Pseudorandom Sequence Generation Using Binary Cellular Automata

A thesis submitted to the
Graduate School of Natural and Applied Sciences

by

Nihal VATANDAŞ

in partial fulfillment for the
degree of Master of Science

in

Electronics and Computer Engineering

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science in Electronics and Computer Engineering.

APPROVED BY:

Assist. Prof. Ahmet Bulut .....
(Thesis Advisor)

Assist. Prof. İsmail Demirkan .....
(Thesis Co-advisor)

Assoc. Prof. Vural Aksakallı .....

Assist. Prof. Barış Arslan .....

This is to confirm that this thesis complies with all the standards set by the Graduate School of Natural and Applied Sciences of İstanbul Şehir University:

DATE OF APPROVAL: 14.07.2014

SEAL/SIGNATURE:

Declaration of Authorship

I, Nihal VATANDAŞ, declare that this thesis titled, 'Pseudorandom Sequence Generation Using Binary Cellular Automata' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: N Vatandaş

Date: 14.07.2014

Pseudorandom Sequence Generation Using Binary Cellular Automata

Nihal Vatandaş

Abstract

Random numbers are an integral part of many applications from computer simulations, gaming, security protocols to the practices of applied mathematics and physics. As randomness plays more critical roles, cheap and fast generation methods are becoming a point of interest for both scientific and technological use.

Cellular Automata (CA) is a class of functions which attracts attention mostly due to the potential it holds in modeling complex phenomena in nature along with its discreteness and simplicity. Several studies are available in the literature expressing its potentiality for generating randomness and presenting its advantages over commonly used random number generators.

Most of the researches in the CA field focus on one-dimensional 3-input CA rules. In this study, we perform an exhaustive search over the set of 5-input CA to find out the rules with high randomness quality. As the measure of quality, the outcomes of NIST Statistical Test Suite are used.

Since the set of 5-input CA rules is very large (including more than 4.2 billions of rules), they are eliminated by discarding poor-quality rules before testing.

In the literature, generally entropy is used as the elimination criterion, but we preferred mutual information. The main motive behind that choice is to find out a metric for elimination which is directly computed on the truth table of the CA rule instead of the generated sequence. As the test results collected on 3- and 4-input CA indicate, all rules with very good statistical performance have zero mutual information. By exploiting this observation, we limit the set to be tested to the rules with zero mutual information. The reasons and consequences of this choice are discussed.

In total, more than 248 millions of rules are tested. Among them, 120 rules show outstanding performance with all attempted neighborhood schemes. Along with these tests, one of them is subjected to a more detailed testing and test results are included.

Keywords: Cellular Automata, Pseudorandom Number Generators, Randomness Tests

İkili Cellular Automata Fonksiyonları ile Rasgele Dizi Üretimi

Nihal Vatandaş

Öz

Rasgele sayılar simülasyonlardan şans oyunlarına, güvenlik protokollerinden uygulamalı matematik ve fizik alanlarına kadar bir çok uygulamanın işleyişinde yer alan temel unsurlardan biridir. Rasgele sayıların bilimsel ve teknolojik amaçlı kullanım alanı genişledikçe hızlı ve ekonomik üretim yöntemleri de araştırmacılar için ilgi konusu olmaktadır.

Cellular Automata (CA), basit yapısının yanında tabiattaki kamaşık yapılı olayları modellemeye uygunluğuyla ön plana çıkmış bir çeşit ayrık fonksiyonlar grubudur. Rasgele dizi üretmeye olan elverişliliğini ve yaygın olarak kullanılan rasgele sayı üreteçlerine üstün gelen yönlerini açıklayan bir çok çalışma halihazırda literatürde yer almaktadır.

CA alanında ekseriyetle 3 girdi alan tek boyutlu fonksiyonlar üzerine araştırmalar bulunuyor. Biz bu çalışmada, 5 girdi alan CA fonksiyonları üzerinde bir tarama yaparak rasgele sayı üretme kabiliyeti yüksek olan fonksiyonları belirlemeyi hedefledik. Ölçü olarak NIST tarafından hazırlanan istatistiksel test grubu sonuçlarını baz aldık.

5 girdi alan CA fonksiyonları kümesi 4,2 milyardan fazla fonksiyon içeren çok geniş bir küme. Dolayısıyla fonksiyonları teste tabi tutmadan evvel iyi sonuç vermeyeceği tahmin edilen fonksiyonların elenmesi gerekiyor.

Literatürde, bu tarz bir eleme söz konusu olduğunda entropi değerlerinin baz alındığını görürüz. Fakat biz bu çalışmada karşılıklı bilgiyi (mutual information) esas aldık. Bu değişikliğe gitmekteki asıl amaç, entropi gibi üretilmiş sayı dizisi üzerinde hesaplanan bir ölçü yerine doğrudan fonksiyon üzerinde hesaplanan pratik bir ölçünün kullanılabilirliğini araştırmaktı. 3 ve 4 girdi alan fonksiyonlardan edinilen verilere göre çok iyi istatistiksel nitelikte dizi üreten fonksiyonların tamamının karşılıklı bilgi değerinin sıfır olduğu görülüyor. Bu gözlemden yola çıkarak, 5 girdi alan fonksiyonlar üzerinde sıfır karşılıklı bilgiye sahip olmayı bir eleme kriteri olarak kullandık. Bu seçimin sebepleri ve sonuçları da çalışmada geniş olarak incelendi.

Sonuç olarak, 248 milyonun üzerinde fonksiyon teste tabi tutuldu ve test sonuçları sunuldu. Bunların arasından istisnai nitelikte iyi performans gösteren 120 fonksiyon çalışmanın sonunda belirtildi. Ek olarak, 120 fonksiyon arasından seçilen bir fonksiyonun ayrıntılı istatistiksel incelemesine yer verildi.

Anahtar Sözcükler: Cellular Automata, Rasgele Dizi Üretimi

Acknowledgments

This study was initiated and continued under the supervision of Prof. Çetin Kaya Koç until his departure from İstanbul Şehir University. I would like to express my deepest gratitude to him for his encouraging guidance and support all through the time.

I am particularly grateful to my advisors Ahmet Bulut and İsmail Demirkan for their generous help and patience with me.

I would like to thank İsa Sertkaya for his sincere support and advices. His willingness to give his time so generously has been very much appreciated.

Finally, special thanks to Eren Yener from the IT department for his help in offering me the resources in running the programs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Random numbers are an integral part of many applications from computer simulations, statistical sampling, gaming to the practices of applied mathematics and physics. They are becoming increasingly important as mathematical models involving probability and statistics find wider use in science and technology. Beside the randomized algorithms and probabilistic methods, a great deal of the current cryptographic protocols use randomness in a crucial way that a failure in randomization could lead a failure in the whole security system [1–3]. As randomness plays more critical roles, random number generation becomes a more tricky problem to consider especially on the issues of reliability and production speed. Cheap and fast generation methods are becoming a point of interest for both computational and security purposes.

Looking at the big picture, there are two basic methods for producing randomness and the output sequence is named according to the chosen method of generation. When producing *true random numbers*, the source of randomness is the nature itself, evoking the philosophical question in mind if there exists a phenomenon as randomness in nature at all. In case of generating *pseudorandom numbers*, a deterministic algorithm is employed as the generator though this sounds quite contradictory to the very nature of randomness. The second method leaves behind the existential discussion on randomness and directly focuses on the practical needs and features that a desired random sequence must possess. Both of these techniques have their own advantages and usage, that is, one of them is not strictly preferred to the other.

True random numbers are more costly to generate for they require a specialized hardware to extract randomness from a physical phenomenon in the nature. On the other hand, pseudorandom sequences are completely produced in a digital environment, therefore allow for fast and inexpensive products. A selection between the two is made according

to the utilization purpose. Actually, in many cases, true randomness is not necessary. This opens a wide ground for use of pseudorandom number generators (PRNGs).

However, PRNG algorithms are not so easy to find [4]. The outcome of a good PRNG is expected to fulfill certain requirements like uniform distribution, lack of correlation and unpredictability, although not all of them are required for every application. For example, unpredictability is a vital norm for cryptographic uses but not for Monte Carlo simulations. But in all cases, statistical quality is a must. In some cases, a PRNG algorithm is constructed on a suitable algebraic structure so that it produces sequences with provably uniform distribution [5]. For the other cases, there are statistical test suites such as NIST [6], DIEHARD [7], ENT [8] available to verify the statistical quality of a generated sequence.

This study focuses on exploring the functions with good statistical quality among a certain family of one dimensional binary Cellular Automata (CA) rules, namely the ones with 3-, 4- and 5-input and adjacent neighborhood. The measure of statistical quality is derived from the outcomes of NIST Statistical Test Suite.

CA is a class of functions that is mainly characterized with local interaction, parallel evolution and its discreteness in time, space and value [9]. Since its first big rise in 1950s with the work of von Neumann, it has been subjected to various mathematical and physical analysis [10]. With its ability to express dynamic systems, CA shares a common ground with many chief branches of science such as biology, physics, mathematics and computer science. It gained popularity mostly due to the potential it holds in modeling complex phenomena in nature along with its simplicity. In the later works, a strong emphasis was paid to its extremely complex and varied behavior despite being a discrete model of simple construction. This is the very quality that makes it a promising tool for fast random number generation.

The studies on CA so far have provided various angles through how CA can be used in modeling complex systems. The first suggestion of using CA in PRNG structure came from Wolfram. In [9], Wolfram claims that the function known as *Rule 30* has an extraordinary potential of complexity therefore is a good candidate for random number generation. Later, Serra et. al. proved that any LFSR can be modeled as a combination of linear CA rules [11]. Hortensius showed that CA is very efficient in pseudorandom number generation because of its superior architecture suitable for parallelization and it is even better than LFSR, which is commonly preferred for low-cost generation [12, 13]. Also, higher dimensional examples have been studied. In [14], it was shown that unsurprisingly two-dimensional CA performs better than one-dimensional CA in quality of randomness. But whether the enhanced quality of randomness compensates for the implementation complexity is a question. In the context of pseudorandom generation

problem, many other models with performance evaluations of some hybrid and uniform models are available and presented in Chapter 3.

The general research trend in CA area, both in mathematics and computer science fields, focuses more on one-dimensional 3-input CA rules and mostly elaborating on specific examples. Comprehensive conclusions are difficult to draw. This may be due to the fact that despite the seventy years of analysis on several fronts, CA can still be considered as yet not totally explored and it seems that its exploration is closely related with the developments in dynamic and complex system theories. However, there is evidence to suggest that CA is quite promising for PRNG algorithms seeing its advantage over conventional methods [15–17].

In this study, we perform an exhaustive search over the set of 5-input CA to find out the rules with high randomness quality. Since the set of 5-input CA rules is very large, an elimination is processed to discard the poor-quality rules before testing. In the literature, generally entropy is used as the elimination criterion, but we preferred mutual information. Since mutual information is not known to be used for elimination in any previous study, the reasons and consequences of this choice are discussed via examples from 3- and 4-input rule sets. In total, more than 248 millions of rules are tested. Among them, 120 rules show outstanding performance with all attempted neighborhood schemes. Along with these tests, one of them is subjected to a more detailed testing and test results are included.

In the following, Chapter 2 provides the background information about random number generation and randomness testing. In Chapter 3, we review CA, its definitive parameters and the work done by using various forms of CA for PRNG constructions. The testing strategy, the experimental setup and the obtained results are presented in Chapter 4. Chapter 5 concludes with an overall assessment of the followed procedure and the test results.

Chapter 2

Random Number Sequences

2.1 Introduction

Randomness is a controversial issue both philosophically and technically. Its existence against hard determinism has long been a matter of debate while formalization problems, like definition and measurability, lead to another track of discussion. Yet, randomness is a part of everyday life that this hard-to-define word quickly associates with a number of other words in our minds. Literally, randomness describes lack of pattern, arbitrariness and unpredictability. Although defining formally such a notion that is identified with disorder seems to be a contradictory act, several attempts have been made at conceptualizing randomness.

Many natural phenomena might seem to happen randomly, without following an exact rule. As human knowledge broadens and becomes able to read inner mechanism of happenings in nature, randomness becomes more questionable. This discussion takes us to the issue of whether the whole universe works deterministically. Initially one may even give credit to the idea that the science can ever grow to be able to describe everything in nature so that there remains no room for randomness. But with the laws of quantum mechanics we know that the outcomes of certain physical experiments are unpredictable. That is, some physical phenomena exist in nature that are believed to exhibit random behavior and they can be exploited as a source of randomness.

Indeed on the practical side, there are innumerable suggested or in-use generation methods for obtaining random sequences. All the efforts in this area can be roughly classified under two headings according to the utilized source to produce random sequences. One method is directly extracting randomness from a physical phenomenon in nature as mentioned above. The sequences generated in this way are called *true random numbers*.

Another way is to use a limited length of true random number sequence as a seed and produce much longer seemingly random sequences via deterministic algorithms. The output in this case is named as *pseudorandom numbers*.

Looking at the formalization efforts in the literature, one may come across with several theories grounded over mathematics and computer science, each working on randomness approaching through different notions like compressibility, patternlessness, etc. But, as it will be clear in the following sections, many of the theories have either serious limitations or they are not practical at all.

In this chapter, after a brief review of theoretical concepts defining randomness, we study the classification of random number generators, general requirements of them and evaluation criteria on them.

2.2 Theoretical Approaches to Randomness

The first known study for measuring randomness dates back to a century ago with Borel's normality property [18], and many other definitions and measures on randomness have been introduced since then. Below is not a detailed history of the works put forth in this realm, but a brief review of three prominent theories which talk about randomness of a finite sequence. These approaches constitute the theoretical effort to conceptualize (pseudo) randomness, yet none of the definitions stated below presents a constructive method for generating random sequences.

2.2.1 Information Theory

Information theory, which is rooted on the probability theory, is quantifying the randomness of a finite sequence looking at its *entropy*. A finite string of numbers is considered as the realization of a random variable of which outcomes could follow the values in the string. Each unit of information (state of behavior) is decided over a finite alphabet. So that the information in this discretized data could be measured in units and it is called entropy. Entropy is calculated over the probability distribution of states. If certain states occurs with high probability compared to the others, then the uncertainty decreases, in return randomness decreases. By following this logic, a random sequence is accepted as perfectly random when the information contents maximize, i.e. its entropy maximizes, which happens in case of the uniform distribution. Thus, this theory directly associates perfect randomness with the uniform distribution.

The information theory does not provide a deterministic procedure to produce random strings starting from a shorter truly random sequence [19], which means that the general practice of pseudorandom number generation does not fit with this theory. This is because deterministically produced sequence of pseudorandom numbers in no way symbolize a random variable in this framework, so we cannot meaningfully speak of a random variable with the uniform distribution.

2.2.2 Complexity Theory

This concept, which is mainly developed by the works of Solomonoff, Kolmogorov and Chaitin, views randomness of a data as its incompressibility or lack of pattern. In this view, a set of data with little randomness is considered to be more structured, and therefore it is expected to be generated via a short computer program. As randomness of data increases, the shortest computer program needed to generate the data will be more lengthy. To clarify the form of a "computer program", a fixed universal Turing machine must be specified.

This approach quantifies the complexity of a string as the length of the shortest program that can produce it. The randomness occurs at the extreme case where the length maximizes: a string is considered perfectly random provided that the string is shorter than any program that can generate it. (The shortest computer program would be at most as long as the string itself.) Unlike the first theory, this approach considers randomness as a property of the string itself depending on the specified universal machine. Measuring randomness in this way provides no practical guidance because there is no algorithm known for computing the shortest program which produces a particular string [20].

Just as in the first theory, this theory does not admit a pseudorandom sequence as random because, the fact that it is generated deterministically from a shorter truly random sequence directly implies failure in fulfillment of the required condition.

2.2.3 Computability Theory

The third approach, which is created with the works of Blum, Goldwasser, Micali and Yao, evaluates randomness in a different view. This theory does not support a quantitative approach, rather, it takes randomness of a string not as one of the string's intrinsic properties to be measured, but as a notion which is to be realized to the extent of the observer's ability. A string is considered random if it cannot be distinguished from a uniformly distributed sequence by any efficient algorithm. Here, two points need to be

clarified to better understand the idea: computational distinguishability and the bounds of efficiency.

Computational indistinguishability is central to the concept. Two objects are considered as equivalent if the observer's computational ability cannot distinguish them in any efficient procedure. Efficient procedures are most generally associated with polynomial-time algorithms, or algorithms with some other boundaries on computational resources. Therefore, a string is considered as random if it cannot be distinguished from the uniform distribution via any polynomial-time algorithm by means of computing resources of the observer. In order to decide whether a sequence is random or not, one should make an assumption on the computational power of the adversary or the observer.

This approach does not impose a strict measure on randomness or how to quantify it, rather it defines the randomness subjective to the observer. The current practice of producing pseudorandom sequences by stretching a shorter random seed does not fit with either of the two theories (2.2.1 and 2.2.2) mentioned above [19]. As opposed to the first two approach, generating a deterministically random sequence from a shorter true random sequence is achievable in this complexity theory.

2.3 Random Number Generator Classification

Definition 2.1. A *random bit generator* is a device or algorithm which outputs a sequence of statistically independent and unbiased binary digits [20].

In literature, various terms like 'random number generator', 'random bit generator' are generally used and occasionally they are being used interchangeably. As the former is not restricted to the binary scope, it admits of much wider range of values. Therefore one may take it as a more extensive term. However, in practice, a good bit generator can be used as a number generator as well [20]. For example, to produce pseudorandom numbers in the interval $[0, n]$, taking $\lfloor \log n \rfloor + 1$ bits and converting to integer works without problem. One way to eliminate the integers exceeding n is simply to discard them. With this claim, we also assumed that it is aimed to have random sequences with uniform distribution in both practices. Actually, in order to produce sequences with non-uniform distributions, the way to transform the uniformly distributed sequence into the desired probabilistic distribution. So basically, the basic instrument to produce a random sequence in any interval and with any distribution is a random bit generator with uniform distribution. In the rest of this text, 'random number ', 'random bit' and 'random sequence' are all used interchangeably to refer to 'random bit sequence'.

As we point out earlier there are two basic procedures for random number generation: using a deterministic method for producing pseudorandom sequences and exploiting a natural phenomenon for producing a true random sequence. A hybrid RNG mechanism consists of a combination of the both methods so that it can reach a higher speed than a TRNG and a higher entropy than a PRNG. Figure 2.1 maps out the types of RNGs explained in the following sections.

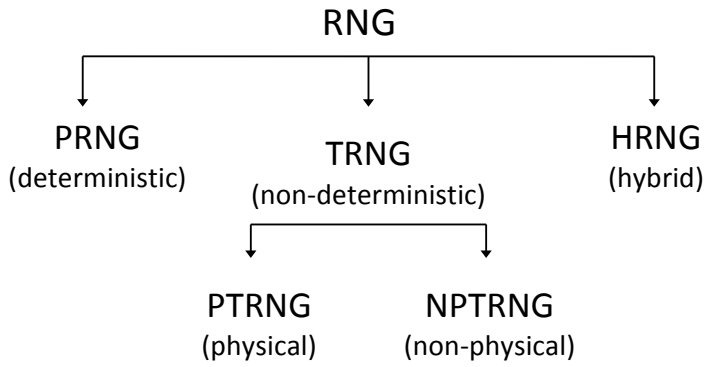


FIGURE 2.1: Classification of RNGs

2.3.1 Physical TRNGs

A physical true random number generator (PTRNG) extracts randomness from a physical phenomenon which is scientifically guaranteed to produce unpredictable outcomes. PTRNGs consist of a specialized hardware to sample from a physical source of entropy. The sources are chosen among the ones which are easy to digitize, i.e. connect to a computer such as the measurements on atmospheric noise, thermal noise, radiation, etc.

Physical TRNGs exploit unpredictable changes in the behavior of the element under inspection. For example, [21] gathers randomness using the changes in the amplitude of atmospheric noise. Likewise, [22] benefits from the unpredictability induced by the differences in time elapsed between the emission of particles during a radioactive decay. Use of hardware for digitizing the analog source is a slowing-down factor for PTRNGs. As given on their websites, the effective speed of production is about 300 bytes per second for [21] and 100 bytes per second for [22].

The analog input derived from the physical source is then converted to digital data so that a raw random sequence is obtained as seen in Figure 2.2. The sequences produced in this way are completely non-deterministic and irreproducible. However, this does

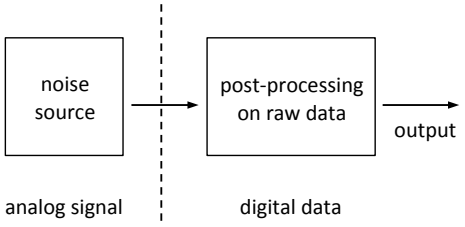


FIGURE 2.2: General Structure of a TRNG

not mean that PTRNGs produce 'true' or 'perfect' instantiations of random sequences. Typically, a filtering process is needed to follow the sampling to eliminate the patterns, such as long strings of zeros and ones, which distort the uniform distribution so that the final output appears more similar to a 'statistically perfect' random sequence than the raw random sequence does.

2.3.2 Non-Physical TRNGs

A non-physical TRNG (NPTRNG) differs from a physical TRNG in the type of entropy source it employs. Instead of a natural phenomenon that is scientifically proved to be unpredictable, NPTRNGs use more practical sources like system data of a PC or the information derived from human-computer interaction. Mouse movements of a computer user, air turbulence inside the disk drive and environment noise in an office recorded by a microphone might be entropy sources for NPTRNGs. Obviously, no theoretical justification is available here rather, unpredictability stems from the complexity in the inter-operation of several deterministic processes. These are cheap and fast methods to use in cases where physical TRNGs are too costly or too slow to provide randomness.

