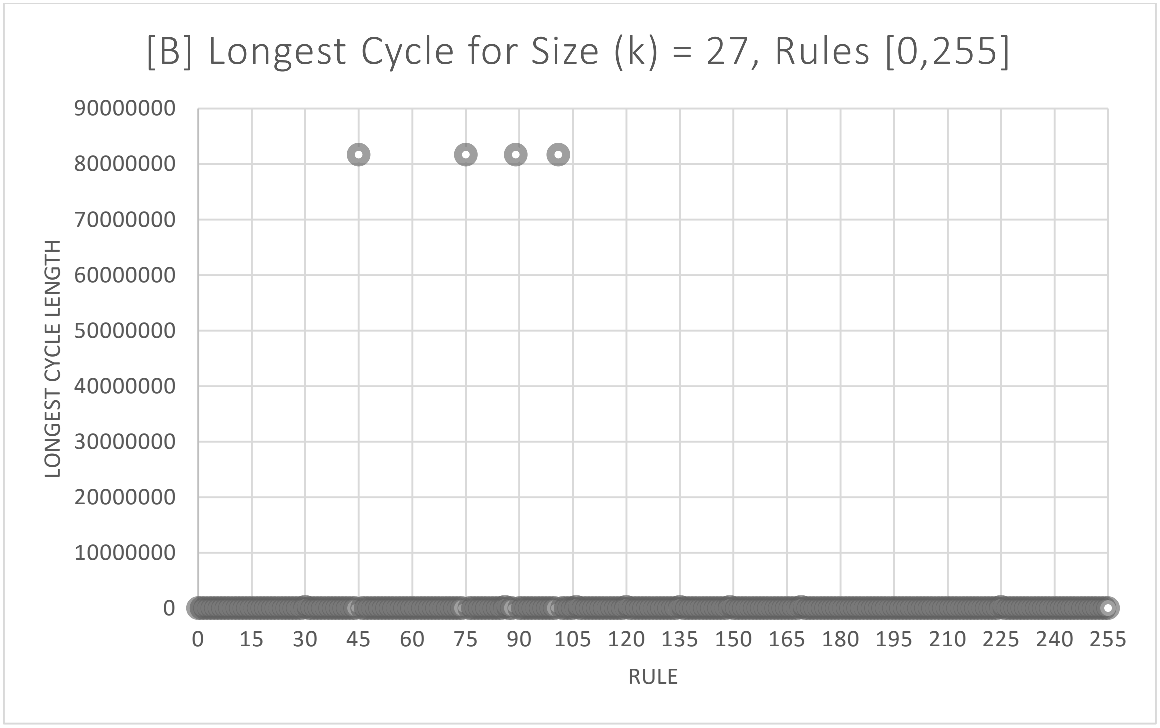
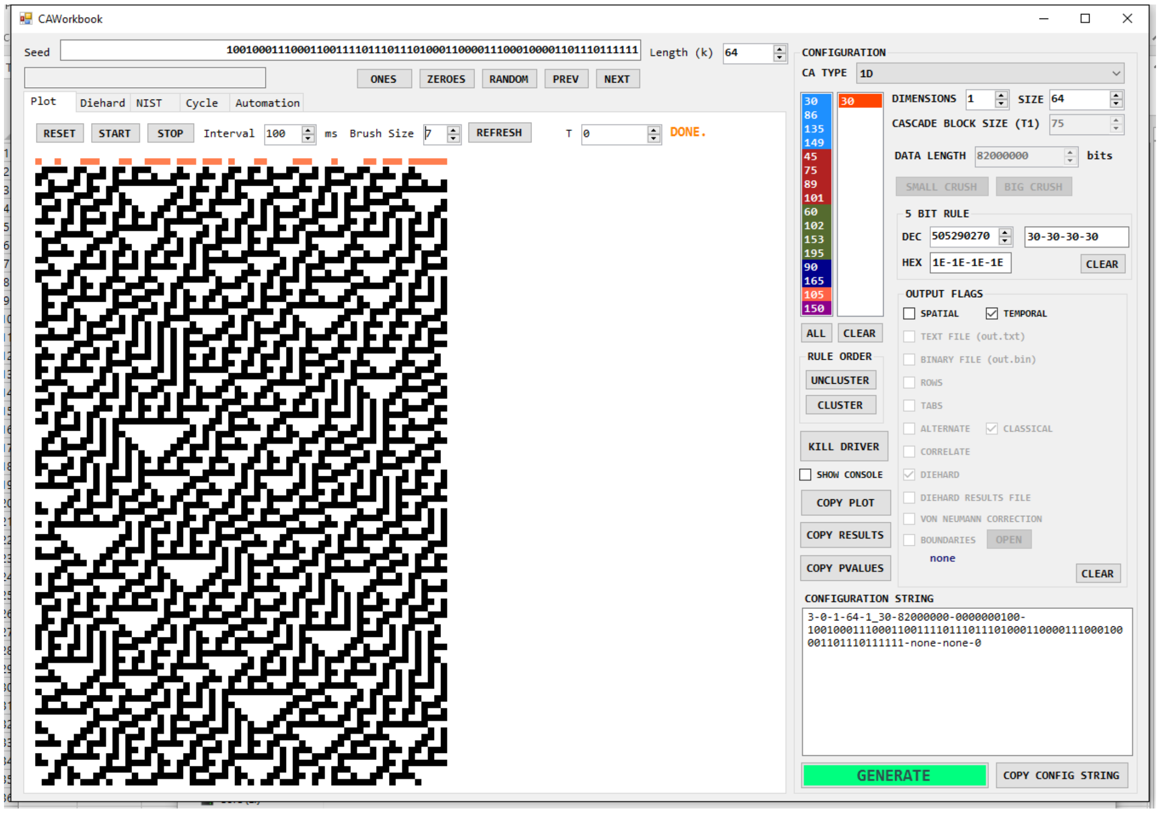