NPTRNGs can provide irreproducible and high level of randomness, however design details require special attention mainly for two reasons. First, TRNGs are generally used for cryptographic purpose. Since the entropy source is a computer component, the access to the source might be open to others in which case the prediction of the outcomes might be possible. Second, the quality of the entropy source should be well-analyzed since the behavior of a digital source depends on several factors. Assume that, similar to a physical TRNG using atmospheric noise, a non-physical TRNG is built to exploit background noise in a room. In such a case, some repeating noises (e.g., from the fan of a computer), may cause certain periodic patterns to appear in the output sequence, which in turn reduces the statistical quality of the produced sequence. Suggested designs must consider such factors that weaken overall quality.

Comparing two types of TRNGs, it can be said that the entropy sources of PTRNGs are more amenable to scientific analysis therefore more suitable for modeling TRNG mechanisms while non-physical TRNGs use rather environment-dependent sources which are not easily analyzable. On the other side, NPTRNGs are favorable for ease of availability and high throughput. A study on an NPTRNG [23], which uses the jitter of a ring oscillator as its entropy source, reports a throughput speed of 18.5 Mbps with post-processing on the raw sequence. The same study also states that their model satisfies the statistical requirements without any post-processing and its throughput goes up to 125 Mbps.

2.3.3 Pseudorandom Number Generators

A PRNG is a deterministic algorithm which takes a finite length truly random bit string as input and produces much longer seemingly random bit sequences. The input to the PRNG is called *the seed*. Here, the word 'deterministic' refers to any method which produces identical outputs given the same input.

The sequences generated by PRNGs are not random in the sense that they are completely predetermined; but the algorithms designed for this purpose perform so good that their outputs effectively appear random. So much so that some PRNG outputs have even better statistical quality than truly random sequences.

2.3.3.1 Generic Design of Pseudorandom Number Generators

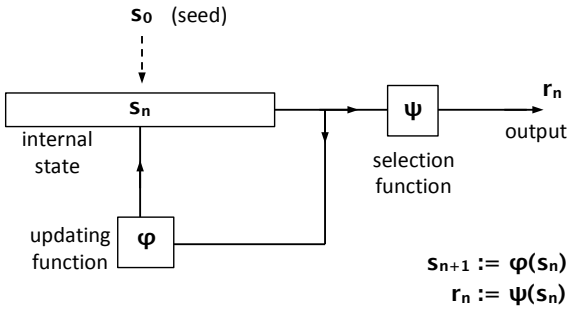


FIGURE 2.3: General Structure of a PRNG

Figure 2.3 illustrates the functional model of a PRNG, whose activity could be summarized as a two-step process. Let us name this two main functions as φ and ψ as seen in Figure 2.3. *The internal state* is stored in memory and contains the data to be modified at each time step.

The updating function φ continuously updates the content of the internal state in a recursive way such that $s_{t+1} = \varphi(s_t)$ where $t \in \mathbb{N}$ and s_0 is the seed. At time $t = 0$, the internal state is derived from the seed s_0, which is provided by the user.

The source of entropy is the seed assuming the PRNG algorithm being public. Indeed, this assumption corresponds to the actual case, such that the unpredictability of the whole output sequence becomes reduced to the unpredictability of the seed. It is, therefore, recommended to obtain the seed value from a truly random sequence to increase entropy where unpredictability is vital as in cryptography and gambling applications. This also means that in the cases where secrecy is required, it is the seed which must be secured.

The seed has a life. Since it is a finite entropy source it must be renewed periodically. The initial state may also be derived from a combination of the seed and some other information, for example, a nonce or a personalization string. Note that, disclosure of the additional information does not risk the security of the PRNG as long as the seed is protected properly.

ψ is *the selection function.* At each time point t, ψ takes the internal state s_t as input and produces the output sequence r_t. ψ can be selected to output one bit at a time as well as multiple bits at each time step. The selection function is needed especially to prevent the correlation between successive initial states from appearing in the output string.

PRNGs are favorable to TRNGs in that a large amount of output can be obtained in a short time with low cost. The fact that there is no need for hardware devices to convert analog signals is the major advantage for high-speed and low-cost generation. On the other hand, PRNGs are inevitably periodic, meaning that the output sequence repeats itself. But good algorithms have very long periods that this practical drawback can be minimized. Therefore PRNGs are widely used for applications, such as simulation and modeling, where high throughput and reproducibility are intended. But they are not suitable for applications where unpredictability is the key to effectiveness.

2.3.3.2 Cryptographically Secure Pseudorandom Number Generators

It can be said that the practical conveniences that make PRNGs preferable are also the ones that make them more vulnerable to security attacks which could exploit their deterministic nature. It should be noted that even though good PRNG algorithms exist, they are not suitable for every application that needs randomization. In cryptographic

procedures, either truly random sequences are used for their unpredictability or a special class of PRNG, named *cryptographically secure PRNG* (CSPRNG) is employed for random number generation.

Definition 2.2. [24] A CSPRNG is an algorithm G that, upon receiving a random number (the seed) i as input, outputs a sequence of pseudorandom bits $a_1, a_2, a_3, \ldots$ with the following properties:

1. The bits a_i's should be easy to generate such that each a_i should be generated in polynomial time in the length of the seed.

2. The bits a_i's should be unpredictable. Given the generator G and $a_1, a_2, ..., a_s$ the first s output bits, but not the seed i, it should be computationally infeasible to predict the $(s+1)^{\text{st}}$ in the sequence with better than 50-50 chance. Here, s is polynomial in the length of the seed.

Definition 2.3. [20] A PRNG is said to be passing all *polynomial time statistical tests* if no polynomial-time algorithm can correctly distinguish between an output sequence of the generator and a truly random sequence of the same length with probability significantly greater than $1/2$.

The second condition of the definition 2.2 is called *next-bit unpredictability*. Next-bit unpredictability can be set up on such a scenario: An adversary who get the first s bits of a random sequence has only negligible probability over $1/2$ in predicting the $(s+1)^{\text{st}}$ bit. Deciding over negligibility could be done over assuming an adversary with polynomial time computation power [25]. Obviously, predicting the next bit or previous bit has the same difficulty from the statistical viewpoint. In [26], Yao proved that a PRNG passes the next-bit test if and only if it passes all polynomial time statistical tests.

The security of CSPRNGs relies on cryptographic primitives or mathematical problems that are assumed to be intractable. Some examples of such problems can be listed as the symmetric-key algorithm AES, the hashing algorithm SHA, the intractibility of number-theoretic problems such as the quadratic residue problem as in Blum Blum Shub algorithm and the discrete logarithm problem as in Blum-Micali algorithm. Based on the intractibility assumptions, the security properties of CSPRNGs can be proved. On the other hand, since the problems that the generators rely on are assumably intractable, advances in science and technology leading a possible solution to the problems could imply a compromise in the security of CSPRNGs.

CSPRNGs may be designed according to various objectives considering their functionality. They must be chosen according to the practical needs. In some practices, like creating a nonce, the main focus is the uniqueness of the generated string rather than its entropy. On the other hand, generating an encryption key mainly necessitates high entropy. If the security is more sensitive issue, then a TRNG might be more suitable solution than a CSPRNG.

2.3.4 Hybrid Random Number Generators

The hybrid RNG (HRNG) designs include a combination of deterministic and non-deterministic methods. The deterministic part is as known from a PRNG mechanism and the non-determinism comes from an additional entropy source. Figure 2.4 illustrates a good abstraction of such a design.

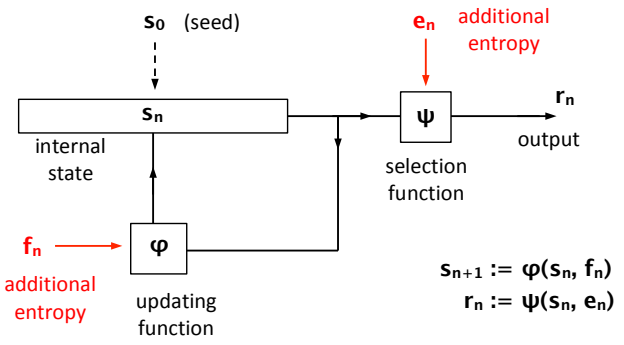


FIGURE 2.4: General Structure of a Hybrid PRNG

PRNGs are advantageous for their speed and statistical quality but in the case of a pure PRNG, the entropy source is limited due to a finite initial state. Similarly, some shortcomings of a pure TRNG is known to be the cost and the low speed of generation. A hybrid RNG admits additional input at intermediate steps so that extra entropy is introduced to the system. With this change on the design, an increased security level is intended. However, whether the security increases or not depends on the unpredictability and entropy of the additional input. Even if the additional input is a constant string (that is, it has a weak randomness quality), the level of security will not drop below the security level of the pure correspondent of the hybrid version.

In the literature, HRNG designs may also be classified as hybrid PRNG and hybrid TRNG. In such a case, the distinction between hybrid PRNG and hybrid TRNG is typically made according to the type of challenge that the strength of the security mainly relies on [27].

TABLE 2.1: RNG suggestions for various applications

Application	**Recommended RNG**
Lotteries and Draws	TRNG
Games Gambling	TRNG
Random Sampling (e.g., drug screening)	TRNG
Simulation and Modeling	PRNG
Security (e.g., generation of data encryption keys)	TRNG

2.4 A Comparison between True and Pseudo RNGs

As stressed earlier, each type of RNG has its own utility, drawback and purpose. PRNGs are favorable for the tasks which require only good statistical quality and high throughput. PRNGs owe their advantage to the fact that they run solely on software level. This makes PRNGs both a cheap and a fast way of producing randomness. Shortcomings of the PRNGs are on the unpredictability side. Since PRNG mechanisms are deterministic and public, all the burden of unpredictability is on the seed, because the seed is the only secret part. Also, seed is the only source of entropy, which means that entropy source of a pure PRNG is limited. Moreover pure pseudorandom sequences are reproducible and inescapably periodic. Though reproducibility cannot be strictly said to be a disadvantage and periodicity is curable to a large extent, these are not the properties intended for cryptographic use. In case of CSPRNGs, security can be achieved in expense of sacrificing some of the production speed. All in all, PRNGs provide practical security and they are vulnerable to various kind of attacks.

TRNGs are deemed quite inefficient according to throughput level compared to PRNGs because use of hardware devices to sample from an analog source unavoidably causes impractical, slow and expensive production. However, true random sequences are non-periodic and irreproducible. Most importantly, truly random sequences provide theoretical security, which is the best reason for using TRNGs.

Table 2.1 is taken from [21] in order to give the general account of selection schemes. It helps to understand which type of RNG accommodates to which purpose.

2.5 General Requirements on Random Number Sequences

With no assumption on practical purpose, a random sequence is expected to contain all possible values with equal probability and each part of the sequence must be independent from its predecessors and successors. However, this represents an ideal case and not all applications of randomness requires that high level of quality. Moreover, it is quite

nontrivial to prove that a RNG meet these requirements. To provide an insight into which quality serves for which purpose, we can go into a formulation of the general requirements. The first requirement (R1) is a sine qua non for all RNGs:

R1: *A random sequence should show no statistical weaknesses* [27].

The statistical quality of a RNG is generally examined via statistical test suites. A statistical analysis over a binary random string involves an analysis of distribution of 1s and 0s as well as distribution of substrings with more than one-bit-length, linearity of the sequence, correlatedness. Therefore, the satisfaction of R1 is expected to prevent replay attacks and correlation based attacks.

A good statistical quality alone may be suitable for random sequences used in stochastic simulations and Monte Carlo integration but not in more sensitive procedures like key generation or digital signature generation for security protocols. A sequence with good statistical quality might be well predictable through a detailed inspection. In the simplest case, the outcome of a linear function could be wholly recovered via mathematical inspection when obtaining a certain finite part of the sequence. In general, making the assumption that an adversary has some part of a hidden random sequence is not unreasonable. Assume person A uses 256 bits of the random sequence r as a public key in an asymmetric encryption protocol. When A sends this 256-bit public key to a message receiver, some part of r will become revealed. So the remaining part of r will no longer be secure to be used as a key if the generator's output is predictable. For such sensitive applications, unpredictability of the sequence is of vital significance.

R2: *With the knowledge of a subsequence, it must have only a negligible probability to obtain the predecessors or successors of the sequence than it has without the knowledge of any subsequences* [27].

R2 is a generalized condition on unpredictability of a sequence given some part of that sequence. The following requirements elaborates on the circumstances that could lead the exposure of any bits during the generation procedure. R3 and R4 are stated as they appear in [27].

R3: *Even the knowledge of the internal state shall not allow one to practically compute 'old' random numbers or even a previous internal state or to guess these values with non-negligibly larger probability than without the knowledge of the internal state* [27].

R3 is called *backward upredictability* and especially needed to keep the seed secret. It requires that the correlation between the seed and the output sequence must be undetectable.

R3 goes beyond predicting the output sequence, i.e., the output of the selection function ψ as shown in figure 2.3, and draw attention to the possible information leakage arising from predicting the internal state s_t. To meet R3, a RNG must have an uninvertible updating function φ. Here, uninvertibility should not be taken in the strict sense, but in the sense that recovering or predicting the preimage of any state s_t (which is equivalent to $\varphi(s_{t-1})$ and may not be unique) must have a negligible probability within the capability of a polynomial-time adversary.

R4: *Even the knowledge of the internal state shall not allow one to practically compute the next random numbers or to guess these values with non-negligibly larger probability than without the knowledge of the internal state* [27].

R4 is called *forward unpredictability*: With a secret seed and a public algorithm, the future bits must be unpredictable even with the knowledge of previously generated bits. A pure PRNG cannot meet R4, because its output is completely determined by the internal state and the algorithm. Remember that the algorithm is always assumed to be public. R4 comes into question only in the case of a hybrid PRNG. Adding a high-entropy additional input to the system helps fulfill R4.

Considering a pure TRNG, there is no question as to whether R3 and R4 are fulfilled. The fulfillment of R2 resolves predictability problem for a TRNG. The predictability of a TRNG might happen if the source or the hardware used leaks information. A CSPRNG is expected to fulfill all four requirements but for the other RNGs, it is decided according to the application necessities as to which requirements should be fulfilled.

2.6 Evaluation Criteria of PRNGs

As the requirements of RNGs are summarized as statistical quality and unpredictability, the evaluation criteria of RNGs center around these two concepts, as well.

Predictability is directly associated with the security of a RNG. Typically, the security assessment of a pure PRNG is made based upon two points: the complexity of the intractible problem that a PRNG algorithm relies on and the size of key space, which determines the number of all possible guesses to find the seed or an internal state with

non-negligible probability. These two components together determine the cost of a brute-force attack to be aimed for a pure PRNG in terms of time, depending on computational power. However, security measures may change over time, a PRNG may have become less secure as new attacks to the challenging problem lying at the heart of PRNG become available. In case of a hybrid PRNG, the size of key space will be larger due to the added entropy to the system.

On the TRNG side, unpredictability is directly tied with the entropy of the random source that is used by a particular TRNG. Contrary to PRNGs, there is no risk of unstable security bounds once the entropy source is analyzed thoroughly during the design phase of the TRNG. So, it is more feasible to use a truly random sequence for encrypting data which is to be protected over a long term. A detailed information over security (predictability) analysis of RNGs can be found in [27].

In order to evaluate the statistical quality of a RNG, a commonly used method is to apply statistical test batteries. Statistical tests are not to justify the sufficiency of a RNG, but to provide a probabilistic interpretation about the behavior of a RNG by examining its output. As pointed out earlier in Section 2.3, by transforming the output of a uniformly distributed random bit sequence, it is possible to generate any random sequence within a certain interval and with the desired distribution. Therefore, the main focus of statistical tests is to detect deviations from the uniform distribution.

2.7 Statistical Test Suites

We expect a truly random sequence to exhibit certain features (e.g. including equal amount of 1s and 0s) as well as to lack some other features (e.g. periodicity). This means that there is a general idea of how an ideal random string should behave, this enables us to determine a theoretical reference point as to show how a deterministically generated string should be in order to look random. When expected properties from a truly random sequence are clearly specified, each of the properties can be checked over a candidate string, say R, so that a comparison can be made between R and a truly random sequence. However, it is not possible to make a strict statement about randomness of the candidate string R, since there is no limit to the extent of features to be checked for. That is, there are infinitely many possible tests for randomness therefore, no finite set of tests can be considered 'complete' to reach a strict judgment. Statistical test suites are designed for the purpose of bringing together a list of properties of an ideal string as extensively as possible.

One of the early attempts to build up a set of preconditions for a binary string to appear random came from Golomb [28]. His study, known as *Golomb's Randomness Postulates*, contains three conditions, two of which are involved in the distribution of substrings and the third one concerns the autocorrelation in the string.

Later, Knuth presented a larger set of statistical methods in his book [29] to check for uniform distribution of binary strings. First published in 1969, this work is one of the most popular and highly referenced one in its realm. However, it can no more said to be an effective measure because it has been found out in time that some of the tests approves weak-quality strings as random enough.

A comprehensive test suite of 15 statistical tests (named *DIEHARD*) was developed by Marsaglia and his work including the codes implemented in C was published on a CD-ROM in 1995 [7]. With the advancement of computer usage in scientific applications, better-quality random numbers became important for sophisticated works in physics and mathematics. DIEHARD test suite responded to emerging needs in the area and became an important platform for development of even better statistical test suites.

Crypt-XS and ENT [8] are also among the outstanding examples of randomness tests but we will focus on the Statistical Test Suite developed by National Institute of Standards and Technologies (NIST). This study uses outcomes of the NIST Test Suite as the indicator of statistical quality.

NIST Test Suite was intended to serve as a comprehensive and up-to-date test suite and created by using the previously worked examples as a scaffold. The test is quite stringent and passing all the test is a strong indication of a good statistical quality even for CSPRNGs. It can be used for all type of RNGs. In the next section, purpose of the individual tests are presented. A detailed description of the tests with their mathematical background can be found in [6].

2.8 NIST Test Suite

2.8.1 Hypothetical Testing

If randomness is taken as a probabilistic property rather than all-or-nothing property, the characterization of a random sequence could also be made in probabilistic terms. There are 15 statistical tests in the NIST test suite and each of them is designed to verify/reject that a certain property of randomness is observed on the string R being tested.

TABLE 2.2: Possible outcomes of a hypothetical testing.

	TEST RESULT	
ACTUAL CASE	**Accept H_0**	**Accept H_a**
Sequence is random (H_0 is true)	No error	Type I Error
Sequence is not random (H_a is true)	Type II Error	No error

In hypothetical testing method, a test is conducted to experiment a *null hypothesis* H_0. The result of the test is either accepting or rejecting H_0. If H_0 is rejected, this means *the alternative hypothesis* H_a, which is specified with reference to H_0, is accepted as true. For this operation, H_0 is chosen as "The string R being tested is random." and H_a is "The string R being tested is not random."

Test procedure goes as follows: The sequence R to be tested goes through calculation steps and its statistical values are obtained. Then this value is compared to the reference value which must be determined in advance according to the desired properties. These reference values are calculated by mathematical methods and they shape a theoretical target to achieve. In the comparison step, *the critical value* comes in question.

Critical value is a boundary which is exceeded by the test statistics with a very low probability if H_0 is true [6]. If the statistics of R reaches the reference value then H_0 is accepted. Not only the reference value but also other values very close to it might be satisfactory to accept H_0. Here, the critical value determines the scope of acceptable values to admit that H_0 is correct.

As a natural consequence of probabilistic approach, there is always a risk of false decision. If the reality and the test results are in contradiction then a false result will turn up. As shown in the table 2.2 two kinds of errors are probable.

The probability of having a Type I error is called *level of significance* and generally denoted by α. It indicates the probability that the test result reports that the sequence being tested is not random when it is actually random. α is set to a fixed value prior to start of the test by the practitioner. Likewise, the probability of having a Type II error is denoted by β. It indicates the probability that the test result accepts the sequence at hand as random when it is actually non-random. β is not fixed. α, β and the sample size n are dependent such that specifying two of them directly determines the third one. Generally, α and n are set before the test starts then the critical value is determined so as to minimize β, in order to prevent the test results to offer a bad-quality sequence as random.

The underlying logic behind the parameter specification and the overall reliability of the testing method is as follows: A truly random sequence is used to set the reference values

then the critical value is determined. If the sequence R to be tested is actually random, then the probability that the test result shows the opposite is very low, like 1% (namely it is α percent). If the statistics of R exceeds the critical value, H_0 is rejected because exceeding the critical value while H_0 is true is quite unlikely. Accordingly, H_0 is either false or at least doubtful.

After the test statistic of R is compared with the critical value, a *P-value* is calculated. *P-value* gives the probability that a perfect random number generator produces a sequence which is less random than R. A zero *P-value* indicates that the tested sequence R is completely non-random. At the other extreme, when *P-value* is one, the sequence R turns up to be perfectly random.

P-value summarizes the statistical knowledge gained through testing in a single value and its comparison with α directly tells the test result. If *P-value*$< \alpha$, H_0 is rejected, otherwise H_0 is accepted as true.

2.8.2 Tests in NIST Test Suite

This section aims to give brief descriptions of each test and the deficiencies detected by them but not includes mathematical background of the test designs. Detailed descriptions are available in the documentation of the test suite [6]. In this section R is used to denote the sequence being tested and n is used to denote the length of R in bits. They are given as input to all the tests in the NIST test suite. This information will not be repeated for each test description below. When a test requires any other input which is to be determined by the practitioner, it is stated under the heading "Input Details".

2.8.2.1 Frequency Test

Description: A binary truly random string is expected to contain approximately equal amount of ones and zeroes. Frequency test checks if the ratio of ones to the length of R is approximately 1/2. The sequences with too many ones or too many zeros fail the test. The remaining tests in the NIST Test Suite are not conducted if R fails in this most basic one.

Input Details: To obtain a healthy statistical result, n should be at least 100.

2.8.2.2 Block Frequency Test

Description: The test divides R into M-bit-long non-overlapping subsequences then checks whether the frequency of ones is nearly $M/2$ within each block. If the Frequency

Test would be said to make a global check of frequency, then Block Frequency Test can be said to do the same locally. This property is required for R with reference to the assumption that every subsequence of a truly random sequence is also truly random. The strings with approximately equal amount of ones and zeroes overall but with irregular distribution (e.g., consisting long chains of zeroes or ones) are rejected by this test.

Input Details: M: length of subsequences in bits. M should be at least 20 and number of subsequences should be no more than 100.

2.8.2.3 Runs Test

Description: A *run* is an uninterrupted chain of identical bits which is bounded in both ends by the opposite bit. This test counts the number of runs of each length and compare if it is as expected from a random sequence. It also checks the rate of oscillation between two kind of runs. (Oscillation is used to refer a change from a run of ones to a run of zeroes and vice versa.) An ideal random sequence of length n is expected to have $n/2$ oscillations. Low rate of oscillation implies low number of runs and long chains. This might be due to local sequential dependencies therefore reduces randomness.

2.8.2.4 Longest Run of Ones in a Block

Description: The test divides R into M-bit-long non-overlapping subsequences then checks if the length of the longest run of ones is as expected from a random sequence. The reference values are preset and given in a look-up table. There is no other check for longest run of zeroes because a defect in the longest run of ones implies a defect in the longest run of zeroes, as well. For the sake of the independence of individual tests in the test suite, checking only one of them is sufficient.

Input Details: n should be at least 128. The block size M is set by the program in accordance with n.

2.8.2.5 Binary Matrix Rank Test

Description: The sequence R is divided into parts to create 32 by 32 matrices as many as possible. Each 32-bit-long subsequence is placed horizontally in a matrix as a row and the rank of the resulting matrices are calculated. During the matrix formation step, the unused bits at the end of the string are discarded. If a linear dependence exists in the string R, an unlikely case for a random sequence, then the rank of the matrices will be

low. The test code contains pre-calculated rank distribution estimates that is expected from a random sequence. Deviation from the reference distribution breeds failure.

Input Details: The number of matrices created should be at least 38. This implies a lower bound of 38,912 bits on the length n of R.

2.8.2.6 Spectral Test

Description: Discrete Fourier spectrum of the sequence R is computed and peak points in the spectrum are examined to detect the periodic features of R. If the sequence cannot pass this test, this implies that repetitive patterns which are close to each other exist in the sequence and therefore the sequence is not likely to be random.

2.8.2.7 Non-overlapping Template Matching Test

Description: The sequence R to be tested is first divided into 8 blocks of the same length (the user has no choice on partitioning). Pre-defined strings of length m, for various values of m, are present in the test code. Test practitioner selects one of them by specifying the length m, then it is determined as the target string. The target string is searched through 8 partitions to see how many times it appears in each of them. Searching on partitions is carried out in the following manner: A window of length m is sliding over the sequence by 1 bit at a time. Each time the part residing into the window is checked whether it matches to the target string. If it does, counter of that partition increases by 1 and the next time window slides by m bits. If not, then slides by 1 bit and continues in this way until the window reaches the end of that partition.

Depending on m, several target strings are available and the test is repeated for each of the target strings. A regular distribution of the target strings in 8 partitions are expected to pass the test.

Input Details: m: the length of the target strings in bits. In the test code, templates are designed for m values up to 10. 9 and 10 are recommended values for m. Also, the number of blocks, which is fixed to 8 in the test code, can be changed to another value less than 100.

2.8.2.8 Overlapping Template Matching Test

Description: To a large extent, this test works in the same way with Non-overlapping Template Matching Test. The only operational difference is that the window slides by

not m bits but 1 bit if the string in window matches to the target string. There are also some changes in parametrization. Partition size is fixed to 1032. In all partitions, the test looks for the runs of ones of the specified length m and compare the results to the expected values.

Input Details: n is recommended to be at least 10^6 bits.
m: the length of the target strings in bits. In the test code, templates are designed for m values up to 10. 9 and 10 are recommended values for m. Also, the number of blocks are set to 968, which explains the lower bound on n.

2.8.2.9 Universal Statistical Test

Description: This test is based on the assumption that random sequences are not compressible. As an indicator of compressibility, distances between repetitions of all possible strings of length L are examined. First Q blocks of length L are used to make initialization for distance calculation. Then, $\log_2$ distance values are summed up to compare to the reference values supplied in a look-up table. The reference values include expected value and variance of the sum which are calculated for varying L values. If there are so many repeating patterns in R, then the sum will be much higher than the expected value. Consequently, the string R will fail the test for it is found out to be compressible to a large extent.

Input Details: Parameter names are different than the other tests because this test was originally taken from another test suite [30]. The original names of the parameters mentioned in [30] are used in this document, too.

L: block size in bits. The test analyzes the repetition pattern of every possible string of length L. L must be chosen between 6 and 16. Other parameters are initialized accordingly.
Q: number of blocks to use at initialization step.

2.8.2.10 Linear Complexity Test

Description: Linear Complexity Test works on the idea that the shorter LFSR producing a sequence gets, the more far away the resulting sequence gets from perfect randomness. The string to be tested is first divided into blocks of length M, then each block is subjected to Berlekamp-Massey algorithm [20] to find out the shortest LFSR that produces it. Then the distribution of the complexity values are compared to the expected values. If LFSR of a block is short, this implies a strong dependence on the string and causes failure.

Input Details: M: the length of blocks in bits.

2.8.2.11 Serial Test

Description: A random sequence is expected to contain approximately the same amount of each possible string of length m. To check for this property, the sequence to be tested is searched in an overlapping manner (each time an m-bit window slides by 1 bit). The frequencies of 2^m m-bit strings are determined and compared to the theoretical target.

Input Details: m: the length of blocks in bits.

2.8.2.12 Approximate Entropy Test

Description: This test focuses on the frequencies of m-bit and $m+1$-bit strings to find out joint distribution for m-bit strings. An approximate entropy depending on m is calculated in intermediate steps. Small values of the approximate entropy imply regularity while high values mean strong fluctuations. A small entropy value is an indicative of non-uniform joint distribution, which is also a good reason to fail the test.

Input Details: m: the length of blocks in bits.

2.8.2.13 Cumulative Sums Test

Description: The binary sequence to be tested is converted to $(+1, -1)$ sequence such that zeroes in the sequence are changed to -1 and the ones stay the same. Over the new, adjusted sequence, partial sum is computed and the absolute value of the partial sum at each step is checked to find out its maximum. Note that all the tests in the test suite is conducted provided that the sequence to be tested passes the Frequency Test. That is, the overall sum of the adjusted sequence does not substantially deviate from zero. So, a high value of the maximum indicates that, too many zeroes or too many ones are present at least at one place of the sequence. So, a local non-uniformity can be detected in this way and the sequence fails the test.

2.8.2.14 Random Excursions Test

Description: The sequence to be tested is converted from $(0, 1)$ to $(-1, +1)$ basis and partial sum sequence is formed to find out number of *cycles*. A cycle is a subsequence of the partial sum sequence which starts and ends with zero and includes no other zero

value inside. Over a length of 10^6-bit-long sequence, at least 500 cycles are expected, otherwise the sequence directly fails the test. The values in a cycle indicates how much the partial sum deviates from zero. Numbers with high absolute value and long cycles are marks of too many zeroes or too many ones in the original string. A statistics is obtained examining the contents of each cycle so that sequential dependencies in the original sequence can be detected.

The test consists of eight parts: It calculates the distribution of each of the values $-4, -3, -2, -1, 1, 2, 3, 4$ appearing in the partial sum sequence. And the test mainly focuses on the cycles which visit these states and compares their amount to the expected values.

Input Details: n is expected to be 10^6 bits.

2.8.2.15 Random Excursions Variant Test

Description: The sequence to be tested is converted from $(0, 1)$ to $(-1, +1)$ basis and again, the partial sum sequence is computed to find out number of cycles. The test consists of eighteen parts: It calculates the distribution of each of the values from -9 to 9 (excluding zero) appearing in the partial sum sequence. For a random sequence, the cumulative sum should not exhibit a substantial deviation from zero. The number of visits to each of these states are examined and compared to the expected values.

Input Details: n is expected to be 10^6 bits.

Chapter 3

Cellular Automata

3.1 History of Cellular Automata

The term Cellular Automata (CA)[1] started to appear in scientific literature by 1950s, although the history of conceptually similar works may be traced back to a few decades earlier. The field, in the way it is known today, has essentially begun to be shaped by the works of von Neumann and Ulam in 1940s. Since then, CA has been a prime topic for discussions and research on self-organization and complex systems. Along with its self-organizing nature, its underlying simplicity was making it a powerful tools for conceptualizing complex systems in nature. Therefore, CA has become a useful mathematical tool for interpreting dynamical behavior of large number of variables undergoing simple local interactions.

Looking at the early studies, it is possible to say that the interest for CA has been mostly kindled by the efforts to find answer to the computation-theoretic problems. Unlike the generic automata, which are generally used for designing finite and the simplest model that fulfills a specific purpose, CA allows designing infinite models. That makes it a quite adequate tool to think over theoretical subjects such as Turing Machines and computation universality [31].

The development of the notion has gone through certain stages, which were triggered by prominent studies of von Neumann, Conway and Wolfram. As the experimental and theoretical inspections –mostly in the realm of mathematics and computer science– suggested, CA is seen to have an intrinsic potency justifying this interest and finds use in several applications of chief scientific areas such as physics, biology and chemistry. However the field still lacks a solid theoretical construction which reveals its strengths

[1] Throughout this text CA is used to abbreviate both singular and plural forms: cellular automaton and cellular automata.

and perhaps is waiting for an intuitional leap that help go beyond the current dynamical and complex system theories.

3.1.1 von Neumann's Work

von Neumann was the first to attract a great attention to CA. What motivated von Neumann to study CA was a question he took up about biological evolution in nature. von Neumann's aim was designing a deterministic model which can display the necessary logical interactions to simulate the evolution in living organisms.

With this motivation, he set out to construct a model capable of self-reproduction. Self-reproduction might be summarized very roughly in the way that it is the quality of a dynamical system which enables the ability to copy itself.

von Neumann was believing that a continuous model is more powerful than a discrete one and only a highly complex model could be capable of representing the complexity in a natural phenomenon. Therefore he first planned to construct a model in continuous domain for his study.

Ulam, a Russian mathematician studying discrete applications of mathematics to biology, suggested von Neumann using a discrete model to represent the transformations in nature rather than a continuous one [32]. Taking advantage on that suggestion, von Neumann decided on a two-dimensional CA arranged on an infinite-size grid with 29 states per cell. That work would be the first CA design which is proven to be capable of self-reproduction. This was also the first discrete model shown to be a universal computer.

von Neumann's study initiated and substantially shaped the growth of the field. It can be said that CA had been characterized by discrete structure with his study and had become known for its ability to simulate complex systems in nature. Such a model was certainly a valuable one since its discrete structure provides an exact computability [10]. Because, analyzing the behavior of complex phenomena in nature necessarily requires computer simulations and a discrete model is quite useful for it causes no perturbation due to approximations or round-off errors on digital environments.

Since von Neumann's design was not claimed to be the simplest universal machine, it also triggered an effort to discover whether simpler designs are possible. Actually von Neumann himself had realized that this was possible during his study. Subsequently, CA was examined with changes on its parameters as in von Neumann's study and there have been created similar discrete, self-reproductive CA examples in the following period.

CA had also been subjected to rigorous mathematical and physical analysis. During the period of 1960s and 1970s, CA has become diversified up to various parameters.

The resulting constructions were scrutinized in the context of chaotic and dynamical systems, phase transitions and entropy in physics. Mathematicians generally connect CA with differential equations as well as studying basic notions like invertibility and completeness of a certain set of CA rules (i.e. CA functions) over a configuration space. On the information theory side, CA's ability of translating information was the point of interest.

3.1.2 Conway's *Life*

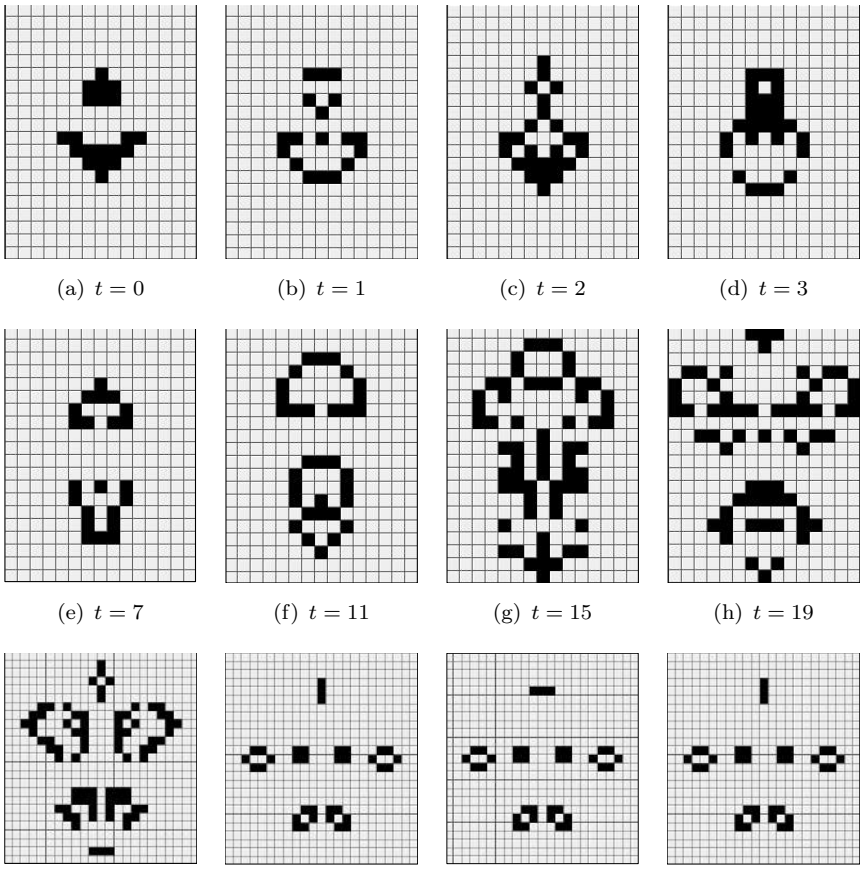
Nearly twenty years later than its first rise with von Neumann, CA gained a great popularity by John H. Conway's work. Conway was a mathematician researching in pure math. As part of his interest in recreational mathematics, he was studying on two dimensional CA models. His original objective was to find out a self-reproducible universal machine design simpler than von Neumann's example. His attempt became successful and he constructed a self-reproductive model which was capable of simulating a universal Turing machine. He named his model as "Game of Life". It was published in a science magazine in 1970. Unlike von Neumann's construction, this one had a fairly simple configuration as described below.

Game of Life was analogically designed to conceptualize the evolution of a living society in nature. Its environment is an infinite orthogonal grid with square cells. Each cell can exist in one of two states: alive or dead. In visual representation, alive cells are shown in black and dead cells in white. Each cell's future is decided by its eight neighbors: four orthogonally and four diagonally adjacent cells.

The rules guiding the evolution in Life are as follows:

1. An alive cell keeps alive if it is surrounded by 2 or 3 live cells.
2. A live cell dies from isolation if it has less than two neighbors alive.
3. A live cell dies from overcrowding if more than three neighbors are alive.
4. A dead cell comes in to life if exactly three neighbors are alive.

Figure 3.1 shows an evolution in *Life* starting from the initial state (a). Each cell in a state is subjected to the given rules to generate its next state. Once the initial state is set out, Life starts its evolution. Applying the rules simultaneously to each cell in a state generates the next state in time, i.e. the next generation, and it continues until reaching a termination state.



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$ (d) $t = 3$

(e) $t = 7$ (f) $t = 11$ (g) $t = 15$ (h) $t = 19$

(i) $t = 22$ (j) $t = 45$ (k) $t = 46$ (l) $t = 47$

FIGURE 3.1: Figures show an evolution in Life starting from the initial state (a). The configuration enters into a repetitive trend starting from $t = 45$. Figures are generated via [33].

Termination state is all white if society completely dies out (which happens rarely). It may settle down to a stable state which remains unchanged in the rest of the time or there may be a set of states repeating themselves or, termination state may consist of moving objects which grow without limit.

Following its publication in a science magazine, Game of Life gained a great popularity among computer programmers. Many variations with different environments (for example, on a hexagonal grid) and different rules specifying death and birth conditions or defining more than two states per cell were created. Some of them shown to be self-reproductive. Interesting patterns occuring in configurations were searched out and examined thoroughly in science magazines.

What made Life so interesting and popular is that it was then one of the simplest examples of self-organizing systems with these properties. However, this popularity did not lead to a systematic exploration of CA dynamics but mostly stayed in a recreational level [34]. Nevertheless, it arose questions in minds as to which extent CA rules are

capable of simulating complexity in the nature. And, an extensive exploration of CA had waited until Wolfram undertook the subject in 1980s.

3.1.3 Wolfram's Work

von Neumann's motivation was to find a specific example satisfying certain criteria, namely universal computation and self-reproduction. On the other hand, Conway first designed a CA rule which shows interesting behaviors then set out to explore its properties and finally it was shown to fulfill Neumann's criteria by contribution of many others. What Wolfram did was a more integrated study. He started off going over the structures generated by one dimensional rules and end in a classification over the elementary rules and extended his findings over larger rule spaces.

Wolfram's first important study about CA [35] came into view in 1983. In this paper, he studied a class of one dimensional CA which he named as *the elementary CA rules*, choosing to start with the most simple type of CA before delving into the big picture. The main emphasis in [35] is how simple rules create unexpectedly complex structures as a result of self-organization.

Based on the level of disorder in the behavior of CA rules, Wolfram proposed a classification that recognizes four groups within the elementary rules. This classification can be extended to cover all one-dimensional CA rules, as well. The classes are the following:

- **Class 1:** The rules in this class are the simplest in behavior and converge fastest among the four to a stable state. They converge to a homogeneous state, in that, they show a resemblance to the continuous system with fixed-point attracting states. Some examples to the rules in this class are Rule 32, Rule 160 and Rule 232[2].
- **Class 2:** Rules enter into a periodic regime in a longer time compared to the first class and indicate predictable behavior. Their analogous counterpart in continuous systems are the ones with continuous limit cycles. Examples are Rule 4, Rule 108, Rule 218 and Rule 250.
- **Class 3:** Rules in this class do not converge to any stable state, but continually produce similar kind of patterns representing a random appearance overall. They resemble to the continuous systems with strange attractors. Example rules are Rule 22, Rule 30, Rule 126, Rule 150 and Rule 182.

[2]See the section 3.4 for naming conventions

- **Class 4:** The fourth class produces structures propagating locally and a complicated interaction appears where they cross each other. There is no obvious analogue in continuous systems to the fourth class [10]. An example is Rule 110.

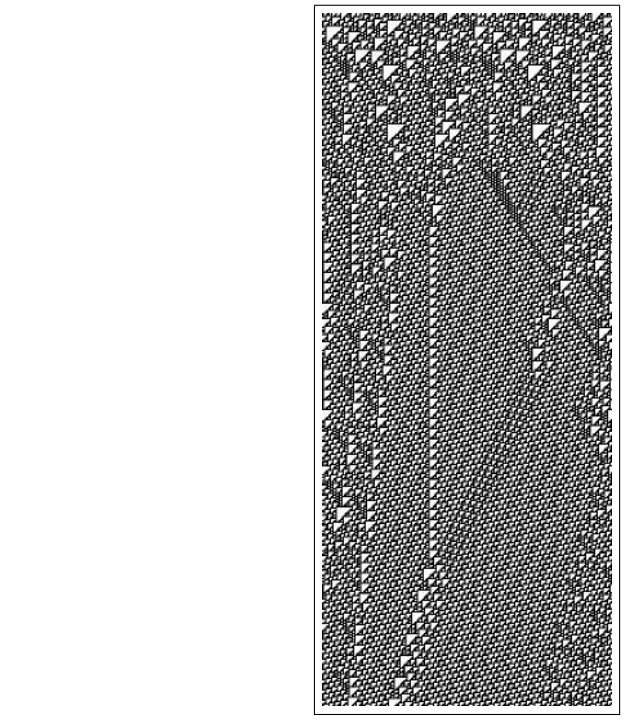
Figure 3.2 demonstrate the evolution of rules from different classes. Rules in the same class display similar behavior with almost all initial states. Wolfram claims that, the behavioral pattern of the Class 3 is convenient to produce pseudorandom sequences, and the Class 4 more likely to include the rules with capability of universal computation. Indeed, Rule 110 is known to be one of the simplest example of the universal computers.

Wolfram's classification has come in for criticism about its practicality since it does not based on any measurement or valuation metric. So indeed, Wolfram had arranged this categorization by examining each rule's behavior visually and individually. That examination includes an entropy comparison of the rules as well, but there is no certain boundaries between the classes. As a result, some of the rules displaying unusual behavior do not precisely fall into any of the classes. Despite its arguable aspects, Wolfram's classification provides a useful approach to the subject and set ground for further and better inspection of the rules, as can be deduced from the volume of studies following Wolfram's work.

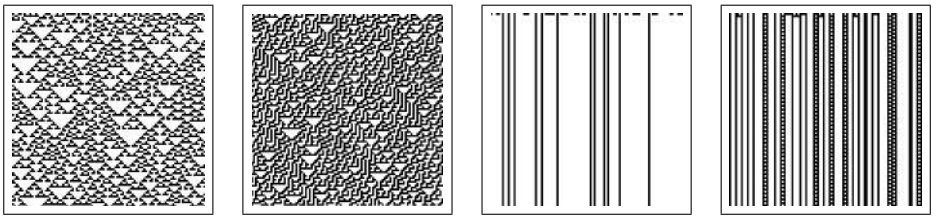
Going beyond the elementary rules, Wolfram asserts that every process in nature should be regarded as a "computation" under the guidance of simple rules and the capability of universal computation draws an upper bound to the complexity of the computation in nature. Therefore he believes that CA provides a very appropriate platform to construe the phenomena in terms of computation in various branches of science from quantum physics to biology and economics. Moreover, he has an idea that ultimately the whole universe can be explained by a simple and short rule in the view of his computation-based interpretation.