Figure 46 – Longest Cycle Diagram, All Rules, k = 27

APPENDIX B: CA WORKBOOK SCREENS

SECTION B.1 – Plot Screen

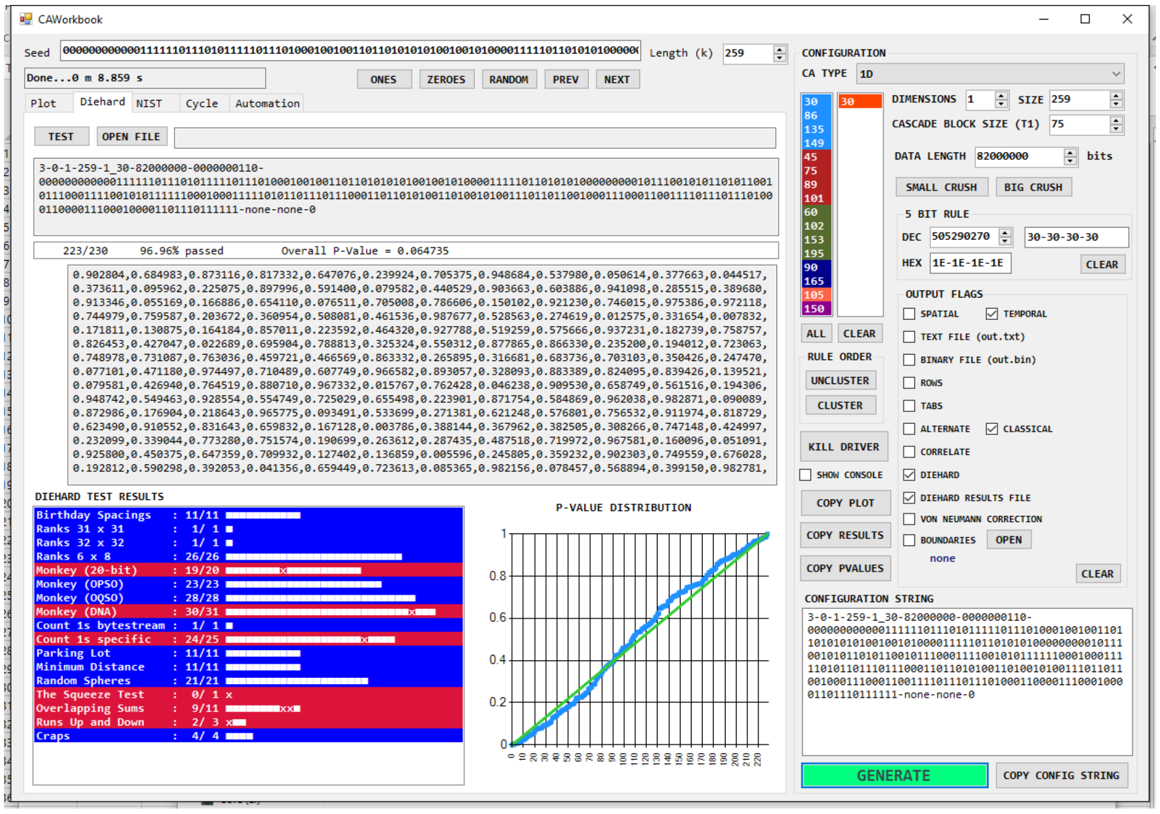


```
Seed  1001000111000110011110111011101000110000111000100001101110111111   Length (k)  64
Interval 100 ms   Brush Size 7   T 0

CONFIGURATION
CA TYPE  1D
30 86 135 149 45 75 89 101 60 102 153 195 90 165 105 150    30
DIMENSIONS  1   SIZE  64
CASCADE BLOCK SIZE (T1)  75
DATA LENGTH  82000000  bits
5 BIT RULE
DEC  505290270   30-30-30-30
HEX  1E-1E-1E-1E
OUTPUT FLAGS
[ ] SPATIAL   [✓] TEMPORAL
[ ] TEXT FILE (out.txt)
[ ] BINARY FILE (out.bin)
[ ] ROWS
[ ] TABS
[ ] ALTERNATE   [✓] CLASSICAL
[ ] CORRELATE
[✓] DIEHARD
[ ] DIEHARD RESULTS FILE
[ ] VON NEUMANN CORRECTION
[ ] BOUNDARIES
none
[ ] SHOW CONSOLE
CONFIGURATION STRING
3-0-1-64-1_30-82000000-0000000100-
1001000111000110011110111011101000110000111000100
001101110111111-none-none-0
```

Figure 47 – CA Workbook, Plot Screen

Plot allows a user to easily observe the ECA with a variety of configurations and rules, including starting seed, scrolling to specific timesteps, and observing the changes between seeds on the same rule.