3.2 Cellular Automata and the Definitive Parameters

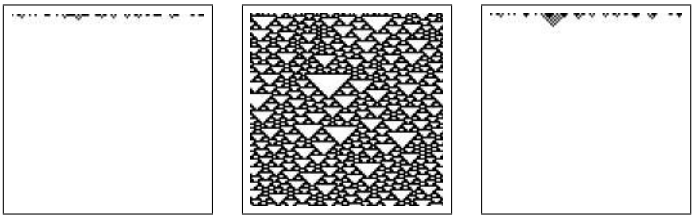
In the simplest form, a CA is an autonomous system with a regular (possibly infinite) lattice of cells, which is spatially discrete, and updated its content in discrete time steps according to a specific rule. Each cell contains a value within a predetermined set of k values, so that possible combinations of state values determine *the configuration* (or *state*) at a certain time. The configuration is updated at each time step. A cell's content at any time is determined by the previous contents of neighbor cells and optionally its own content. This scenario portrays a local interaction between the cells in the same



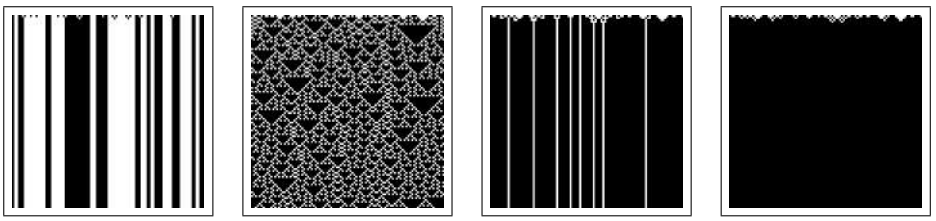
(a) Rule 110



(b) Rule 22 (c) Rule 30 (d) Rule 4 (e) Rule 108

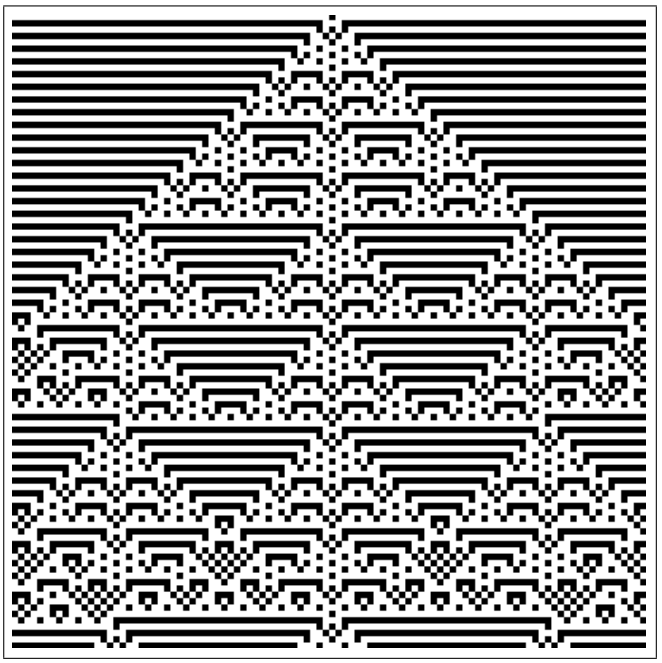


(f) Rule 32 (g) Rule 126 (h) Rule 160

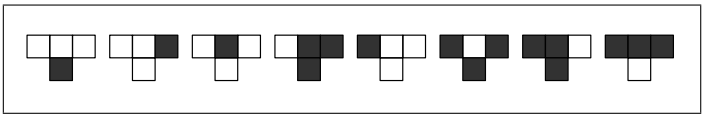


(i) Rule 232 (j) Rule 182 (k) Rule 218 (l) Rule 250

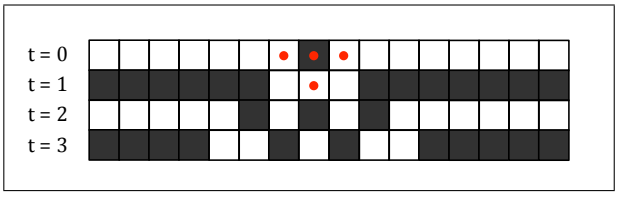
FIGURE 3.2: Time-space diagrams of rules from different classes are shown. (a) is extended to present more iterations to demonstrate the behavior of Class 4 rules.



(a) Rule 105



(b) Local Function: Rule 105



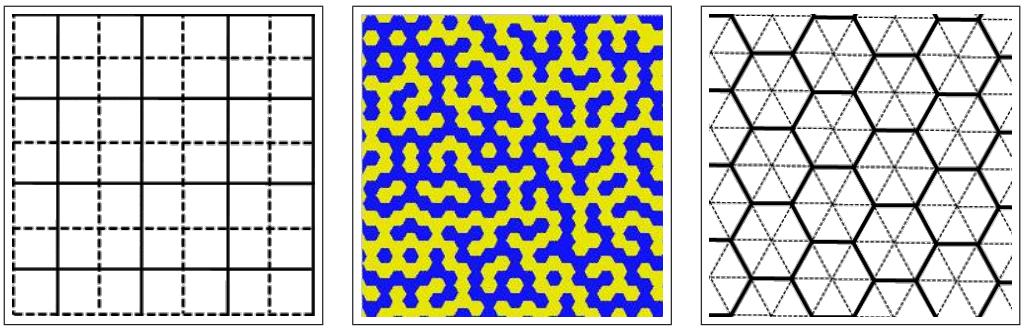
(c) Application of the rule

FIGURE 3.3: (a) shows evolution of Rule 105, of which truth table is given in (b). Figure (c) shows how the rule is applied on a cell. Three cells marked with red dots in the first row are inputs for the marked cell in the second row.

neighborhood. Proceeding in this way, the whole line of cells are modified at each time step as each cell synchronously passes through this local interaction.

Figure 3.3(a) shows an example of binary, one dimensional, 3-neighborhood, finite CA with a state length of 100 bits. There are four parameters that characterize a CA construction. These are lattice structure, cell contents, neighborhood scheme and the guiding rule. To scrutinize more, the number of variables characterizing a CA could be increased however, the independency between the variables may be lost in that case. For example, defining the lattice size also tells whether the CA is finite or infinite, so there is no need to assign a new variable to indicate finiteness.

In Figure 3.3, the lattice is chosen to be a finite line of 100 cells. The cell content is either



(a) Orthogonal Grid (b) Hexagonal Grid (c) Triangular Grid

FIGURE 3.4: Various lattice styles used in CA models

zero (marked with a white cell) or one (marked with a black cell). As seen in 3.3(c), each row of cells represents the state at the respective time step and time proceeds downward. At time $t = 0$ the state is called *the initial state* and given to the system as an input. In this example, the initial state is all zero with a single one in the middle.

Figure 3.3(b) displays the guiding rule and Figure 3.3(c) demonstrates a restricted closer look to the first few steps in 3.3(a). Each cell in a state is modified according to the guiding rule and the newly generated state is written under the first one to obtain a visualization of the evolution in time.

In order to determine the value of the cell which is marked with a red dot at time $t = 1$, the three cells with red dot at time $t = 0$ are considered. The three-cell-combination is mapped to a white cell in the figure 3.3(b), so the cell with the red dot at time $t = 1$ will be white. All cells are modified in the same way. This operation can be run simultaneously for each cell on a state at time t, because the values at time t only depends on the values at time $t - 1$. So that the whole evolution can be computed quickly. This brings the advantage of "parallelism" in CA.

3.2.1 Lattice Geometry

Figure 3.4 shows some possible grid structures used for two-dimensional CA[3]. Most studied one is the orthogonal grid. In any lattice form shown above, both finite and infinite styles are available. As a definitive variant of a CA, the lattice geometry is intended also to give information about the dimension of the CA and the size of the configurations.

CA with infinite lattice is also termed as *tesellation automata* in math literature. Infinite models are favorable to work on theoretical problems. In math and information theory

[3]Figures are taken from [36] and [37].

articles, it is also common to assume an infinite lattice for both finite and infinite cases. In such a case, finiteness is implied by having a finite number of non-quiescent (see Section 3.2.2) cells on the lattice at any time t. But this approach may be problematic in application-oriented texts.

Also one may encounter some studies in literature which designs lattice structures as graph. In that case cells become nodes of a graph and the edges determines the adjacent cells.

3.2.2 Cell Content

Each cell on the lattice takes a value from a finite set of states at each discrete time step. State values are generally chosen integers mod n, $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$. One of the states is named as *quiescent state*, it is generally zero. Sometimes, it is required that the local rule of the CA assigns the configuration with all quiescent cells to the quiescent state (assign all-zero state to zero). This is to ensure that, a finite number of non-quiescent cells in one configuration at time t result in a finite number of non-quiescent cells in the following configuration at time $t+1$ during evolution. This property is generally required for obtaining finite CA characteristic in infinite form.

In the literature, various state specifications can be encountered. Some designs have different state sets for different cells, or they may consider real numbers instead of only integer cell content [34]. There are some mathematical approaches to this issue such as considering state set being taken as a finite field, i.e. $\mathbb{Z}_n$ with a prime n.

3.2.3 Guiding Rule

CA constructions are mainly governed by *the local function* (or *the local rule*) that is applied repeatedly to individual cells. Figure 3.3(b) shows the local function and Figure 3.3(c) shows that each cell is modified under the same local rule to obtain the next generation. As in that example, if every cell is subjected to the same local rule, it is called a *uniform CA*.

In Figure 3.5, it is seen that the aggregate impact of the local function on the configuration at time t produces the configuration at time $t+1$. The function which maps one configuration to the next in time is called *the global function*. Surely, the global function is defined with reference to the local function.

While local rule may be chosen to be the same for all cells and for all time steps, one can choose to apply different rules to different cells at a time according to some control

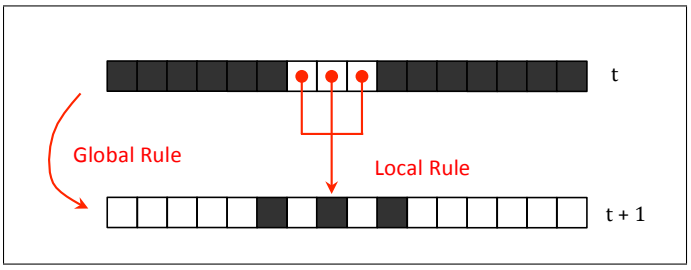


FIGURE 3.5: Global and Local Functions

scheme. Such designs are called *hybrid CA*. Combinations of Rule 90 and Rule 150 were extensively studied in 1990s and it was proven that one of those combinations is a necessary condition to have a maximal-length cycle in a linear hybrid CA design [38]. Yet, another approach is *programmable CA*, in which the local rule changes both spatially and temporally according to a fixed guideline. A dynamic control on the rule selection was also attempted [39].

There are many variations of CA in the literature such as CA with a real-valued local function (*continuous CA*) [34], CA with a probabilistic local function (*stochastic CA*), etc.

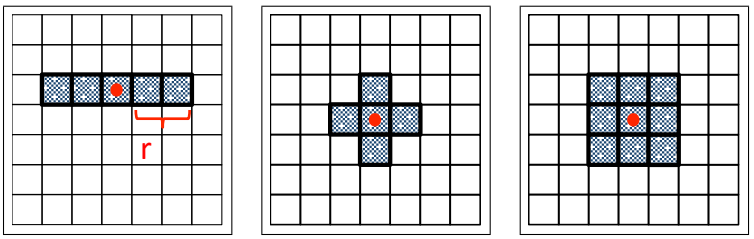
3.2.4 Neighborhood Scheme

A cell's value on the next configuration is determined by some finite number of neighborhood cells and optionally itself. Depending on the lattice structure, several combinations may be possible. On a d-dimensional lattice, neighborhood cells are not only the adjacent ones to the cell to be modified, they can be chosen some distance apart, too.

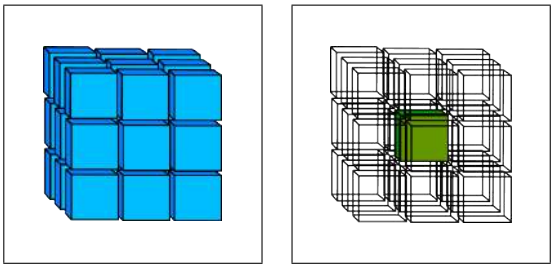
The number of cells in neighborhood scheme determines the number of input to the local function. A commonly studied one is taking the adjacent cells with radius r, where center is the cell to be modified. In that case, next state of the cell to be modified is determined by its neighbor cells which are at most in r distance. In one dimensional space, this approach creates a local function with $(2r + 1)$ inputs as shown in Figure 3.6(a). Figures 3.6(b) and 3.6(c) show the 2-dimensional neighborhood scheme used by von Neumann and Moore. The shaded area marks the input cells, the cell with the red dot indicates the output cell. Figures 3.6(d) and 3.6(e) show the Moore neighborhood in 3-dimensional CA[4].

Boundary Conditions: If a finite lattice is to be used, in addition to the neighborhood scheme, boundary conditions must be specified as well. The cells on the edge of the grid have no neighborhood cells at least on one side. So when modifying the cells at the

[4]Figures 3.6(d) and 3.6(e) are taken from [36].



(a) $r = 2$ (b) von Neumann neighborhood (c) Moore neighborhood



(d) 3D Moore domain (e) 3D Moore range

FIGURE 3.6: Neighborhood illustrations for 1-, 2- and 3- dimensional CA

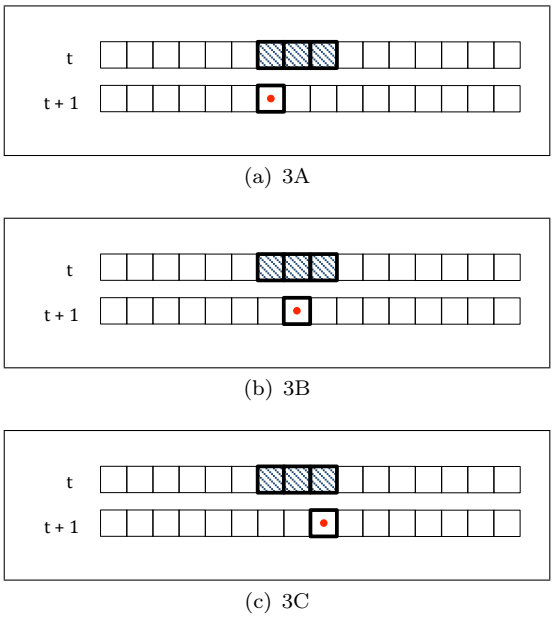
boundary, state of the non-existent neighbors should also be known to determine the next state of a boundary cell. This non-existent neighbor cell might be taken from the opposite end of the grid so that a periodic, *cyclic boundary* is obtained. Another choice is to assign a fixed value to all non-existent neighbors and use it throughout the whole evolution. If this fixed value chosen to be the quiescent state, it is called *null boundary*. Also a predetermined intermediate cell can be chosen as a boundary neighbor. This scheme is then called *intermediate boundary*.

3.3 A Formal Definition of Cellular Automata

Definition 3.1. A Cellular Automaton is a quadruple $(\mathcal{S}, \sigma, \mathcal{N}, \ell)$ such that

1. $\mathcal{S}$ is the set of values a cell can take.
2. σ is the local function.
3. $\mathcal{N}$ is the neighborhood scheme.
4. ℓ is the lattice size.

Remark: The scope of this study is limited to one dimensional CA only. The test results to be presented in Chapter 4 belong to deterministic, binary and uniform CA with finite line of cells. Also, the boundary condition is selected to be cyclic boundary for all CA constructions. Accordingly, the formal definitions related to CA in this chapter



(a) 3A

(b) 3B

(c) 3C

FIGURE 3.7: Naming Convention for Neighborhood Schemes

are made with reference to these specifications. So, a CA construction defined outside of this frame (e.g., a multidimensional, hybrid CA) may not be concretely identified under the following definitions.

Lattice Geometry: The lattice of a one dimensional finite CA can be defined as a finite sequence. If the state length is ℓ, then the sequence will consist of ℓ elements.

$$S^{\ell} = \{s_0, s_1, \ldots, s_{\ell-1}\}$$

Cell Content: On a binary CA, cell content is limited to $\{0, 1\} \subset \mathbb{N}$. So, a state at time t will be defined as:

$$S^{\ell}(t) = \{s_0(t), s_1(t), \ldots, s_{\ell-1}(t)\} \in \{0, 1\}^{\ell}$$

where $s_i(t) \in \{0, 1\}$ for every $i \in \{0, 1, \ldots, \ell - 1\}$ and $\{0, 1\}^{\ell}$ denotes the set of all possible binary sequences of length ℓ.

Neighborhood Scheme:

For a one-dimensional CA with a k-input local function, a k-tuple is needed to denote the neighborhood of a site. Let us denote a k-tuple neighborhood with $\mathcal{N}_k$. There will be only one 0 (zero) in the neighborhood $\mathcal{N}_k$ to mark the position of the cell to be modified. The remaining elements will be integers indicating the distance of that cell to the cell to be modified.

Figure 3.7 demonstrates the abbreviations chosen for different neigborhood types. For a 3-input local function which puts the output at center as in Figure 3.7(b), $\mathcal{N}_3 = (-1, 0, 1) \in \mathbb{Z}^3$. If the output will be written at the leftmost cite as in Figure 3.7(a), then the neighborhood will be $\mathcal{N}_3 = (0, 1, 2) \in \mathbb{Z}^3$. If the output will be written at the rightmost cite as in Figure 3.7(c), then the neighborhood will be $\mathcal{N}_3 = (-2, -1, 0) \in \mathbb{Z}^3$.

Now, the neighborhood for a finite sequence with cyclic boundary can be defined as follows:

Definition 3.2. Let S^ℓ be a finite sequence of length ℓ such that $S^\ell = \{s_0, s_1, \ldots, s_{\ell-1}\} \in \{0,1\}^\ell$. Then *the $\mathcal{N}_k$-neighborhood of the site s_i on a cyclic boundary state* is a k-tuple such that

$$s_{i+\mathcal{N}_k} = (s_{i+n_1 \,(\mathrm{mod}\ell)}, s_{i+n_2 \,(\mathrm{mod}\ell)}, \ldots, s_{i+n_k \,(\mathrm{mod}\ell)}) \in \{0,1\}^k \qquad \forall i \in \{0, 1, \ldots, \ell-1\}$$

Local Function: A local rule can be expressed as a look-up table or via its algebraic formula. If the local function σ is coupled with a neighborhood $\mathcal{N}_k$ then it should be defined from $\{0,1\}^k$ to $\{0,1\}$

Definition 3.3. A *binary, uniform, cyclic boundary Cellular Automaton* is a triple $(\sigma, \mathcal{N}_k, \ell)$ such that

1. σ is the local function: $\sigma : \{0,1\}^k \to \{0,1\}$
2. ℓ is the length of the finite state with cyclic boundary condition.
3. $\mathcal{N}_k$ is the neighborhood scheme: $\mathcal{N}_k \in \{(n_1, \ldots, n_k) |\, n_j \in \mathbb{Z},\ \forall j \text{ and } \exists!\, i \in \mathbb{N} \text{ such that } 1 \leq i \leq k \text{ and } n_i = 0\}$

   Let $S^\ell(t) = \{s_0(t), \ldots, s_{\ell-1}(t)\} \in \{0,1\}^\ell$ denote the configuration at any time t. The evolution occurs as follows:

$$s_i(t+1) = \sigma(s_{i+\mathcal{N}_k}(t)) \qquad \forall i, t.$$

Global Function: The global function χ of a binary, uniform and cyclic boundary CA $(\sigma, \mathcal{N}_k, \ell)$ is defined as $\chi : \{0,1\}^k \to \{0,1\}^k$.

3.4 Elementary Rules

Elementary Cellular Automaton is a class of one-dimensional binary CA which have the neighborhood scheme as (-1,0,1). That is, each cell is updated according to the values

TABLE 3.1: Truth table of Rule 30.

abc	111	110	101	100	011	010	001	000
$\sigma(abc)$	0	0	0	1	1	1	1	0

TABLE 3.2: Rules 75, 89, 45 and 101 form a family.

	111	110	101	100	011	010	001	000
Rule 75	0	1	0	0	1	0	1	1
Rule 89	0	1	0	1	1	0	0	1
Rule 45	0	0	1	0	1	1	0	1
Rule 101	0	1	1	0	0	1	0	1

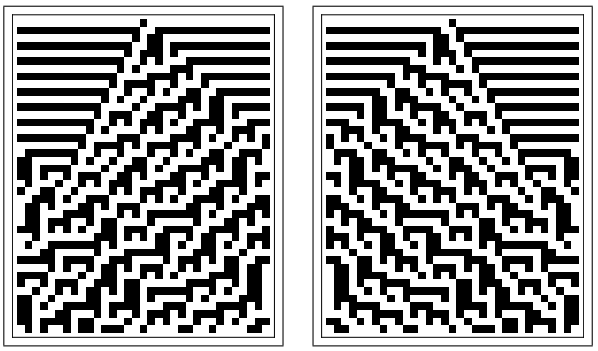
of two adjacent cells and itself. Considering the local functions, there are 2^{2^3} possible deterministic rules which take 3-bit input and output 1 bit. In order to uniquely name this set of 256 local functions, such a way was followed in the literature, put the inputs in binary descending order 111, 110, 101, ... Then read the outputs as a binary number with the most significant bit coming from $\sigma(111)$. The corresponding decimal number is given to that function as a name. Rule number uniquely describes the function's truth table.

3.5 Rule Families

Figure 3.8 shows four CA evolution plotted down with different local rules. Figure 3.8(b) is the mirror image or *the reflection* of Figure 3.8(a) and figure 3.8(c) is complement of figure 3.8(a) in the sense that ones and zeros are swapped in the configuration. So Rule 45 is called *the conjugate* of Rule 75. Figure 3.8(d) is both reflection and conjugate of figure 3.8(a). These four CA produce the same behavior pattern given the identical initial state.