SECTION B.2 – Diehard Screen

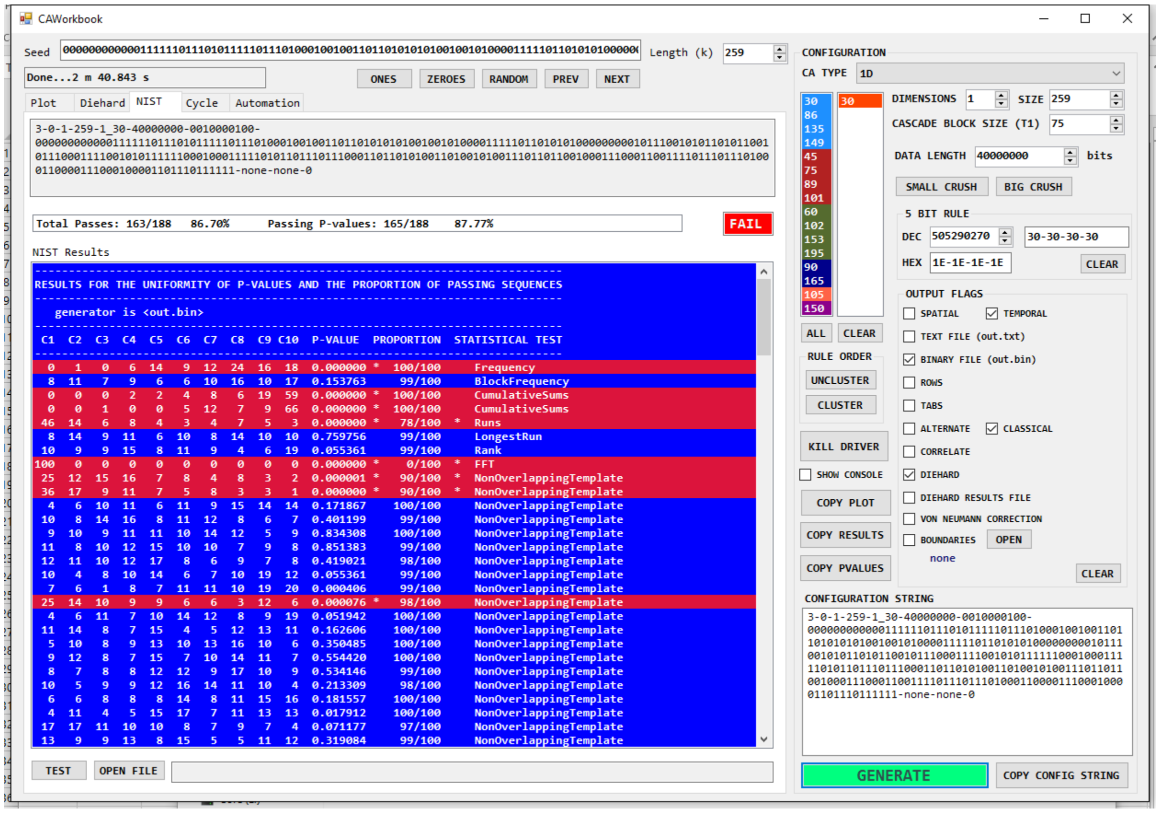


```
Seed  000000000000111111011101011111011101000100100110110101010100100101000011111011010101000000
Length (k) 259
Done...0 m 8.859 s

3-0-1-259-1_30-82000000-0000000110-
0000000000001111110111010111110111010001001001101101010101001001010000111110110101010000000001011100101011010110010
0111000111100101011111100010001111101011011101110001101101010011010010100111011011001000111000110011110111011110100
011000011100010000110111011111-none-none-0

    223/230     96.96% passed         Overall P-Value = 0.064735

0.902804,0.684983,0.873116,0.817332,0.647076,0.239924,0.705375,0.948684,0.537980,0.050614,0.377663,0.044517,
0.373611,0.095962,0.225075,0.897996,0.591400,0.079582,0.440529,0.903663,0.603886,0.941098,0.285515,0.389680,
0.913346,0.055169,0.166886,0.654110,0.076511,0.705008,0.786606,0.150102,0.921230,0.746015,0.975386,0.972118,
0.744979,0.759587,0.203672,0.360954,0.508081,0.461536,0.987677,0.528563,0.274619,0.012575,0.331654,0.007832,
0.171811,0.130875,0.164184,0.857011,0.223592,0.464320,0.927788,0.519259,0.575666,0.937231,0.182739,0.758757,
0.826453,0.427047,0.022689,0.695904,0.788813,0.325324,0.550312,0.877865,0.866330,0.235200,0.194012,0.723063,
0.748978,0.731087,0.763036,0.459721,0.466569,0.863332,0.265895,0.316681,0.683736,0.703103,0.350426,0.247470,
0.077101,0.471180,0.974497,0.710489,0.607749,0.966582,0.893057,0.328093,0.883389,0.824095,0.839426,0.139521,
0.079581,0.426940,0.764519,0.880710,0.967332,0.015767,0.762428,0.046238,0.909530,0.658749,0.561516,0.194306,
0.948742,0.549463,0.928554,0.554749,0.725029,0.655498,0.223901,0.871754,0.584869,0.962038,0.982871,0.090089,
0.872986,0.176904,0.218643,0.965775,0.093491,0.533699,0.271381,0.621248,0.576801,0.756532,0.911974,0.818729,
0.623490,0.910552,0.831643,0.659832,0.167128,0.003786,0.388144,0.367962,0.382505,0.308266,0.747148,0.424997,
0.232099,0.339044,0.773280,0.751574,0.190699,0.263612,0.287435,0.487518,0.719972,0.967581,0.160096,0.051091,
0.925800,0.450375,0.647359,0.709932,0.127402,0.136859,0.005596,0.245805,0.359232,0.902303,0.749559,0.676028,
0.192812,0.590298,0.392053,0.041356,0.659449,0.723613,0.085365,0.982156,0.078457,0.568894,0.399150,0.982781,

DIEHARD TEST RESULTS
Birthday Spacings    : 11/11
Ranks 31 x 31        :  1/ 1
Ranks 32 x 32        :  1/ 1
Ranks 6 x 8          : 26/26
Monkey (20-bit)      : 19/20
Monkey (OPSO)        : 23/23
Monkey (OQSO)        : 28/28
Monkey (DNA)         : 30/31
Count 1s bytestream  :  1/ 1
Count 1s specific    : 24/25
Parking Lot          : 11/11
Minimum Distance     : 11/11
Random Spheres       : 21/21
The Squeeze Test     :  0/ 1 x
Overlapping Sums     :  9/11
Runs Up and Down     :  2/ 3
Craps                :  4/ 4

CONFIGURATION
CA TYPE  1D
Rules: 30 86 135 149 45 75 89 101 60 102 153 195 90 165 105 150
Selected: 30
DIMENSIONS 1    SIZE 259
CASCADE BLOCK SIZE (T1) 75
DATA LENGTH 82000000 bits
5 BIT RULE
DEC 505290270    30-30-30-30
HEX 1E-1E-1E-1E
RULE ORDER
[ ] SHOW CONSOLE
OUTPUT FLAGS
[ ] SPATIAL    [✓] TEMPORAL
[ ] TEXT FILE (out.txt)
[ ] BINARY FILE (out.bin)
[ ] ROWS
[ ] TABS
[ ] ALTERNATE  [✓] CLASSICAL
[ ] CORRELATE
[✓] DIEHARD
[✓] DIEHARD RESULTS FILE
[ ] VON NEUMANN CORRECTION
[ ] BOUNDARIES
    none

CONFIGURATION STRING
3-0-1-259-1_30-82000000-0000000110-
0000000000001111110111010111110111010001001001101
1010101010010010100001111101101010100000000010111
0010101101011001011100011110010101111110001000111
1101011011101110001101101010011010010100111011011
0010001110001100111101110111010001100001110001000
01101110111111-none-none-0
```

Figure 48 – CA Workbook, Diehard Screen

Diehard allows a user to see pseudo-random analysis on a specific CA configuration, starting seed, and rules. Functionality for testing a user-generated binary file is also available. The configuration string, raw test results, grouped results, and p-value distribution are all shown to aid in quick one-off testing. A line above the raw test data shows the overall p-value and passing test results.

SECTION B.3 – NIST Screen



Seed 0000000000001111110111010111110111010001001001101101010101001001010000111110110101010000000

Length (k) 259

Done...2 m 40.843 s

```
3-0-1-259-1_30-40000000-0010000100-
000000000000111111011101011111011101000100100110110101010100100101000011111011010101000000000101110010101101011001
011100011110010101111110001000111110101101110111000110110101001101001010011101101100100011100011001111011101110100
01100001110001000011011101111111-none-none-0
```

```
Total Passes: 163/188   86.70%      Passing P-values: 165/188    87.77%
```

NIST Results

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <out.bin>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
  0   1   0   6  14   9  12  24  16  18  0.000000 *  100/100     Frequency
  8  11   7   9   6   6  10  16  10  17  0.153763     99/100     BlockFrequency
  0   0   0   2   2   4   8   6  19  59  0.000000 *  100/100     CumulativeSums
  0   0   1   0   0   5  12   7   9  66  0.000000 *  100/100     CumulativeSums
 46  14   6   8   4   3   4   7   5   3  0.000000 *   78/100  *  Runs
  8  14   9  11   6  10   8  14  10  10  0.759756     99/100     LongestRun
 10   9   9  15   8  11   9   4   6  19  0.055361     99/100     Rank
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  FFT
 25  12  15  16   7   8   4   8   3   2  0.000001 *   90/100  *  NonOverlappingTemplate
 36  17   9  11   7   5   8   3   3   1  0.000000 *   90/100  *  NonOverlappingTemplate
  4   6  10  11   6  11   9  15  14  14  0.171867    100/100     NonOverlappingTemplate
 10   8  14  16   8  11  12   8   6   7  0.401199     99/100     NonOverlappingTemplate
  9  10   9  11  11  10  14  12   5   9  0.834308    100/100     NonOverlappingTemplate
 11   8  10  12  15  10  10   7   9   8  0.851383     99/100     NonOverlappingTemplate
 12  11  10  12  17   8   6   9   7   8  0.419021     98/100     NonOverlappingTemplate
 10   4   8  10  14   6   7  10  19  12  0.055361     99/100     NonOverlappingTemplate