For any given CA rule, its reflection can be found by interchanging left and right neighbors in the local function. Likewise, its conjugate can be found by interchanging ones and zeros in the truth table. Its reflection-conjugate is obtained applying both procedures. Therefore, in a binary one-dimensional k-neighbor CA space, it is possible to group the rules so as to create rule families by collecting the ones that have the same behavior. The family of a rule consists of the rule itself, its reflection, its conjugate and its reflection-conjugate. As these four can be distinct rules, they might all be the same or pairwise same, as well.

Reflection– The rule spaces with central neighborhoods, such as 3B or 5C, will have all family members in the same space. But, a rule in 3A has its reflection in 3C. Likewise, reflections of 4A will be in 4D and 4B will be in 4C.



(a) Rule 75 (b) Rule 89

(c) Rule 45 (d) Rule 101

FIGURE 3.8: Space-time diagrams of a rule family is presented.

Let f be the function which maps a rule in 3A to its reflection in 3C. Since reflection of a rule is unique, f is one-to-one. Also, the domain and the range of f have the same number of functions, therefore f is onto hence, is a bijection. The rule pairs that are matched to each other by f will be in the same family, because they produce the same patterns.

So, in case of a behavioral analysis over the set of all 3-input rules (3A, 3B, 3C), examining one of the set from 3A and 3C is enough to reach a general conclusion, since they exhibit the same behaviour.

Conjugate– Let us denote the conjugate of a local rule σ as σ^{c}. Given the same initial state, σ and σ^{c} produces the same output ones and zeros swapped. Hence the relation between the two is $\sigma^{c}(\overline{a}) = \overline{\sigma(a)}$ where a is a binary input.[5]

The conjugate of a rule is unique and has the same neighborhood scheme with the original rule. Therefore, the conjugate of a rule is either itself or another rule in the same neighborhood set.

[5]The bar over a binary variable denotes its conjugate such that $\overline{0} = 1$ and $\overline{1} = 0$.

3.6 Producing Randomness via Cellular Automata

3.6.1 CA-Based PRNGs

CA was first proposed to generate randomness by Wolfram in a 1986-dated paper [9]. Wolfram especially pointed out Rule 30 among the elementary rules, emphasizing its capability to produce complicated behavior even when it starts its evolution from quite non-random initial states, like all zero but one bit set at the center of the state. He preferred a plain model to express the idea of producing randomness via very simple deterministic rules rather than elaborating on details of a concrete PRNG design. For that design, a uniform, cyclic boundary CA is used as the updating function, and the selection function always chooses the central bit of the internal state. The period of the generated sequence is dependent on the state size and the seed. Its maximal period is nearly $2^{(\ell+1)/2}$ where ℓ is the state length. This PRNG passed a group of 7 statistical tests (Knuth's tests) and was found to have a better quality than LFSR and LCG of the same length but no better than the binary expansions of $\pi, \sqrt{2}$ and e.

In some math articles, different selection functions are suggested to decrease periodicity and linearity of the sequences. Sipper tried taking one bit of every 3 bits on the temporal sequence and reported statistical advancement in [16].

In [40], Wolfram recommends using the PRNG based on Rule 30 to produce key stream to be directly XORed with the plaintext in a stream cipher procedure. However, Meier and Staffelbach [41] showed that the temporal sequence could be found out via a known-plaintext attack and due to the property known as "the left toggle" of Rule 30, guessing only the half of the seed reveals the whole history of the PRNG.

Moreover, [42] describes an inversion algorithm to recover the previous internal state given the current one in Wolfram's uniform Rule 30 design. Inversion is not only intended to work on Rule 30 but other one dimensional uniform CA models. It is performed through finding the best affine approximation of the local rule. Inverting one step back an internal state of length ℓ takes $\mathcal{O}(\ell)$ time and $2^{\ell/2}$ time at the worst case. In [43, 44], it is shown by using Walsh transformation that elementary rules are not correlation-immune, therefore, a uniform CA design created with an elementary rule is not appropriate for cryptographic purpose.

A good amount of articles comparing LFSR and CA-based PRNGs are available in the literature, mainly because LFSR is also known for its simplicity and providing good statistical quality in generating randomness. Hortensius showed that both a very popular hybrid model of Rule 90/150 and Rule 30-based designs have better statistical quality than the basic LFSR model of the same length [45].

As expected, cyclic boundary is found to be better than fixed boundary [46] and two dimensional CA better than one dimensional [14] in statistical quality.

Serra studied constructing a linear CA corresponding to a given LFSR [38]. He showed that there exist a hybrid CA of Rule 90 and Rule 150 for any LFSR proving that they are governed by the same primitive polynomial. Also, an algorithm was provided to determine the control scheme of a hybrid 90/150 CA for a given LFSR. The isomorphism between the spaces of LFSR and linear hybrid CA was proven assuming null boundary on CA constructions. Later it was found out that the characteristic polynomial of a linear CA with cyclic boundary is always factorizable, therefore can never be primitive. That is, the maximum period cannot be achieved via a cyclic boundary linear CA.

Nandi and Chaudhuri examined the connection between algebraic structures and programmable CA (PCA) models created by using linear and affine functions. In [47], they define the state transitions on a null-boundary PCA as even permutations, consequently obtain an alternating group. Based on that group, they proposed symmetric block and stream cipher schemes. However, later it was shown in [48, 49] that what they created is not an alternating group and the proposed encryption scheme is vulnerable to ciphertext-only attack.

In the realm of hybrid CA, the idea of using a genetic algorithm with a metric of entropy to search for the efficient rules initiated a series of works [50–52]. These studies mostly center around the use of rule 90, 105, 150 and 165 among the linear elementary rules. An evaluation over four statistical tests reports that they perform better than the uniform Rule 30 and a hybrid Rule 90/150 designs [51].

Shin et. al. [46] classify the elementary rules according to combinations of the logical operators included in the shortest algebraic formula of a rule. They performed statistical tests on each class and find out that the best statistical results are obtained from the rules including XOR in their formula.

As an example of physical TRNG, Tkacik presents a construction which was also implemented and used for a while within Motorola Company [53]. That PTRNG uses two ring oscillators to clock a 43-bit LFSR and a 37-bit hybrid 90/150 CA. The outcomes are then permuted and combined via an XOR operation to produce a pseudorandom string, which possibly goes through a post-processing. The test results show that while the individual outcomes of LFSR and CA mechanisms do not present a satisfactory statistical quality, their combination behaves pretty close to the theoretical reference.

TABLE 3.3: Rule 184: A balanced rule from 3-input CA set. There are $\binom{8}{4} = 70$ balanced rule for each neighborhood space of 3A, 3B and 3C.

abc	111	110	101	100	011	010	001	000
$\sigma(abc)$	0	1	0	1	1	1	0	0

3.6.2 Balancedness

One of the defining characteristic of an ideal random sequence is uniformity: every possible value should appear in the sequence equally likely. That means, in a PRNG mechanism, the outcome of the whole procedure (the combination of the updating and selection functions) must yield approximately the same amount of 1s and 0s. Considering Wolfram's PRNG model suggested in [9], as well many other CA-based constructions, the output is formed by directly taking the $\lfloor \ell/2 \rfloor^{\text{st}}$ bit of each state. In order to bring an equilibrium to the amounts of 1s and 0s in the internal state, the local function(s) used should be balanced. Otherwise one of the values (0 and 1) will outnumber the other in the internal state and similarly in the temporal sequence.

Definition 3.4. A Boolean function $f : \{0,1\}^k \to \{0,1\}$ is said to be balanced if it produces as many 1s as 0s over its input set.

3.6.3 Mutual Information

For producing pseudorandom sequences, balancedness is a basic requirement of local functions to provide uniformity. So, if we are to choose good candidates from a pool of rules, it is best to eliminate the unbalanced ones first. At one step further, an elimination –just by looking at the local rules– can be made according to the requirement that whenever a zero appears in the random sequence, the probability of having 0 or 1 should be the same in the next bit. Similarly, for one. In that case, the joint distributions of 0 and 1 must be consulted over the local function. The mutual information is the very tool needed.

Mutual information of two random variables X and Y indicates how much knowing one of the variables' state gives information about the state of the other. If X and Y are independent, then their mutual information $\mathcal{I}(X;Y)$ will be zero.

Mutual information of X and Y can be calculated as follows:

$$\mathcal{I}(X;Y) = \sum_{x,y} P_{XY}(x,y)\ \log_2 \left(\frac{P_{XY}(x,y)}{P_X(x)\, P_Y(y)} \right),$$

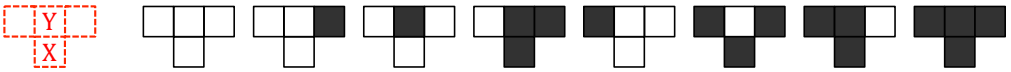


FIGURE 3.9: Random variables X and Y and Rule 232

where $P_X(x)$ and $P_Y(y)$ denote the marginal probability distributions and $P_{XY}(x, y)$ denotes the conditional probability of $X = x$ given $Y = y$.

If the variables X and Y are specified as shown in the figure 3.9. Both variables can take one of the two states: $X, Y \in \{0, 1\}$. $\mathcal{I}(X;Y)$ will give the correlation between two adjacent cell on a vertical line on the space-time plot.

Let us calculate the mutual information of 3B Rule 232 shown in the figure 3.9. Since Rule 232 is balanced, $P_X(x) = 1/2 \ \ \forall x \in \{0, 1\}$. And, the function is defined for all 3-bit inputs, hence $P_Y(y) = 1/2, \ \ \forall y \in \{0, 1\}$ When $Y = 0$, X becomes zero 3 times over the all input set, namely at 000, 001 and 100: $\sigma(000) = 0, \ \ \sigma(001) = 0$ and $\sigma(100) = 0$ Therefore $P_{XY}(0, 0) = 3/4 \cdot 1/2 = 3/8$. Similarly, $P_{XY}(0, 1) = 1/8$, $P_{XY}(1, 0) = 1/8$, $P_{XY}(1, 1) = 3/8$

$$\begin{aligned}
\mathcal{I}(X;Y) &= \sum_{x\in\{0,1\}} \sum_{y\in\{0,1\}} P_{XY}(x, y) \ \log_2 \left(\frac{P_{XY}(x, y)}{P_X(x) \, P_Y(y)} \right) \\
&= 2 \cdot \left(3/8 \ \log_2 \left(\frac{3/8}{1/2 \ \cdot \ 1/2} \right) \right) + 2 \cdot \left(1/8 \ \log_2 \left(\frac{1/8}{1/2 \ \cdot \ 1/2} \right) \right) \\
&= 0.188722
\end{aligned}$$

A balanced local rule with $\mathcal{I}(X, Y) = 0$ directly implies that $P_{XY}(x, y) = 0.25 \ \ \forall x, y \in \{0, 1\}$ for that rule. There are $\binom{4}{2} \cdot \binom{4}{2} = 36$ rules with that property in the 3-input CA space for each neighborhood of 3A, 3B and 3C.

3.6.4 Entropy

In the literature, especially where CA is examined in the context of physics and complex systems, entropy keeps a crucial place for the analysis of CA rules. Various versions of entropy are available especially in the information theory and physics-oriented articles. Here, a definition of entropy which deals with probabilities is chosen.

Definition 3.5. n-bit entropy S_n of a binary string M is calculated as

$$S_n = - \sum_{i=0}^{2^n - 1} p_i \log_2 p_i$$

where i is an n-bit string and p_i is the observed probability of its occurence as a substring in M.

This definition of entropy is used in [35] by Wolfram for measuring entropies of CA-based PRNG outputs. It is also commonly used in CA studies since generally Wolfram's work is taken as reference. In [10], there are references to the studies which measures the entropy similarly but on the internal states instead of the output of PRNG.

Chapter 4

Test Results

This chapter presents the statistical test results collected on 5-input CA. The results also include information about 3 and 4-input rules. Our main motive is to detect the rules with good statistical quality over the space of 5-input CA to see how many of them have potential for random number generation. In the 5-input space there are 2^{2^5} functions for each of the 5 different neighborhood schemes. This implies $5 \cdot 2^{2^5} \approx 2 \cdot 10^{10}$ functions. That is a huge space for an exhaustive search.

Looking at the research done so far, hybrid systems are found useful and better in performance also more challenging against predictability attacks. In the elementary rule space, the rules that gain the most attention are Rule 30 and the linear rules for they allow an algebraic analysis over the models they created. Beside the elementary rules, high entropy is the reason for preference.

Linear CA rules within a space can be specified directly without a need for an exhaustive search over the space. However, the other rules requires individual exploration because there is no quick way to find out the ones with good random behavior. Still, some metrics – like entropy and mutual information or others – are known to be proportional with unpredictable behavior to some extent [54]. Though these metrics do not directly point out the good ones, they help to narrow down the set of candidates for an exhaustive search.

3 and 4-input CA sets have reasonable sizes for an exhaustive search. 3-input CA has 256 functions with 3 different neighborhood scheme (3A, 3B, 3C) which makes a total of $256 \cdot 3 = 768$ distinct rules. In the set of 4-input CA, there are 2^{2^4} rules for each neighborhood type, which sum up to 2^{18}. But 5-input CA space is very large with $2^{2^5} \cdot 5 = 2^{32} \cdot 5$ rules. But most of these rules produces sequences that cannot fulfill even

TABLE 4.1: The tests in NIST Test Suite which return multiple p-values.

Test Name	**Number of p-values computed**
Cumulative Sum	2
Random Excursions	8
Rand. Exc. Var.	18
Serial	2
Non-overlapping T. M.	from 2 to 284 depending on block size

the basic qualities of a pseudorandom sequence. Hence, an elimination is necessary to discard the poor-quality rules before testing.

In what follows, first details of the elimination process will be presented. Then sequences are generated by the PRNG that Wolfram suggested in [9]. Then NIST Statistical Test Suite is applied to all sequences respectively.

4.1 Output of a Statistical Test

The NIST Test Suite has 15 statistical tests. The sequence to be tested will be subjected to all of them in return. Resultantly a p-value is computed for each individual test to indicate the result. NIST Test Suite creates two files as the output of the whole test. One of them presents p-values as well as computational information about the intermediate steps of the tests, the other one only lists the p-values for ease of parsing. P-value above the significance level implies success on that test. Some of the tests return more than one p-value since they are constructed to check more than one target as shown in Table 4.1.

To pass a test which returns multiple p-values, all p-values must be above the significance level. This is a quite stringent condition to satisfy if the block size is large. For example, when the block size is taken to be 10 bits in Non-overlapping Template Matching Test, all 284 p-values must indicate success to pass the test. If only one or two of them are failure, then the PRNG may be tested with various other seeds, so that it will be clear if the failure on that condition is a characteristic of the PRNG or not.

4.2 Testing Strategy

The appropriate approach when testing a particular PRNG is to apply the test suite on various samples created via different seeds or taken from different portions of long sequences. While it is well accepted that the tests must run more than once, it is not clear how many trials should be done. NIST documentation [6] suggests determining the

number of trials according to the selected significance level. If the significance level α is chosen to be 0.001, that means that we assumed that a random sequence may fail at most once in 1000 trials. So, ideally the test should run 1000 times. If the test practitioner makes only 100 tests, there might be no failure even when the tested PRNG was not that good.

However, the approach taken in this study is not making a detailed analysis of a particular PRNG but scanning the set of 5-input CA rules to make an approximate evaluation over the set. 248,474,664 rules from the set of 5-input CA were tested for that purpose. It was very difficult to test each rule several times considering the required computational power and time for testing. Therefore the test suite run only once for the tested rules.

4.3 Interpretation of the Test Results

As stated in the test documentation, the outcome of the whole testing process may be one of the three:

The tested RNG is considered successful if no anomaly is detected that lead a deviation from randomness assumptions. The result is a failure if strong indication of non-random patterns are detected. The test is inconclusive when the test results do not show any clear sign of deviation from randomess. In that case, extra tests should be performed to reach a conclusion.

There is no strict assessment scheme defined on the outputs but the documentation of the NIST test suite suggests two strategies (given in the sections 4.3.1 and 4.3.2) for interpretation of the results. A PRNG is considered successfully passed the test if both methods return positive results. If just one of them is satisfied, it will be the inconclusive case. Two failures indicate an obvious defect. These two evaluation methods are explained in the following.

4.3.1 Rate of success over all trials

If the practitioner made 1000 trials for a particular PRNG, then for each test, there will be a collection of 1000 p-values. Then a confidence interval is determined based on the number of trials $m := 1000$ and the significance level α as $(1-\alpha) \pm 3\sqrt{\frac{\alpha\,(1-\alpha)}{m}}$. The pass rate should be greater than the lower bound for a test to be satisfied, that is:

$$\frac{\#\{\text{P-values greater than or equal to } \alpha\}}{m} > \left((1-\alpha) - 3\sqrt{\frac{\alpha\,(1-\alpha)}{m}}\right)$$

For each test, the above condition is checked and a conclusion is reached as a failure or success.

4.3.2 Distribution of P-values

P-values range between zero and one. For a good PRNG, p-values of different samples on the same test are expected to be uniformly distributed on the interval [0,1) [7]. Provided that the number of test trials is more than 55, NIST recommends a goodness of fit test to see how close the p-value distribution to the uniform distribution. In that case, the following calculation is performed.

Let us assume we made m trials, that is there are m p-values. Divide the interval of $[0,1) \in \mathbb{R}$ into 10 evenly spaced subintervals such that $[0.0, 0.1), [0.1, 0.2), \ldots, [0.9, 1.0)$. Then count the number of p-values that fall within each interval. Let F_i be the number of p-values that fall in the i^{th} interval where $i \in \{1, \ldots, 10\}$.

Over the p-value distribution, χ^2 test will be applied and a new p-value will be calculated as $\mathrm{P}_\tau = Q(9/2, \chi^2/2)$ where

$$\chi^2 = \sum_{i=1}^{10} \frac{(F_i - m/10)^2}{m/10}$$

and $Q(\mathrm{a}, x)$ is the incomplete gamma function which is defined as

$$Q(\mathrm{a}, x) = \frac{\Gamma(\mathrm{a}, x)}{\Gamma(x)} = \frac{\int_x^\infty \mathrm{e}^{-t} t^{\mathrm{a}-1}\, \mathrm{d}t}{\int_0^\infty \mathrm{e}^{-t} t^{\mathrm{a}-1}\, \mathrm{d}t}$$

with $Q(\mathrm{a}, 0) = 1$ and $Q(\mathrm{a}, \infty) = 0$. $\mathrm{P}_\tau \geq 0.0001$ implies uniformly distributed p-values.

4.4 Testing over a big space of functions

If a rule with good statistical quality is to be chosen from a set, exhaustive search is the most costly method to try. A relatively practical way could be computing entropies of the generated sequences. Within a group of pseudorandom sequences, the ones with the highest entropy are expected to score best on the statistical tests.

Also, entropy can be computed on a reasonably short sequence and generally it requires much less operation on the sequence compared to a bunch of statistical tests. However, if entropy is used to screening randomness capabilities of the rules beforehand, one still needs to produce the sequences to reach a conclusion on the rules generating them.

A mathematical tool, which is similar to entropy in application, is the Walsh spectrum. Similarity arises in that both Walsh spectrum and entropy are computed on the generated sequence. In [55], Yuen stated that the Walsh spectrum of a sequence indicates its uncorrelatedness and a good random sequence should have a flatter spectrum (spectrum entries should have small absolute values). Martin used this method to select good rules among the elementary rules [43]. During the selection process, Martin generated several short sequences for each single rule to reach a decision about their spectra.

Is it possible to make any inference about the statistical quality of a sequence just by looking at the local rule instead of producing sample sequences for measurements? Though we can make some simple guesses by looking at the rule, this is still a hard question. It is closely related to the classification problem on CA at one side, and to measurability of randomness at the other side.

Langton [54] defined a parameter on CA rules, which is directly computed on the rule table rather than the generated sequence. That parameter is calculated for each rule separately, then it is compared to a critical value that depends on the number of inputs given to the function and the number of values that a cell can attend. Accordingly, an approximate conclusion can be reached about the behavior of the function (such as approximately after how many iteration the function goes through a phase transition or enters into a converging trend). Langton's parameter gives sensitive results on larger input and cell values like 5 or above for both [54].

4.5 Our Procedure

In the set of all Boolean functions defined from $\{0,1\}^5$ to $\{0,1\}$, there are 2^{2^5} functions. If we consider all 5-input CA rules with adjacent neighborhood, 5 different neighborhood schemes are possible: 5A, 5B, 5C, 5D, 5E. Rules of 5A writes the output at the place of the leftmost bit taken as input, 5C writes the output at center and so on.

Over the space of 5-input CA rules, we made an elimination as described below to apply NIST Statistical Tests on them:

1. Discard the two of the groups for they include reflection of others: Continue with 5A, 5B and 5C. This makes $3 \cdot 2^{32}$ rules to test. As mentioned in the previous chapter, the rules in 5A - 5E and 5B - 5D produce the same patterns since one group includes the reflections of the other group in its pair.

2. Remove the the unbalanced rules for sake of uniformity: The number of functions decreases to $3\binom{32}{16} \approx 3 \cdot 2^{29}$.

3. As an elimination criterion, we selected mutual information. It is applied in the way as described in Section 3.6.3. Choose the functions with zero mutual information: This reduces the number of functions to $3\binom{16}{8}^2 \approx 3 \cdot 2^{27}$.

4. Group the functions with their conjugates and discard one of them. (The one with smaller rule number is kept.) There remained 82,824,885 rules in 5A, 82,824,889 rules in 5B and 82,824,890 rules in 5C.

The test is conducted on those 248 millions of functions. For generating random sequences, we have used Wolfram's model in [9]. That is, the internal states are generated via uniform CA with cyclic boundary and the selection functions chooses the central bit from each state. Every rule is tested once with the same initial state, which is all zero except the central bit.

4.6 Results and Observations

In the following, various figures and tables are presented based on the test results and other measurements like entropy and mutual information. A short explanation is necessary to clarify calculation details.

Entropy: Entropy is calculated as explained in Section 3.6.4. All entropy measurements are performed on 8-bit basis. The sequences are generated by Wolfram's model in [9] with cyclic boundary. The width of the internal state is 64 bits. The first 100 bits of the output were discarded and the next 2056 bits were used for calculation.

Mutual Information: Mutual information is calculated as described in Section 3.6.3. There is no parameter for mutual information calculation beside neighborhood scheme and number of inputs.

NIST Test Parameters: The tested sequences are generated by Wolfram's model in [9] with cyclic boundary. The statistical tests are conducted using different parameters for different graphs. The parameter set will be specified as an 8-tuple $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}, \boldsymbol{f}, \boldsymbol{g}, \boldsymbol{h})$, where

$\boldsymbol{a}$ denotes the width of internal state of CA configuration,

$\boldsymbol{b}$ denotes the length of the generated sequence in bits,

$\boldsymbol{c}$ denotes the block size in the Block Frequency Test,

$\boldsymbol{d}$ denotes the window size in the Non-overlapping Template Matching Test,

e denotes the window size in the Overlapping Template Matching Test,

f denotes the block size in the Linear Complexity Test,

g denotes the block size in the Serial Test and

h denotes the block size in the Approximate Entropy Test.

In all of the tests, the significance level is taken as 0.01.

4.6.1 Change in State Width

In this study, all CA constructions are uniform and runs with cyclic boundary. The selection function is fixed to choose the central bit of internal state at each time t. With these conditions, two of the qualities that influence the statistical testing results are the width of internal state and the length of the sequence.

There are 88 rule families in the elementary (3B) rules. Two of them, {30, 86, 135, 149} and {45, 75, 89, 101}, are remarkably well in producing randomness. In the following, test results of Rule 30 and Rule 45 are included as a representation of overall behaviour of these two groups.

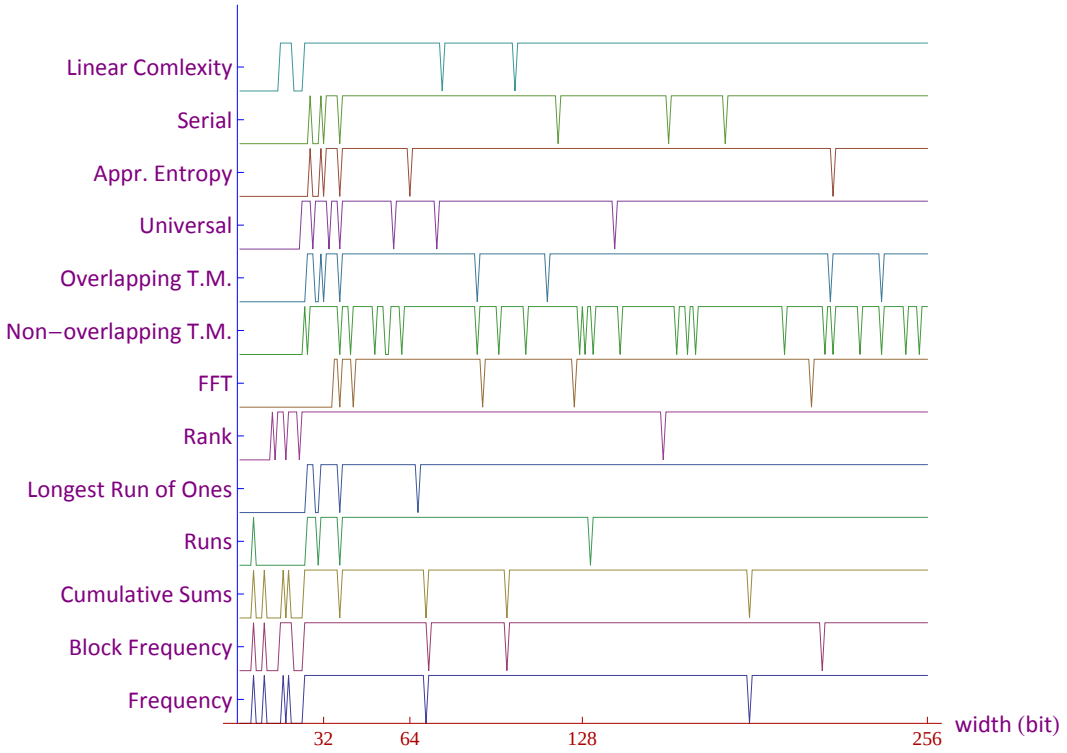
Testing is performed on the outcomes of the Rule 30- and Rule 45-based PRNGs with varying state width and sequence length. Figures are provided to present the effect of these changes on statistical quality of produced sequences. The internal state is taken as all zero with an exception on the central bit. Test parameters are set to (–, –, 10000, 5, 9, 10, 16, 1000). Results of two tests (Random Excursions and Random Excursions Variant) are omitted because they are not applicable on 500,000-bit-long sequences.

Figures 4.1 and 4.2 imply that an internal state with width less than 64-bit is not satisfactory even for Rule 30 and 45. Two more graphs for each rule showing the results for 1 millions and 5 millions of bits are added to Appendix A.

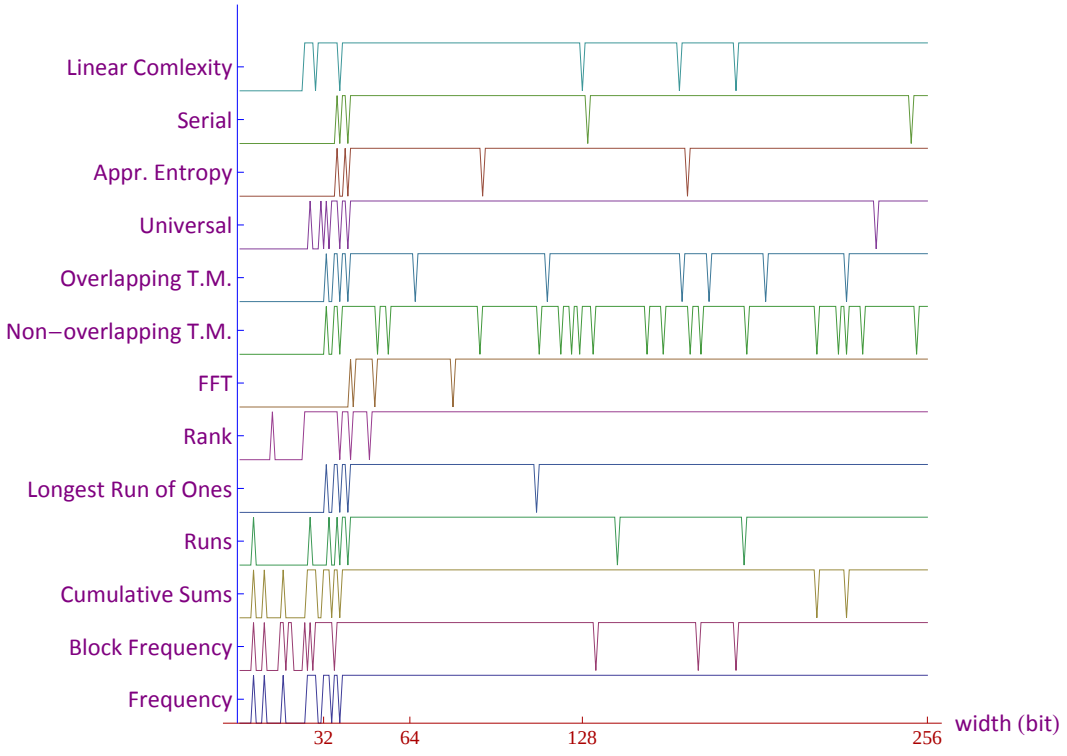
4.6.2 Change in Neighborhood Scheme

The effect of change in neighborhood scheme might be considered in the context of selection function of PRNGs. Entropies are computed on the output strings generated with a cyclic boundary, 100-bit wide internal states. These conditions imply a space-time diagram covered on a cylindrical surface which has a perimeter of 100 bits.

If 3B neighborhood is taken as reference, 3A shifts each new state left by one bit and 3C shifts right by one bit. In this study the selection function is set to take a column

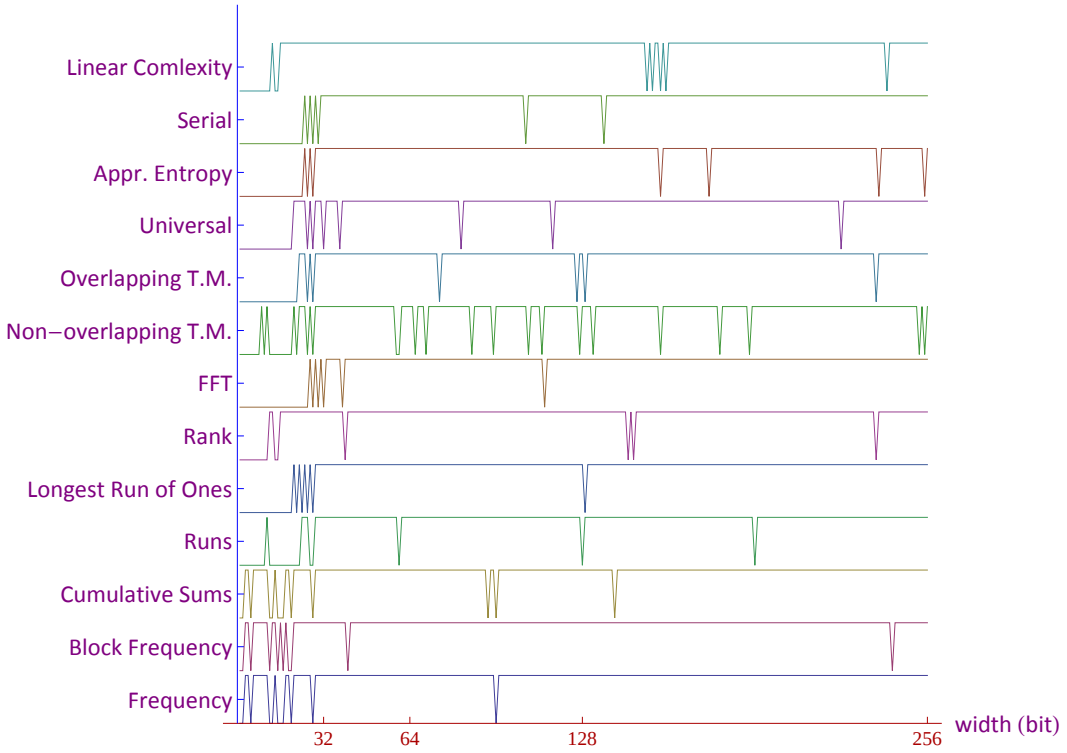


(a) Rule 30, length: 500,000 bits

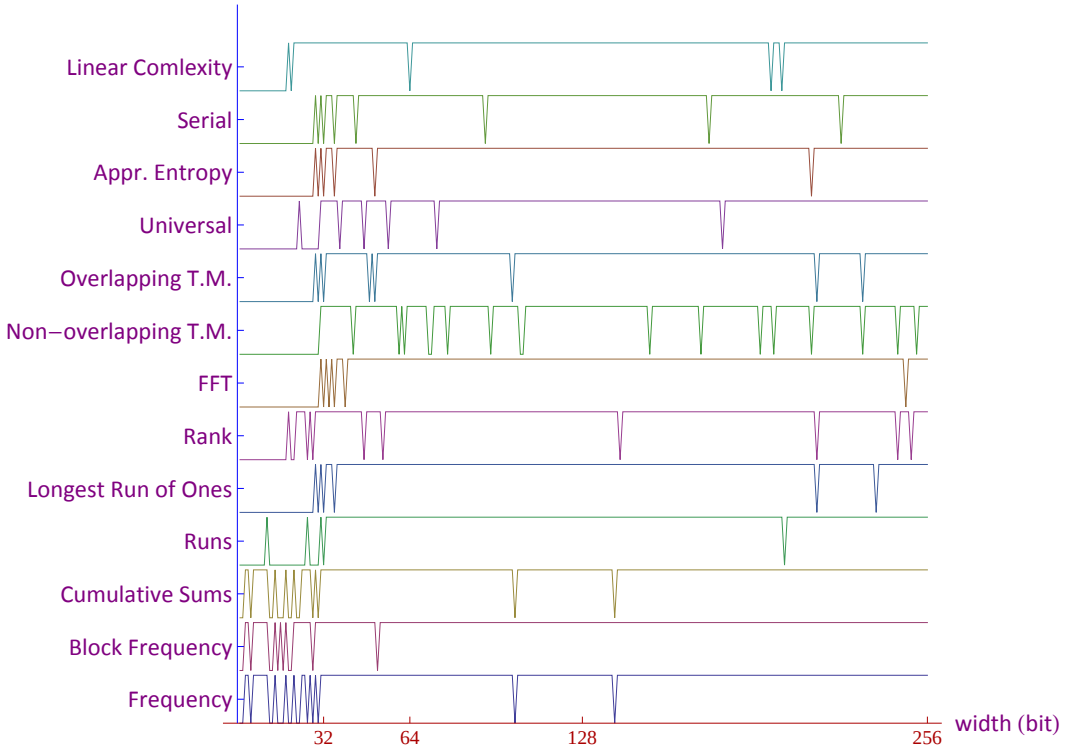


(b) Rule 30, length: 10,000,000 bits

FIGURE 4.1: The horizontal axis shows increasing width of the internal state. The only difference between Figure (a) and (b) is the length of the produced sequence. Test scores are coded as 1 and 0, where 1 implies success and 0 implies failure on that test.



(a) Rule 45, length: 500,000 bits



(b) Rule 45, length: 10,000,000 bits

FIGURE 4.2: The horizontal axis shows increasing width of the internal state. The only difference between Figure (a) and (b) is the length of the produced sequence. Test scores are coded as 1 and 0, where 1 implies success and 0 implies failure on that test.

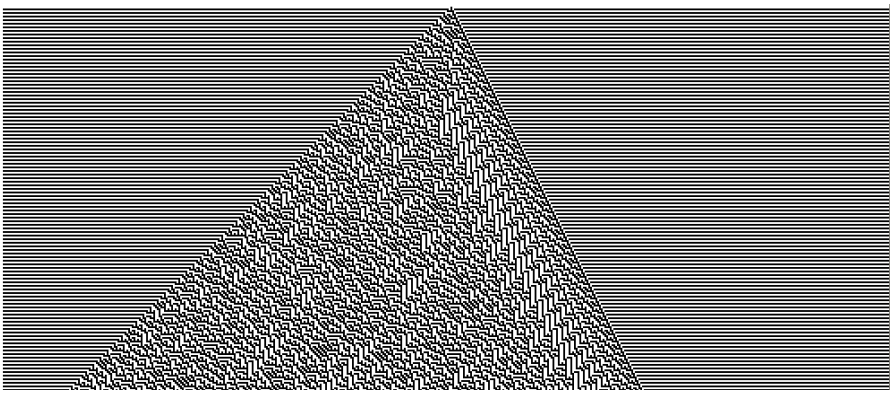


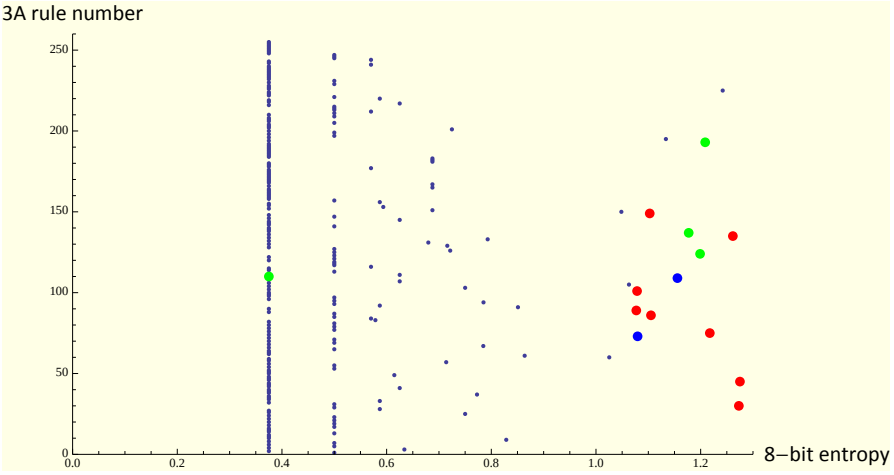
FIGURE 4.3: Rule 101 time-space diagram with 3B. The initial state has only one bit set. The right side of the triangle shows quite regular behavior while the left side appears to be more random.

TABLE 4.2: Test scores (out of 188) of Rule 101 with two different initial state. The width of state is 100-bit.

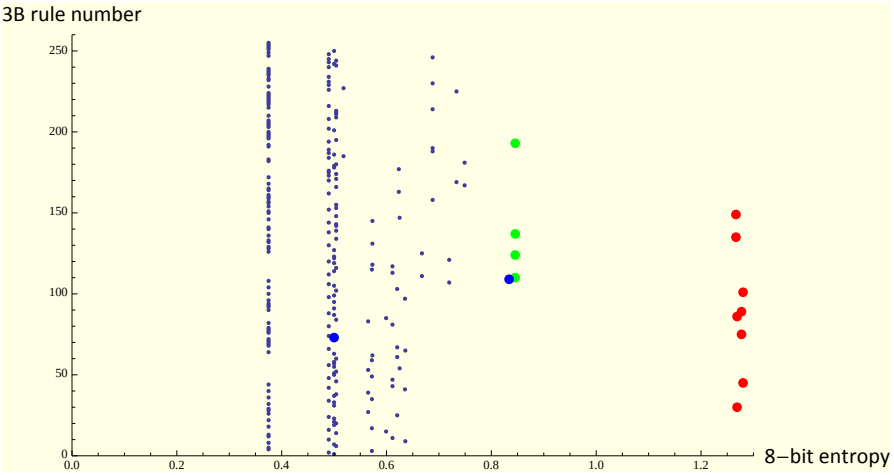
	TEST RESULTS		
Initial State	**3A**	**3B**	**3C**
00...010...00	24	185	187
Random initial state	24	185	188

on the surface of the cylinder. As long as one uses a cyclic boundary, outcomes of 3A, 3B and 3C will be dependent. Therefore, it is not surprising to see the same group of rules getting similar entropies with different neighborhood types. Figure 4.4 provides an example of this situation.

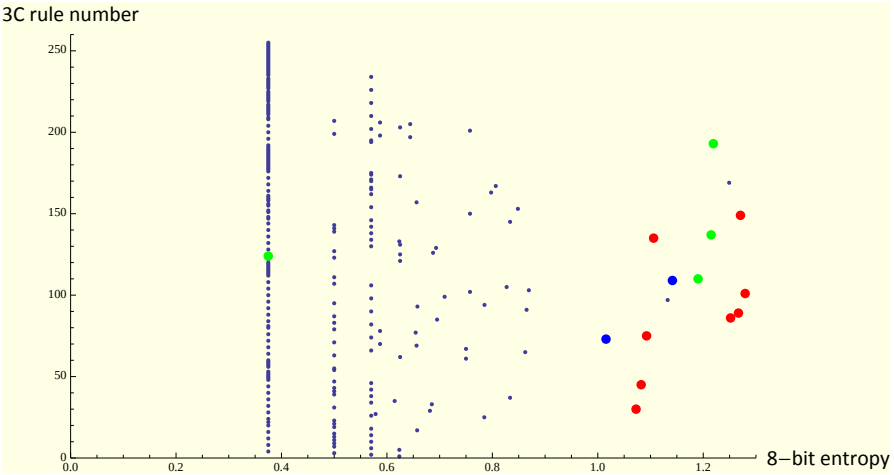
On the other hand, although the differences in entropy values for varying neighborhood schemes are small, there may be big differences in the test scores. For example, on the space-time diagram of Rule 101 in Figure 4.3, left side appears more random while right side of the triangle is quite regular. Therefore, both its entropy and test results are low with 3A neighborhood when tested with the same initial state as seen in Table 4.2. Entropy of Rule 101 with 3A neighborhood is lower than 3B and 3C but it is still among the high values in 3A rule set. This might be due to two reasons. Entropy values are computed on the first 2200 bits of the sequences while tests are conducted on 1,000,000 bits. Therefore the entropy value (as it is computed here) may not be a good indicator of the overall statistical quality. Second, the type of entropy measure used in this study may attain high values even for periodic sequences. Possibly, using an entropy measure based on compressibility may reflect the decrease in test performance better.



(a) 3-input CA rules with 3A neighborhood scheme



(b) 3-input CA rules with 3B neighborhood scheme



(c) 3-input CA rules with 3C neighborhood scheme

FIGURE 4.4: 8-bit entropies of 3-input CA rules with changing neighborhood schemes are plotted. Some high-entropy rules are marked with bigger dots. Red dots identify the rule families {30, 86, 135, 149} and {45, 75, 89, 101}. Green dots are used for {110, 124, 137, 193} and blue dots for {73, 109}. The marked rules are seen to have high entropies in all neighborhood formats.

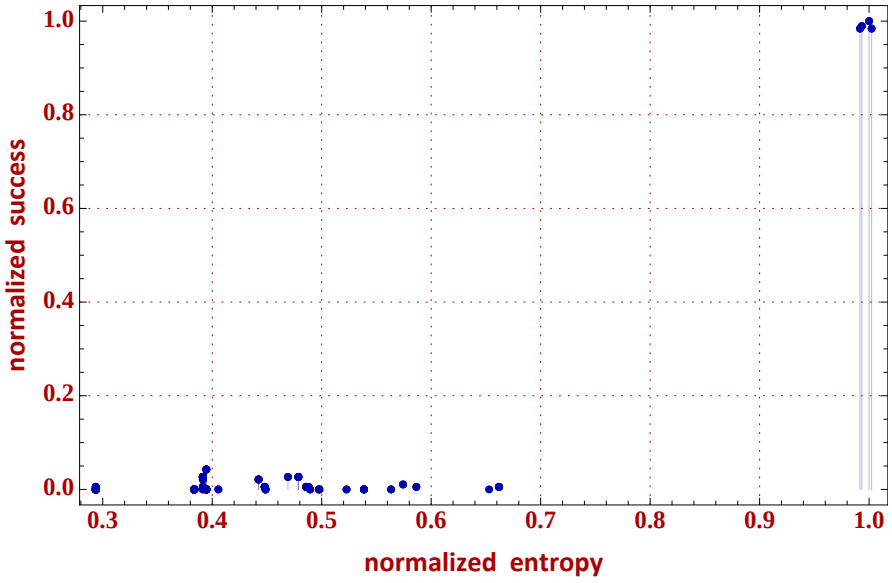


FIGURE 4.5: The relation between entropy and test results of 3B CA rules is shown. Every dot represents a function in 3B. Test results are computed with the parameters (100, 1000000, 11000, 9, 9, 5000, 16, 13).