  7   6   1   8   7  11  11  10  19  20  0.000406     99/100     NonOverlappingTemplate
 25  14  10   9   9   6   6   3  12   6  0.000076 *   98/100     NonOverlappingTemplate
  4   6  11   7  10  14  12   8   9  19  0.051942    100/100     NonOverlappingTemplate
 11  14   8   7  15   4   5  12  13  11  0.162606    100/100     NonOverlappingTemplate
  5  10   8   9  13  10  13  16  10   6  0.350485    100/100     NonOverlappingTemplate
  9  12   8   7  15   7  10  14  11   7  0.554420    100/100     NonOverlappingTemplate
  8   7   8   8  12  12   9  17  10   9  0.534146     99/100     NonOverlappingTemplate
 10   5   9   9  12  16  14  11  10   4  0.213309     98/100     NonOverlappingTemplate
  6   6   8   8   8  14   8  11  15  16  0.181557    100/100     NonOverlappingTemplate
  4  11   4   5  15  17   7  11  13  13  0.017912    100/100     NonOverlappingTemplate
 17  17  11  10  10   8   7   9   7   4  0.071177     97/100     NonOverlappingTemplate
 13   9   9  13   8  15   5   5  11  12  0.319084     99/100     NonOverlappingTemplate
```

CONFIGURATION

CA TYPE 1D

30 86 135 149 45 75 89 101 60 102 153 195 90 165 105 150

30

DIMENSIONS 1 SIZE 259

CASCADE BLOCK SIZE (T1) 75

DATA LENGTH 40000000 bits

5 BIT RULE

DEC 505290270 30-30-30-30

HEX 1E-1E-1E-1E

RULE ORDER

☐ SHOW CONSOLE

OUTPUT FLAGS

☐ SPATIAL ☑ TEMPORAL
☐ TEXT FILE (out.txt)
☑ BINARY FILE (out.bin)
☐ ROWS
☐ TABS
☐ ALTERNATE ☑ CLASSICAL
☐ CORRELATE
☑ DIEHARD
☐ DIEHARD RESULTS FILE
☐ VON NEUMANN CORRECTION
☐ BOUNDARIES
none

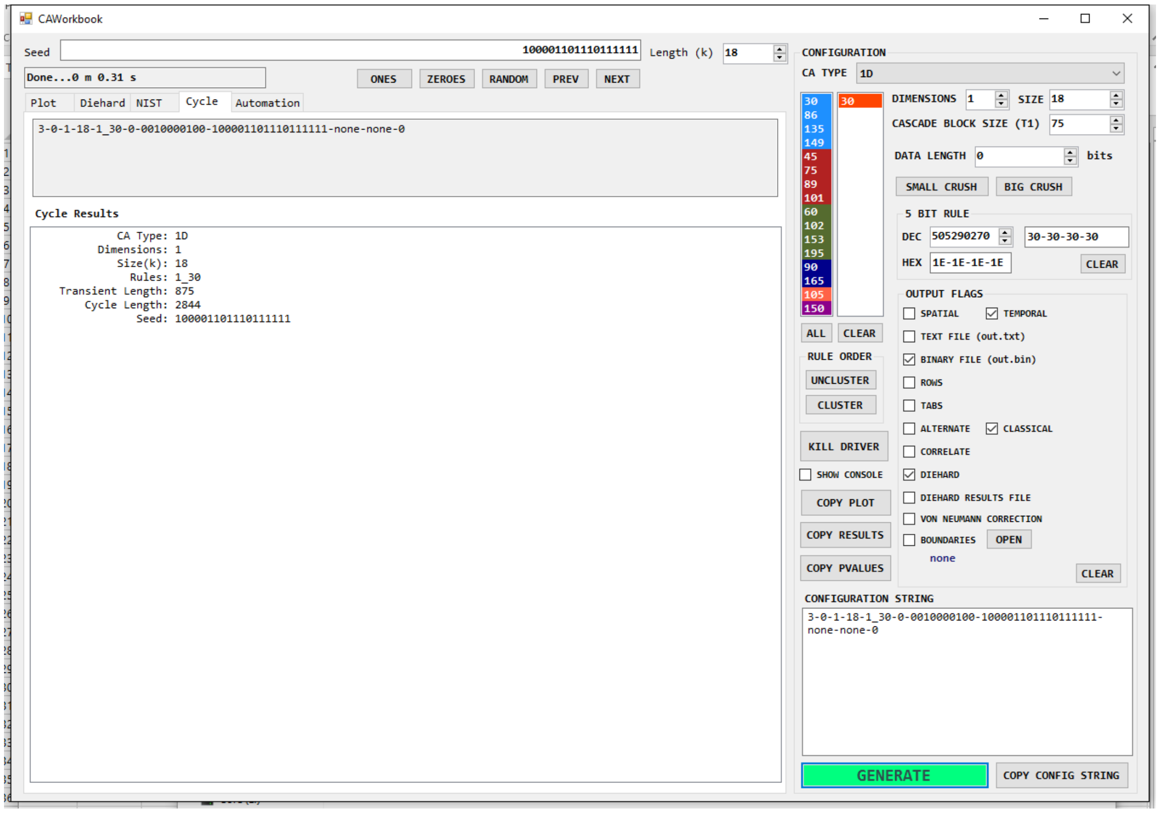
CONFIGURATION STRING

```
3-0-1-259-1_30-40000000-0010000100-
00000000000011111101110101111101110100010010011011
01010101001001010000111110110101010000000001011100
10101101011001011100011110010101111110001000111110
10110111011100011011010100110100101001110110110010
00111000110011110111011101000110000111000100001101
110111111-none-none-0
```

Figure 49 – CA Workbook, NIST Screen

NIST allows a user to run the National Institute of Technology's Statistical Testing Suite against a CA configuration or user-supplied binary file. The final results text is parsed and placed into the grid for viewing following test completion. In addition to the test's configuration string, a line above the results grid shows the total number of passing tests and whether or not the test failed or passed overall.

SECTION B.4 – Cycle Screen



Seed 100001101110111111 Length (k) 18

Done...0 m 0.31 s