4.6.3 Entropy vs. Statistical Quality

The data plotted in Figure 4.5 is derived from 3B rules. Setting up the parameters as specified, the test calculates 188 p-values within 15 tests. A normalized success rate is computed for a function by dividing the number of p-values indicating success by 188. The y-axis shows that normalized success rate and the x-axis shows the normalized 8-bit entropy values. There are 256 points on the figure, which are clustered in two regions. This is because both entropy and test scores indicate a sharp decrease after the 8 rules at the top, as can be seen in Table 4.3. The right top corner has those best 8 rules and the remaining rules lie at the opposite edge indicating their unsatisfactory performance on the test. There is no exceptional rule to the condition that superior statistical performance comes with high entropy.

The 8 rules that perform best are belong to 2 families: $\{30, 86, 135, 149\}$ and $\{45, 75, 89, 101\}$. The rules which are reflections of each other (e.g., 30-86, 75-89) have the same entropy and test scores because they produce the same sequence since the central bit is taken from each state. But the outputs of conjugates (e.g., 30-135, 86-149) are not identical because 1s and 0s are swapped in the generated sequence. Therefore conjugate pairs have different entropy and test scores but their values are still close to each other, implying their dependency.

Figure 4.6 shows the data of 602 rules which are selected randomly from the set of 4C rules via Mathematica RandomSample function. The rules with the best performance on the tests have entropies on the highest levels as shown in Tables 4.4 and 4.5. Though,

TABLE 4.3: The first 15 rules with the highest test score are listed with their entropy values. The significant increase in both entropy and test scores passing from Rule 60 to Rule 149 is a clear indication of the correlation between the entropy and statistical quality.

3B Rule	Test Score	8-bit Entropy
75	188	1.277176
89	188	1.277176
30	186	1.268756
86	186	1.268756
45	185	1.280186
101	185	1.280186
135	185	1.266644
149	185	1.266644
60	8	0.503955
102	8	0.503955
153	8	0.503955
195	8	0.503955
11	5	0.611352
43	5	0.611352
47	5	0.611352

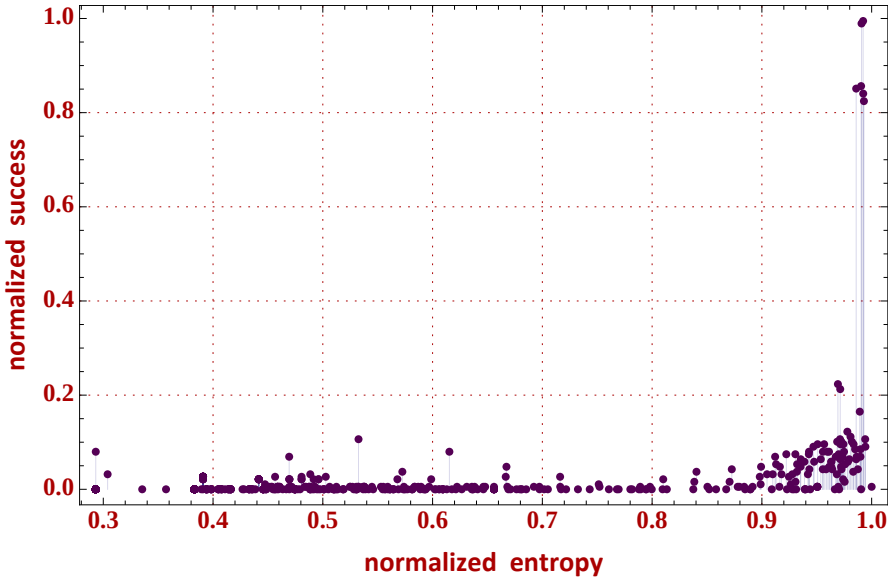


FIGURE 4.6: The purple dots indicate 602 rules drawn randomly from the 4C rules. Test parameters are (100, 1000000, 11000, 9, 9, 5000, 16, 13).

TABLE 4.4: Top 15 rules in the order of descending test scores. The maximum entropy of the set is 1.27921. Entropy values –as in the form it is computed here– are not directly proportional to the test success rate.

4C Rule	Test Score	8-bit Entropy
21866	187	1.269254
17595	186	1.267411
42342	161	1.266932
26982	160	1.261183
31110	158	1.269383
6104	155	1.270204
18300	42	1.239885
49437	40	1.242584
23149	31	1.265360
51001	23	1.250871
16702	21	1.254586
42281	20	1.242353
11734	20	1.271686
59467	20	0.681210
5347	19	1.243937

there are many others, accumulated on the right bottom of the graph, with high entropy and unsatisfactory results on the tests. That might be due to the ineffective entropy measurement taken here.

The entropy is computed on the first 2200 bits of the sequences while tests are conducted on 10^6 bits. When iterations are carried out for long enough, it is seen that some of the rules converge immediately to a constant state or enter into a periodic trend while others take more time before converging or never experience such a transition [54]. Those rules on the right bottom corner may possibly show a good performance on the first 2200 bits but over a 1 million evolution they start converging, therefore their test performance deteriorate.

Computing the entropy on the range that the test is performed would be a better indicator to see the relation between entropy and statistical quality. However, that will bring a high computational burden for the elimination process. On the other hand, if 8-bit entropy over a short sequence is taken up as a quick elimination method over a big space of functions, we will end up a crowded set of rules most of which will be discarded after the testing step.

4.6.4 Mutual Information vs. Statistical Quality

Below figures are presented to give an idea about how good it is to choose the rules with zero mutual information for testing to find out the ones with the best performance

TABLE 4.5: Top 15 rules in the order of descending entropies. The maximum test score is 187. Most of the rules –including the top 3– placed in the right bottom corner of the diagram in Figure 4.6.

4C Rule	Test Score	8-bit Entropy
55589	1	1.279210
33630	17	1.271925
11734	20	1.271686
6104	155	1.270204
31110	158	1.269383
21866	187	1.269254
17595	186	1.267411
42342	161	1.266932
21673	0	1.266932
22910	16	1.266275
11801	13	1.266146
23149	31	1.265360
42021	8	1.263136
18401	13	1.261755
11689	12	1.261214
26982	160	1.261183

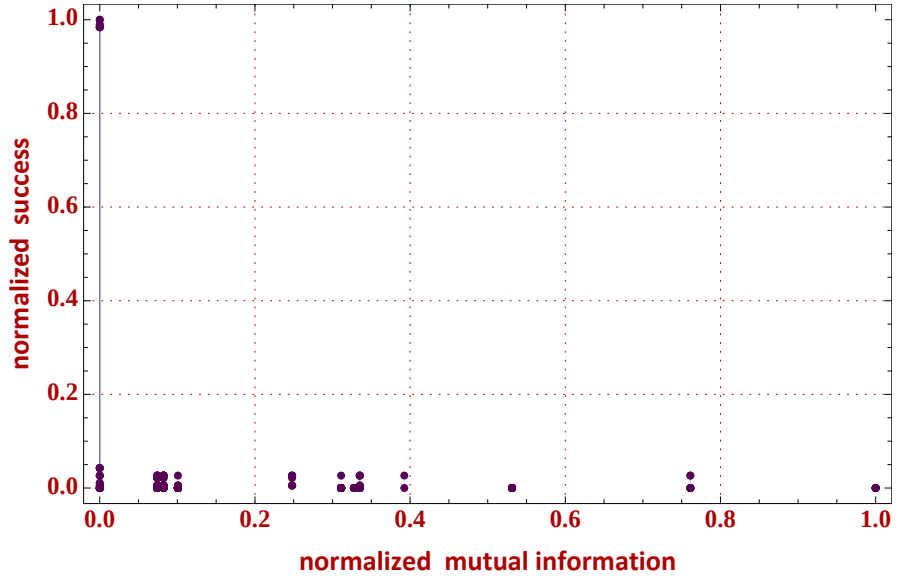


FIGURE 4.7: 3B functions are plotted with their test scores against mutual information values. Mutual information is expected to fall as randomness quality gets better. Most of the rules are located close to x-axis due to low test scores. The good ones are seen to have $\mathcal{I} = 0$.

on the statistical tests. For that purpose, the mutual information compared to the test scores are plotted for 3B, 4C and 4D rules. The data that belongs to 3B shows the performance of the whole set of 256 functions while 4C and 4D are represented by 602 randomly selected rules among the the whole set of 65,536 rules.

Figure 4.7 shows success rate of 3B rules compared to their mutual information ($\mathcal{I}$) which

TABLE 4.6: The best 15 rules according to their test scores. The good 8 rules at the top have zero mutual information. Overall, there are 36 rules in 3B that have zero mutual information. One third of them are located at the top of the table.

3B Rule	**Test Score**	**Mutual Information**
75	188	0
89	188	0
30	186	0
86	186	0
45	185	0
101	185	0
135	185	0
149	185	0
60	8	0
102	8	0
153	8	0
195	8	0
1	5	1.939212
7	5	0.209446
11	5	0.209446

is computed according to Section 3.6.3. Within the set of elementary rules, there are 8 functions which are far better than the rest in testing performance (see Table 4.6). These rules sit close to the point (0,1) and they have $\mathcal{I} = 0$. Therefore, making an elimination using $\mathcal{I} = 0$ seems to work fine in the set of 3B CA. Such an elimination will give us all rules with good randomness quality.

There are 42 rules in 4C group which pass every test and also have zero mutual information. None of those rules is present in the sample rule set. As Table 4.7 and Figure 4.8 show, rules with better random behavior can be distinguished by their mutual information.

There are 22 rules in 4D which pass every test and also have zero mutual information. Only one of them is among the randomly chosen sample of 602 functions. There is one more rule in the sample which scores 187 out of 188. Figure 4.9 and Table 4.8 show these two rules as having $\mathcal{I} = 0$. However, the third best rule in randomness quality does not have zero mutual information. In case of discarding the rules with non-zero $\mathcal{I}$, that one will be eliminated.

4.6.5 Entropy vs. Mutual Information

Before testing the 5-input CA rules, an elimination is performed to narrow down the set of rules to be tested. Only the rules with zero mutual information is selected for testing. The general method, however, is to choose the rules with high entropy. The reason for

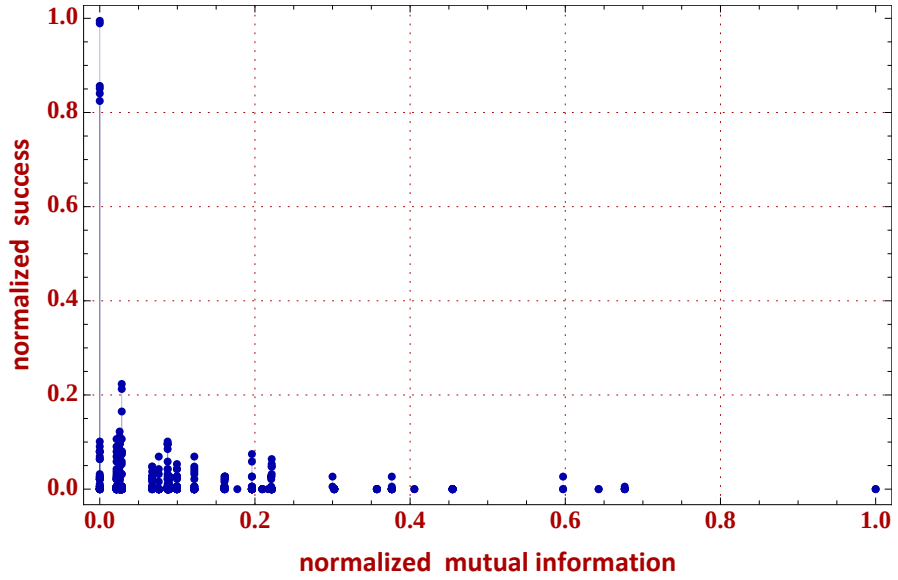


FIGURE 4.8: Mutual information vs. test scores of 602 rules with 4C neighborhood are plotted. Rules are chosen randomly. It seems that zero mutual information is an appropriate criterion to capture the good rules in 4C, too.

TABLE 4.7: Data belong to Figure 4.8. Entries are ordered in descending test scores. Among the rule set, 6 rules are outstanding in terms of the test scores and they all have zero mutual information.

4C Rule	Test Score	Mutual Information
21866	187	0
17595	186	0
42342	161	0
26982	160	0
31110	158	0
6104	155	0
18300	42	0.059652
49437	40	0.059652
23149	31	0.059652
51001	23	0.053956
16702	21	0.053956
42281	20	0.053956
11734	20	0.059652
59467	20	0.045566
5347	19	0.053956

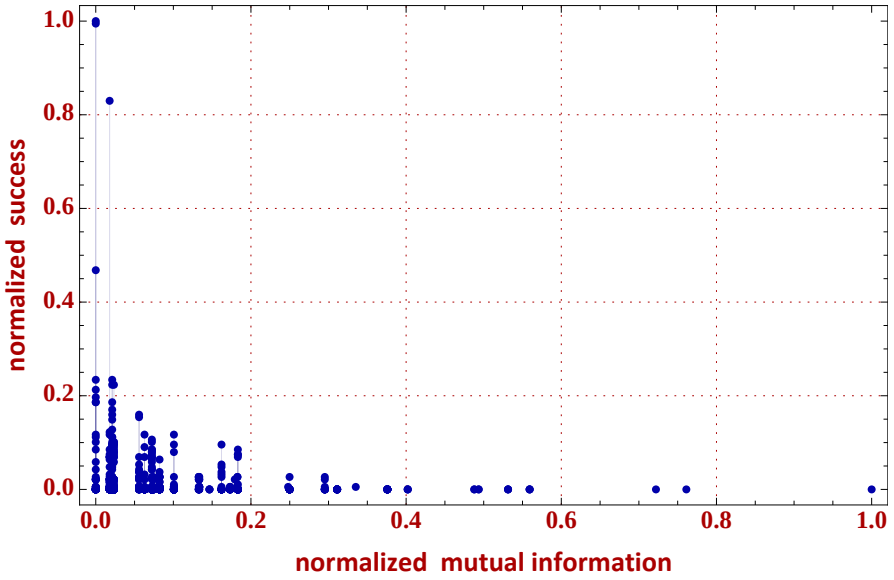


FIGURE 4.9: Randomly selected 4D rules with their mutual information and test scores. The best rules are seen to have $\mathcal{I} = 0$.

TABLE 4.8: The best 15 rules among the randomly selected sample according to test scores.

4D Rule	Test Score	Mutual Information
31110	188	0
17595	187	0
6104	156	0.045566
26985	88	0
26982	44	0
38935	44	0.053956
11734	42	0.053956
15587	42	0.059652
33630	40	0
7353	37	0
9159	35	0
17977	35	0.053956
18348	35	0
33085	32	0.053956
49437	30	0.142217

choosing mutual information is the simplicity of its computation on the rules. Contrary to the entropy, mutual information is directly computed on the truth table of a rule, with no operation on generated output. Figures 4.10, 4.11 and 4.12 are provided to show the distribution of rules with zero mutual information according to their 8-bit entropies.

Referring to Figures 4.10, 4.11 and 4.12, the rules are divided into ten blocks based on their entropy values. Table 4.9 shows the rate of rules which have zero mutual information to all rules in that block of the histogram. The block numbers from 1 to 10 increase with the increasing entropy values.

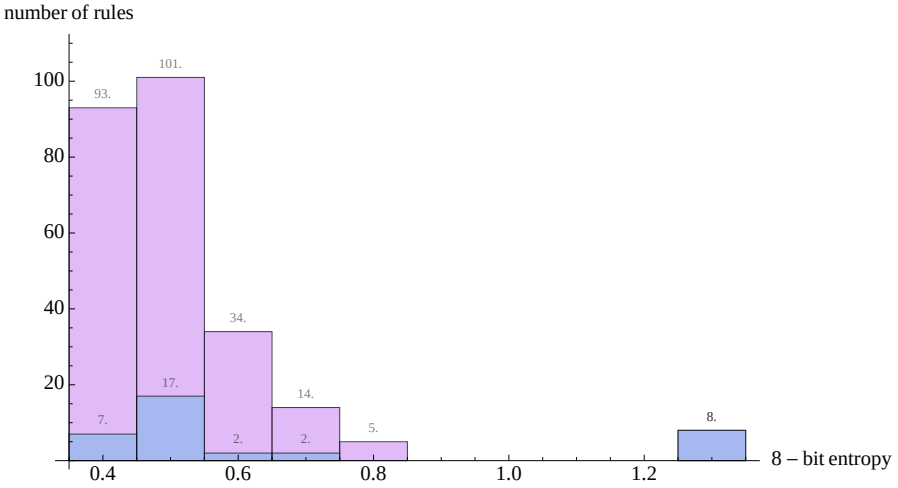


FIGURE 4.10: The figure shows two histograms combined. The purple one shows all 3B rules grouped according to their 8-bit entropy values. The blue one only shows the rules with zero mutual information among them. The leftmost column is all blue, meaning that all the functions within that interval have their mutual information zero.

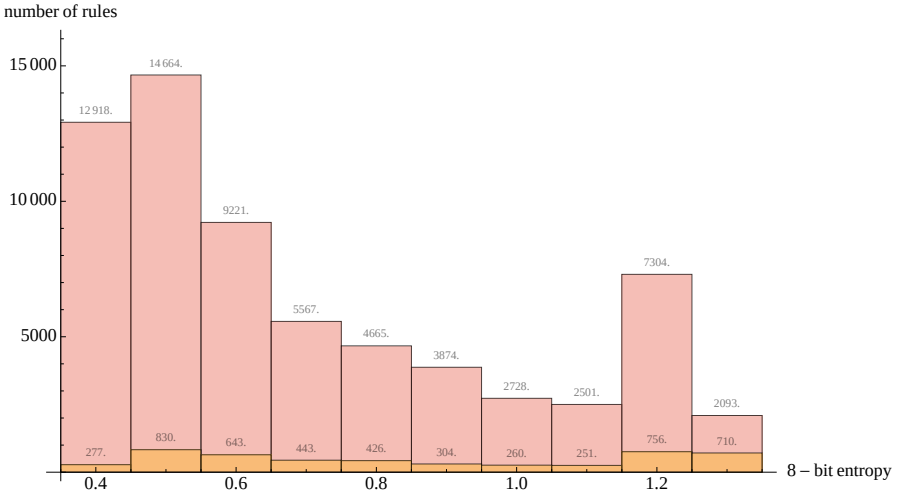


FIGURE 4.11: This figure presents the same kind of data with the figure 4.10 for the rules in 4C. The pink histogram shows all 4C rules grouped according to their 8-bit entropy values. The orange one only shows the rules with zero mutual information in that block.

TABLE 4.9: Dividing the rules in 10 blocks according to their entropies, table shows the rate of rules which have zero mutual information to the amount of rules in that block. The block numbers from 1 to 10 increase with the increasing entropy values.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
3B Rule	7.5%	16.8%	5.8%	14.2%	0%	–	–	–	–	100%
4C Rule	2.1%	5.6%	6.9%	7.9%	9.1%	7.8%	9.5%	10.0%	10.3%	33.9%
4D Rule	5.1%	4.3%	6.6%	6.4%	8.1%	7.8%	9.1%	10.5%	11.5%	20.4%

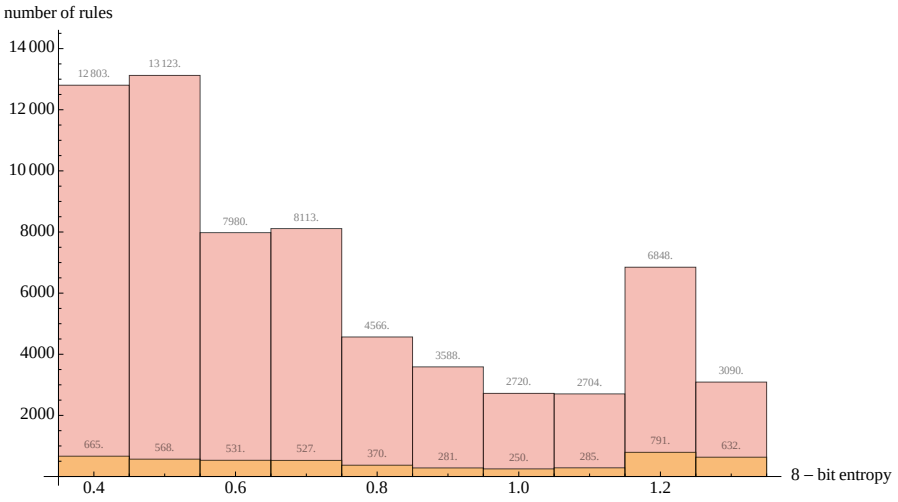


FIGURE 4.12: Entropy histogram for all 4D rules are shown in pink. The orange one shows the amount of rules with zero mutual information in each block.

When choosing an elimination criterion, the goal is to find one which could narrow down the set as much as possible while keeping the good rules within. As we know from the earlier plots, 8-bit entropy selects the good rules as well as many others with unsatisfactory test performance. Zero mutual information, on the other hand, seems to be an efficient criterion though it may miss some good rules as shown in Table 4.8. Still, the best rules have zero mutual information. Moreover, higher percentage of the rules are selected as entropy increases, as Table 4.9 indicates.

Surely, the data presented so far provide no guarantee for mutual information to work with the same efficiency on 5-input CA as it does on 3 and 4-input CA. However, since entropy is a reliable indicator of randomness, the increase in the percentage seen in Table 4.9 may be considered as a positive factor on behalf of using zero mutual information as an elimination criterion.