```
3-0-1-18-1_30-0-0010000100-100001101110111111-none-none-0
```

Cycle Results

```
          CA Type: 1D
       Dimensions: 1
          Size(k): 18
            Rules: 1_30
 Transient Length: 875
     Cycle Length: 2844
             Seed: 100001101110111111
```

CONFIGURATION

CA TYPE 1D

30 86 135 149 45 75 89 101 60 102 153 195 90 165 105 150

30

DIMENSIONS 1 SIZE 18

CASCADE BLOCK SIZE (T1) 75

DATA LENGTH 0 bits

5 BIT RULE

DEC 505290270 30-30-30-30

HEX 1E-1E-1E-1E

OUTPUT FLAGS

☐ SPATIAL ☑ TEMPORAL
☐ TEXT FILE (out.txt)
☑ BINARY FILE (out.bin)
☐ ROWS
☐ TABS
☐ ALTERNATE ☑ CLASSICAL
☐ CORRELATE
☑ DIEHARD
☐ DIEHARD RESULTS FILE
☐ VON NEUMANN CORRECTION
☐ BOUNDARIES
none

☐ SHOW CONSOLE

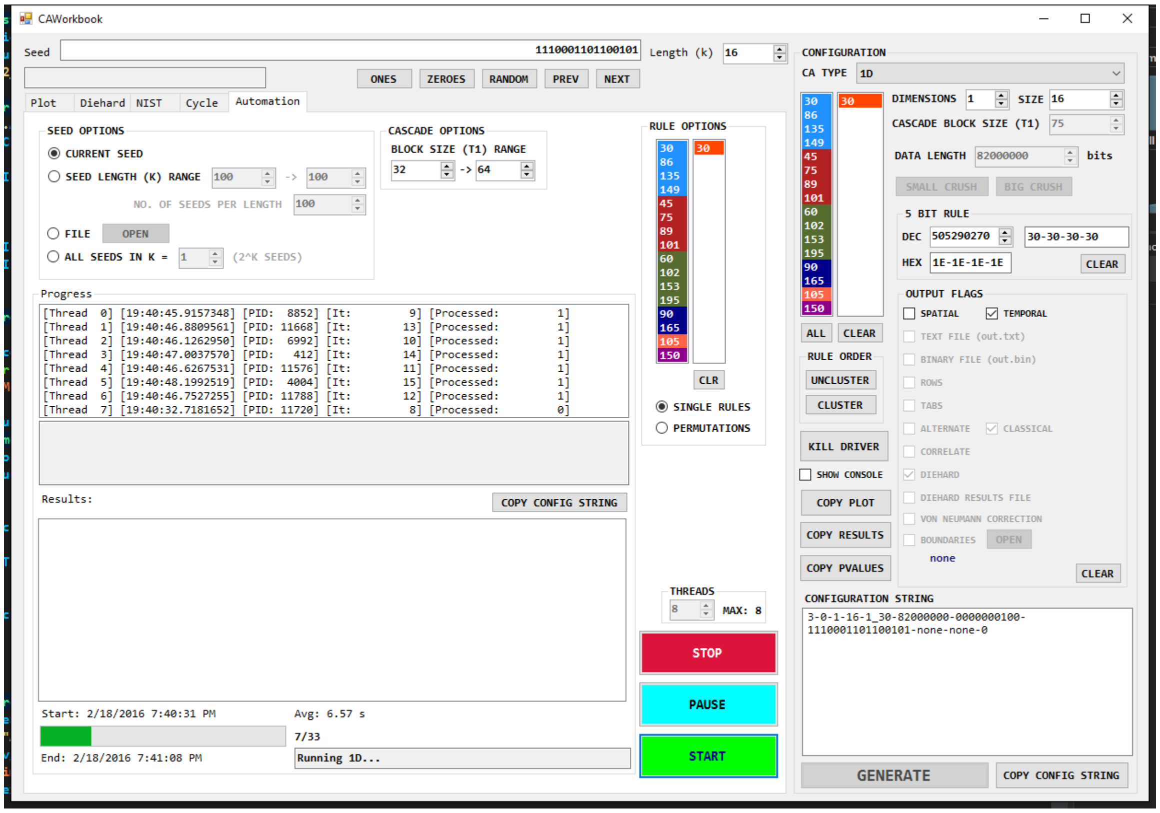
CONFIGURATION STRING

```
3-0-1-18-1_30-0-0010000100-100001101110111111-
none-none-0
```

Figure 50 – CA Workbook, Cycle Screen

Cycle allows the user to run a single cycle analysis for a given configuration and starting seed, usually reserved only for ECA. The test seed, type, dimensions, size, and rules are output in addition to the transient and cycle length.

SECTION B.5 – Automation Screen



Seed 1110001101100101 Length (k) 16

SEED OPTIONS
- [✓] CURRENT SEED
- SEED LENGTH (K) RANGE 100 -> 100
- NO. OF SEEDS PER LENGTH 100
- FILE
- ALL SEEDS IN K = 1 (2^K SEEDS)

CASCADE OPTIONS
BLOCK SIZE (T1) RANGE 32 -> 64

RULE OPTIONS
30 86 135 149 45 75 89 101 60 102 153 195 90 165 105 150 | 30
- [✓] SINGLE RULES
- PERMUTATIONS

Progress

```
[Thread  0] [19:40:45.9157348] [PID:  8852] [It:         9] [Processed:         1]
[Thread  1] [19:40:46.8809561] [PID: 11668] [It:        13] [Processed:         1]
[Thread  2] [19:40:46.1262950] [PID:  6992] [It:        10] [Processed:         1]
[Thread  3] [19:40:47.0037570] [PID:   412] [It:        14] [Processed:         1]
[Thread  4] [19:40:46.6267531] [PID: 11576] [It:        11] [Processed:         1]
[Thread  5] [19:40:48.1992519] [PID:  4004] [It:        15] [Processed:         1]
[Thread  6] [19:40:46.7527255] [PID: 11788] [It:        12] [Processed:         1]
[Thread  7] [19:40:32.7181652] [PID: 11720] [It:         8] [Processed:         0]
```

Results:

Start: 2/18/2016 7:40:31 PM Avg: 6.57 s
7/33
End: 2/18/2016 7:41:08 PM Running 1D...

THREADS 8 MAX: 8

CONFIGURATION
CA TYPE 1D
30 86 135 149 45 75 89 101 60 102 153 195 90 165 105 150 | 30
DIMENSIONS 1 SIZE 16
CASCADE BLOCK SIZE (T1) 75
DATA LENGTH 82000000 bits

5 BIT RULE
DEC 505290270 30-30-30-30
HEX 1E-1E-1E-1E

RULE ORDER

SHOW CONSOLE

OUTPUT FLAGS
- SPATIAL [✓ TEMPORAL]
- TEXT FILE (out.txt)
- BINARY FILE (out.bin)
- ROWS
- TABS
- ALTERNATE [✓ CLASSICAL]
- CORRELATE
- [✓ DIEHARD]
- DIEHARD RESULTS FILE
- VON NEUMANN CORRECTION
- BOUNDARIES none

CONFIGURATION STRING

```
3-0-1-16-1_30-82000000-0000000100-
1110001101100101-none-none-0
```

Figure 51 – CA Workbook, Automation Screen

Automation is a powerful tool that allows parallelized testing of a range of configuration options – seed length, rule combinations, number of seeds, configuration ranges. It creating a separate process for each test, up to a maximum concurrent number of threads that the machine can technically handle – normally the number of cores, sometimes multiplied by two if hyper threading is enabled. The test configuration data and passing number of p-values are collected into a global results file for later analysis.

APPENDIX C: RGB MAPPING FOR ECA RULES

SECTION C.1 – Bitmapping Process

In mapping elementary CA to a bitmap image, each state's *next* state is stored as an integer rather than a binary string, then mapped to an RGB color, which uses up to 24 bits, or size k = 24. Using a square, the resulting image has both a width and height of $\sqrt{2^k}$. Mapping from left to right, top to bottom, 0 at the top left and 2^k-1 at the bottom right. For k = 16, only green and blue (R[GB]) will be used since there are only 2 bytes of data. To accurately portray a 24-bit scheme would require 3D modeling. The following figure shows the normal distribution when colors are created this way, from seed 0 (0x000000) to 65535 (0x00FFFF). Figures 53 and 54 show the resulting bitmap images for rule groups 30 and 45 for k = 16.

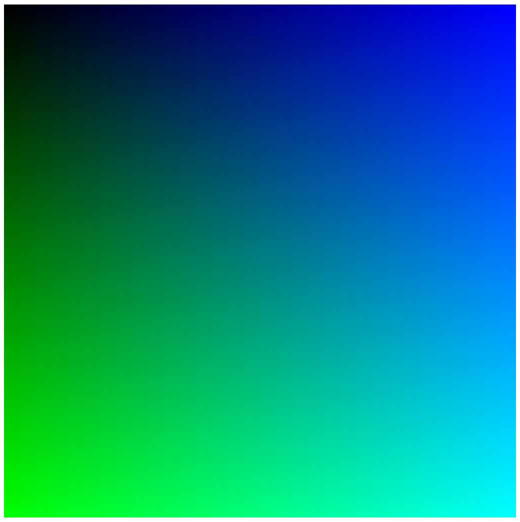


Figure 52 – RGB 16-bit, Normal Colorization

SECTION C.2 – Rule 30 [30, 86, 135, 149]

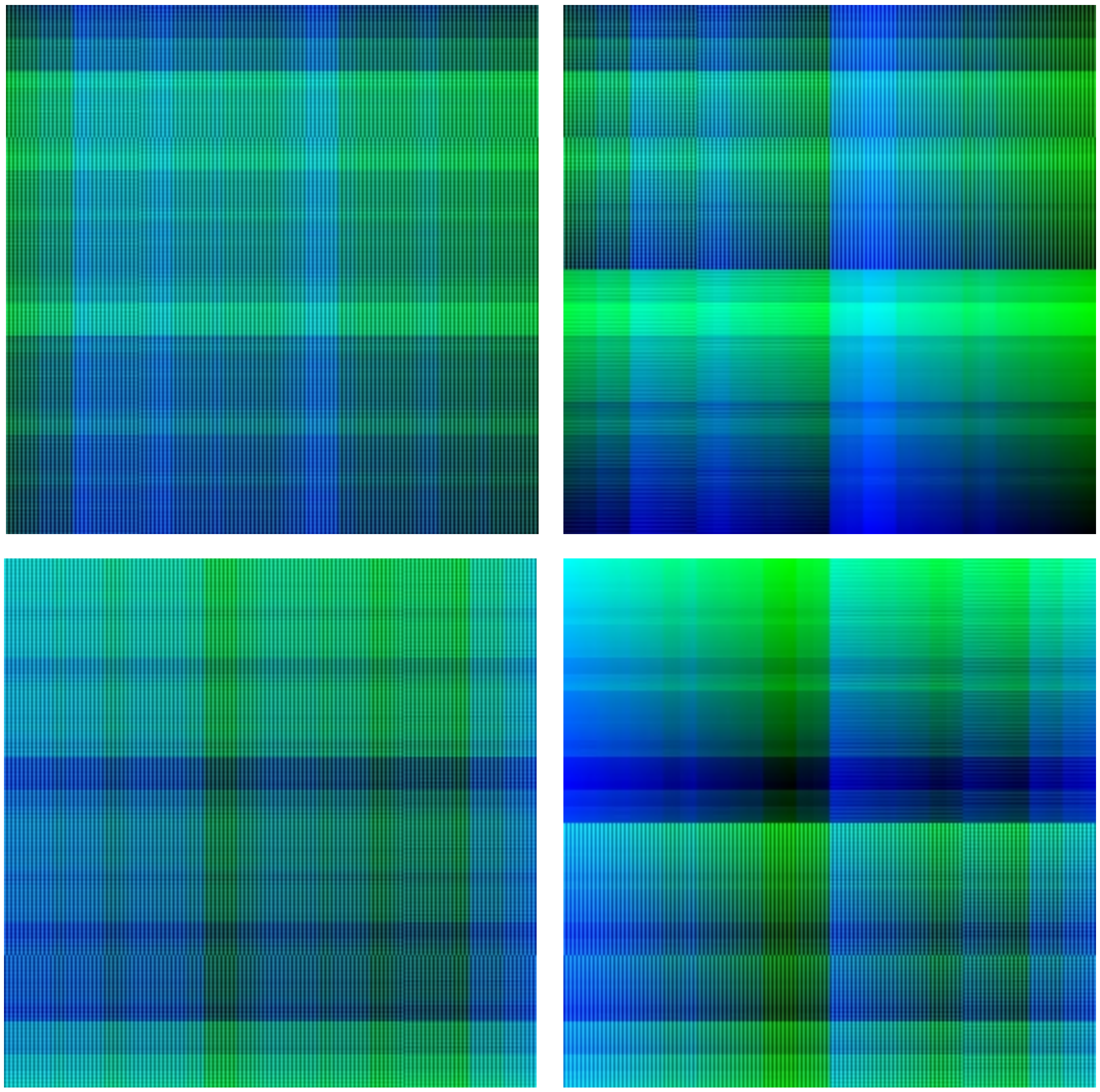


Figure 53 – RGB Mapping, Rules 30, 86, 135, 149 (left to right)

SECTION C.3 – Rule 45 [45, 75, 89, 101]

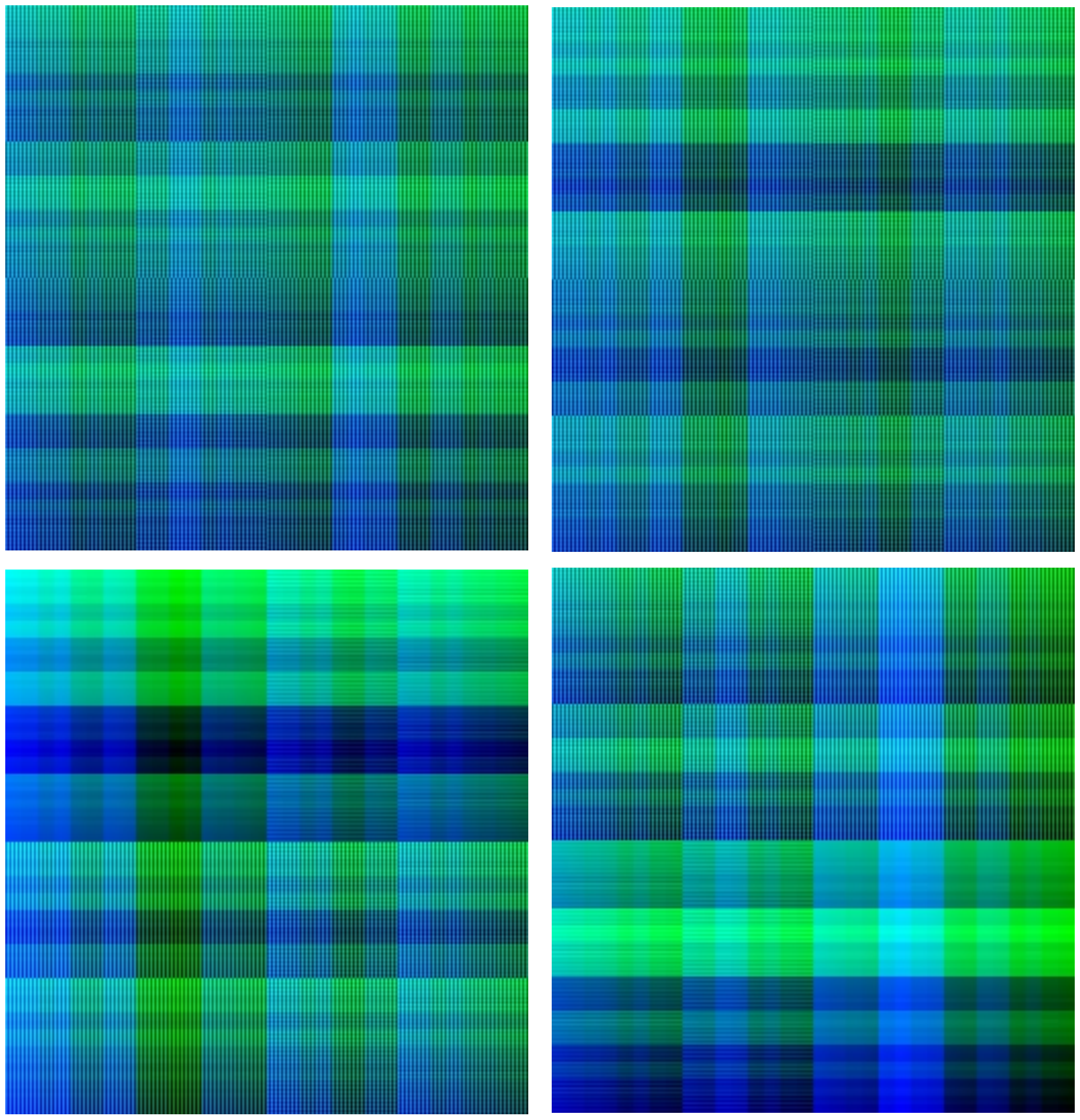


Figure 54 – RGB Mapping, Rules 45, 75, 89, 101 (left to right)