4.6.6 Overall Test Results of 4- and 5-input CA

If a categorization is to be made among the tests, roughly three groups may be distinguished[1]. Type-1 tests focus on the distribution of ones and zeroes on the overall string. These are Frequency, Block Frequency, Cumulative Sums, Runs, Longest Run of Ones, Random Excursions and Random Excursions Variant Tests. Type-2 tests work on the substrings of certain length and their distribution. These are Non-overlapping T. M., Overlapping T. M., Approximate Entropy and Serial Tests. The most challenging is the Non-overlapping T. M. Test since it makes 148 different checks on the sequence. Type-3

[1]This grouping is completely based on the views of the author and does not rely on any reference study about the NIST Test Suite.

TABLE 4.10: The data is derived from the test results of 4-input CA rules. The first row shows the number of functions that are selected for testing. The selected 4900 rules have zero mutual information in their neighborhood type. The second row shows the amount of functions that pass the Frequency Test. Note that passing the Frequency Test is a must for proceeding the other tests. The third row shows the number of functions that pass all tests.

	4C Rules	**4D Rules**
Tested Functions	4900	4900
Frequency	848	899
All Tests	42	22
Block Frequency	828	844
Cum. Sum Forward	843	891
Cum. Sum Backward	844	890
Runs	396	228
Longest Run of Ones	321	185
Rank	335	321
Spectral	312	173
Non-overlapping T. M.	78	42
Overlapping T. M.	313	171
Linear Complexity	339	340
Universal	322	180
Serial	303	167
Approximate Entropy	305	166
Random Excursions	194	113
Rand. Exc. Variant	203	172

tests look for the correlation between different parts of the output string. These are Linear Complexity, Spectral, Universal and Rank Tests.

Table 4.10 presents the number of rules that pass each one of the tests. Test parameters are set as (100, 1000000, 11000, 9, 9, 5000, 16, 13).

Frequency and Cumulative Sum Tests check for mild conditions on the output sequence. Compared to 4C, 4D rules are less successful on the other Type-1 tests. Similar results are also seen in Type-2 tests, which check for more stringent conditions. Looking at Type-3 tests, Rank and Linear Complexity scores are close for both 4C and 4D. These two tests are looking for the linear dependency between the subsequences of the output sequence. But in Spectral and Universal Tests, again many of the 4D rules are eliminated.

Table 4.11 presents some information derived from 5-input CA test results. The first row shows the number of rules that are tested. After obtaining the set of rules with zero mutual information, conjugates of the rules in the set are removed. Therefore the values on the first row differs for each neighborhood type.

Test parameters for 5-input CA are (64, 39000, 400, 9, –, –, 12, 9). During the testing process, the most time-consuming operation is generating the sequence. To test nearly

TABLE 4.11: The data is derived from the test results of 5-input CA rules. The first row shows the number of functions that are selected for testing. The second row shows the amount of functions that pass the Frequency Test. The third row shows the number of functions that pass all tests. 5 tests in the NIST Test Suite cannot be applied due to parameter restrictions.

	5A Rules	**5B Rules**	**5C Rules**
Tested Functions	82,824,885	82,824,889	82,824,890
Frequency	11,445,493	10,578,610	12,456,946
All Tests	18,271	14,419	16,576
Block Frequency	10,764,890	9,687,158	11,599,210
Cum. Sum Forward	11,250,306	10,371,535	12,247,651
Cum. Sum Backward	11,223,704	10,347,984	12,223,965
Runs	1,608,756	2,067,360	1,946,528
Longest Run of Ones	2,207,804	1,617,890	1,578,562
Rank	6,042,984	6,023,546	6,228,668
Spectral	4,920,170	4,287,088	4,200,295
Non-overlapping T. M.	35,119	18,385	20,834
Serial	1,114,511	297,452	349,953
Approximate Entropy	488,307	143,000	252,195

248 millions of functions, sequence length and state width need to be reduced. Using the specified parameters, Universal, Overlapping T. M., Linear Complexity, Random Excursions and Random Excursions Variant Tests are not applicable. Therefore they were omitted. This means that two tests from Type-1 and Type-3, one test from Type-2 were not applied.

Generally tests are applied on sequences with length at least 1 million bits. Passing all the tests is quite challenging with 39,000-bit sequences because of the initial state used in this study. The initial state is set to all zero except the central bit of the state. Using a state of random bits gives chance to weak rules for passing more of the tests. Note that only 2 rules (excluding the reflections and conjugates) among the 3B CA could generate satisfactory results under these conditions.

There are 120 rules in 5-input CA which pass all the tests with all neighborhood types 5A, 5B and 5C. 114 of these rules have rule numbers such that in the binary format of the rule number, every 4-bit chunks include 2 zeros and 2 ones. Now, we will present statistical performance of one of these rules with 500 different initial states.

4.7 The simplest rule: 1435932310

The key advantage of producing random sequences via CA is their simplicity. Rule 30 owes its fame to its simlicity as well as its extraordinary potential for generating randomness.

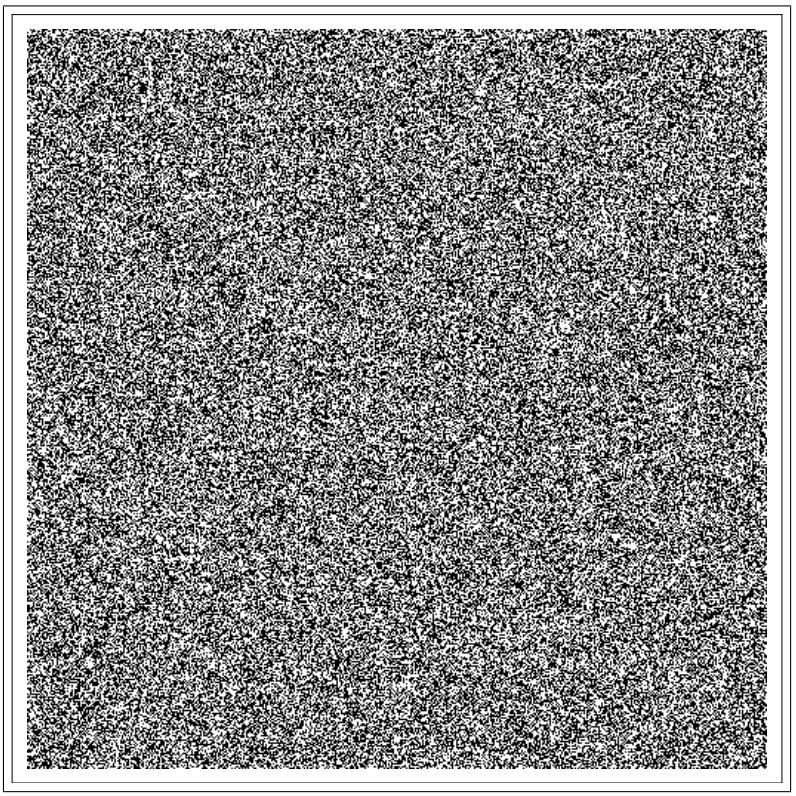


FIGURE 4.13: A random sequence generated by Rule 1435932310 is plotted as a 500x500 array. This sequence is produced with an 64-bit initial state which is all zero except the central bit. Neighborhood scheme is 5C. The part in the plot is the interval between 500,000$^{\text{th}}$ and 750,000$^{\text{th}}$ bits.

Among all 5-input CA which have zero mutual information, 120 rules passed all the tests with all neighborhood types. In Appendix B, these rules are listed with their rule numbers and the shortest Boolean formulae. The formulae are the shortest ones that can be written in terms of the logical operators AND, OR, XOR and negation. They are generated via [56].

A 5-input Boolean function requires at least 4 logical operators. Among 120 rules, there are three rules with 4 operators and only one of them does not have any negation. It has the rule number 1435932310. To present an individual example, we examined the statistical performance of this rule with 500 different random initial states. Test data are evaluated as described in Sections 4.3.1 and 4.3.2. The results are presented in Tables 4.12, 4.13 and in Figures 4.14, 4.15 .

Beside the statistical testing, using the sequence in an application (for example, in a Monte Carlo integration) to compare its performance with truly random sequences or directly plotting the sequence may give idea about its quality. Plotting is a useful way of detecting spatial dependencies. Below in Figure 4.13, a plot of the sequence generated by Rule 1435932310 is provided.

Table 4.12 shows the evaluation of the test results. According to the test parameters, 11 is the upper bound for failures that can be tolerated to be considered as successful in the "Rate of Success" method. The second method require P_τ to be greater than 0.0001 for success. The rule gets the lowest scores from Approximate Entropy Test in both evaluation schemes.

Table 4.13 shows the results of the tests which return multiple p-values. Their results are determined using the "Rate of Success" method only. Our testing procedure requires every p-value to imply success in order for the rule to be considered successful on that test. This means that, we set the threshold at 100%. But it is quite common in the literature to set lower thresholds. Therefore, results according to the lower rates are also stated. Random Excursions and Random Excursions Variant Tests are evaluated over 311 trials because they were not applicable to the 189 of the sequences due to a reason related with the structure of the tests.

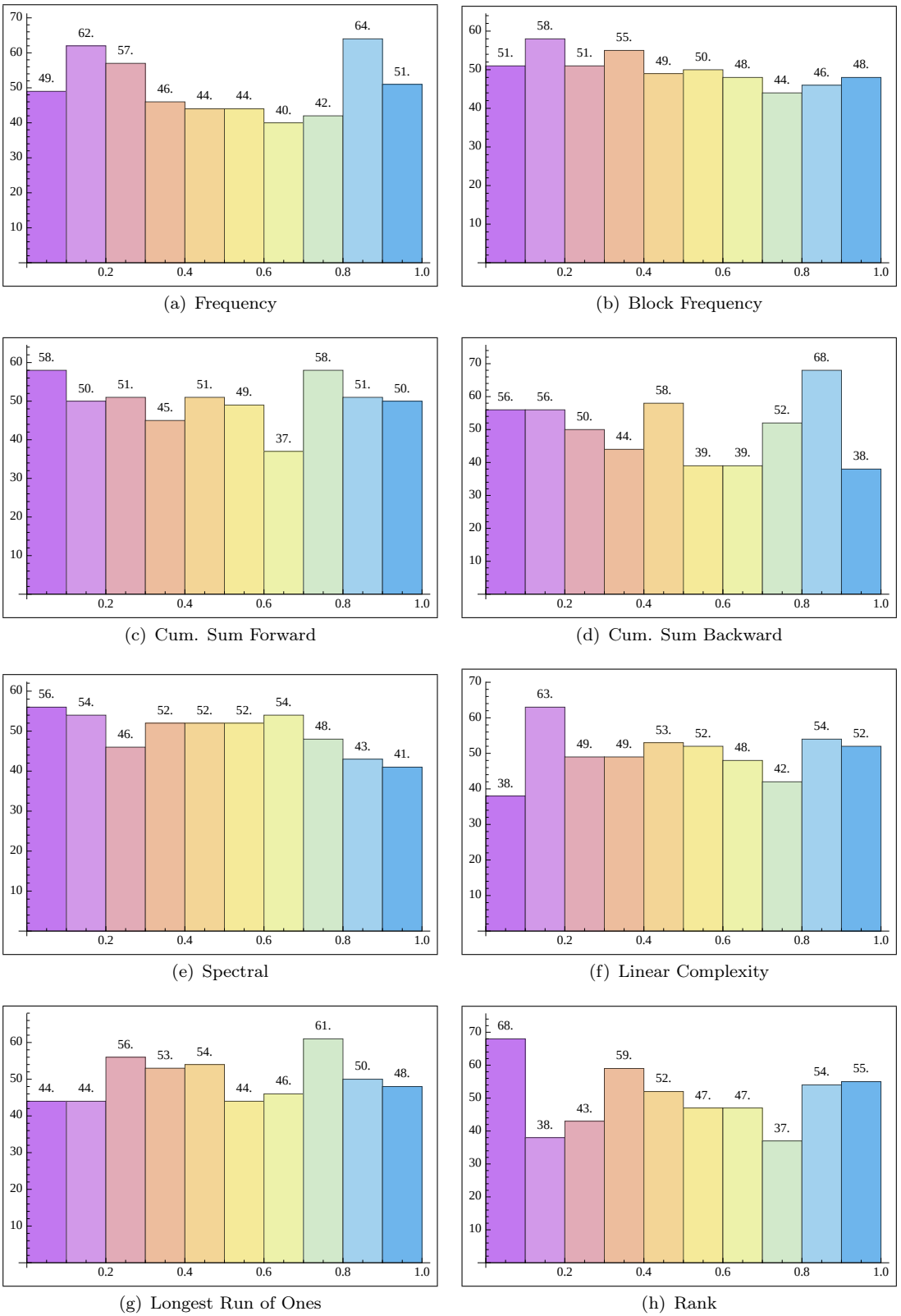
TABLE 4.12: Test results of Rule 1435932310 according to the two evaluation schemes. The central column shows the number of failures out of 500 trials. The leftmost column is P_τ that is computed separately for each test on 500 p-values.

	Rate of Success		**P-val Distr.**	
Test Name	**Result**	**# Failures**	**Result**	**P_τ**
Frequency	Success	5/500	Success	0.187581
Block Frequency	Success	3/500	Success	0.962688
Cum. Sum Forward	Success	6/500	Success	0.686955
Cum. Sum Backward	Success	6/500	Success	0.038565
Runs	Success	6/500	Success	0.530120
Longest Run of Ones	Success	8/500	Success	0.719747
Spectral	Success	4/500	Success	0.914025
Rank	Success	4/500	Success	0.055361
Overlapping T. M.	Success	4/500	Success	0.715679
Linear Complexity	Success	4/500	Success	0.502247
Universal	Success	8/500	Success	0.206629
Approximate Entropy	Success	11/500	Failure	0.000026

TABLE 4.13: Test results of Rule 1435932310. The tests which turn in multiple p-values are evaluated based on the failure rate.

Treshold	**Serial**	**Non-overl.**	**Rand. Exc.**	**Rand. Exc. V.**
80%	Success (8/500)	Success (0/500)	Success (1/311)	Success (6/311)
90%	Success (8/500)	Success (0/500)	Failure (29/311)	Failure (16/311)
95%	Success (8/500)	Success (0/500)	Failure (29/311)	Failure (24/311)
100%	Success (8/500)	Failure (387/500)	Failure (29/311)	Failure (24/311)

Figures 4.14 and 4.15 show p-value distribution of 500 trials. The diagrams 4.15(e), 4.15(f), 4.15(g) and 4.15(h) belong to the tests returning multiple p-values. One of the p-value is chosen for the plots for representation purpose.



(a) Frequency

(b) Block Frequency

(c) Cum. Sum Forward

(d) Cum. Sum Backward

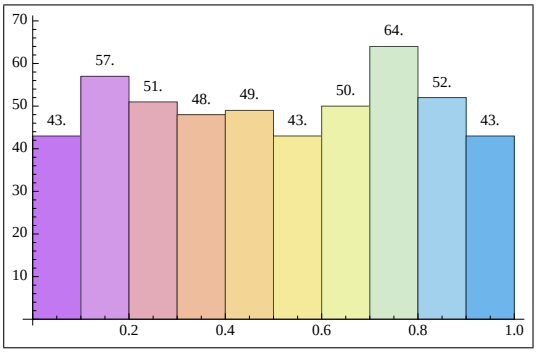
(e) Spectral

(f) Linear Complexity

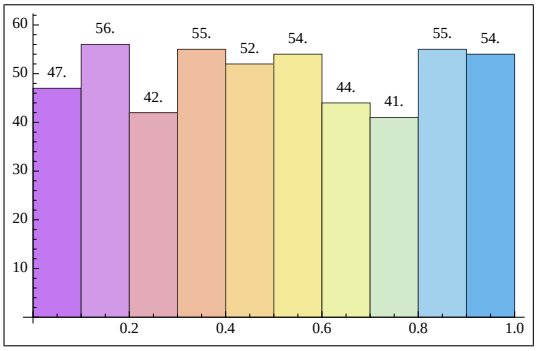
(g) Longest Run of Ones

(h) Rank

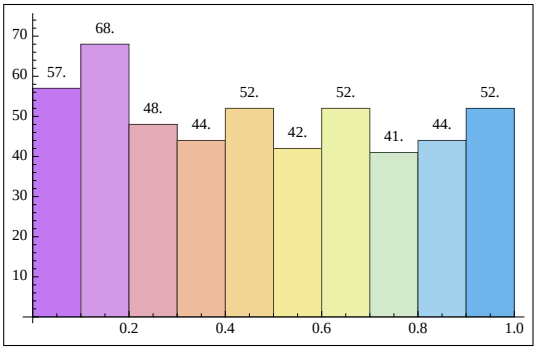
FIGURE 4.14: Distribution of 500 p-values for in each test of Rule 1435932310. Uniform distribution is expected for success.



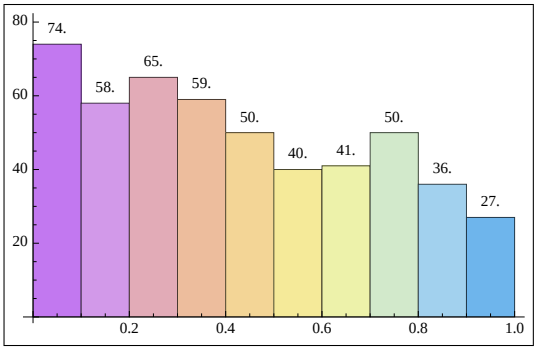
(a) Runs



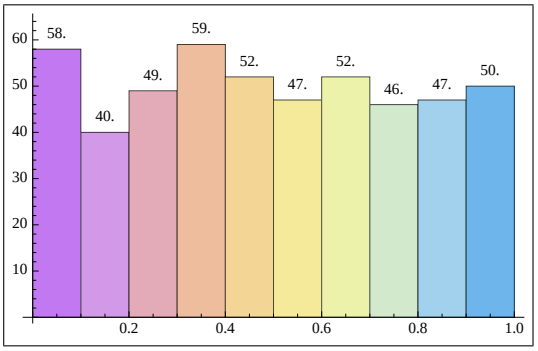
(b) Overlapping T. M.



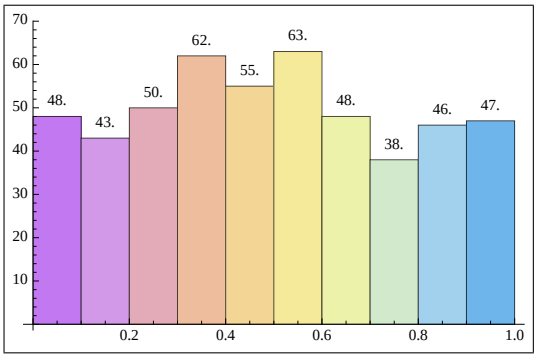
(c) Universal



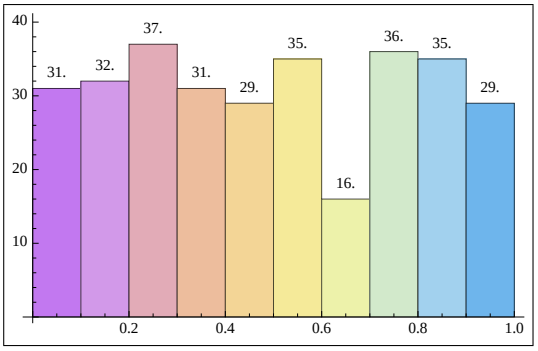
(d) Approximate Entropy



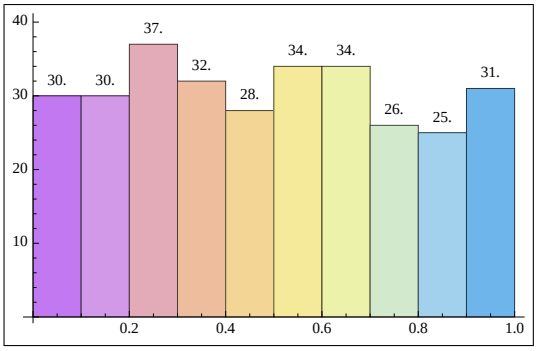
(e) Serial



(f) Non-overlapping T. M.



(g) Random Excursions



(h) Rand. Exc. Variant

FIGURE 4.15: Distribution of p-values for in each test of Rule 1435932310. Uniform distribution is expected for success.

Chapter 5

Conclusion

CA is favorable for random number generation basically for its simplicity and speed. CA-based PRNGs are generally constructed in hybrid form since hybrid designs are better with regard to statistical quality, unpredictability and invertibility.

At the step of choosing CA rules for PRNGs, two approaches are common: either linearity or high entropy is a reason for selection. Linear rules are preferred for they are analyzed mathematically to a great extent so that their advantages and potentiality have been observed well. On the other hand, high entropy rules are simple and useful, yet full of surprise since they lack a comprehensive analysis. Therefore the second kind is not recommended for security-purposed usage.

Most of the studies in this realm deals with the elementary rules. Basically there are two rule in the elementary CA set which have outstanding performance in statistical tests. 4-input CA are not studied so much, but there are some examples. This study aimed to research on 5-input CA to find out the rules eligible for random number generation.

In practice, it is possible to construct n-input linear rules with no search over the n-input CA set. But, high-entropy rules need to be found out through exhaustive search since there is no short-cut for constructing them or detecting them looking at their truth tables. 3- and 4- input rule sets have reasonable sizes for making such an exhaustive search. The set of 3-input CA has 256 rules and 4-input CA have 65,636 rules. 5-input CA set has more than 4.2 billions of rules which makes it impossible to search over the individual rules.

One of the methods to be followed in such a case is to eliminate the rules which have bad randomness quality, then continue with a narrow set to perform exhaustive search. In the literature, entropy is commonly used as a measure to indicate randomness quality. Entropy is a measure to be applied on the generated sequence not on the CA rule itself.

This study uses mutual information instead, which is computed directly on the rule's truth table. This choice implies that mutual information of a rule is used here as an indicator of the randomness quality of the sequence which is to be generated by that rule. To the best of our knowledge, such an approach on CA was only encountered in some physics articles in 1990s. But those works assume higher values for number of inputs to the rules and the number of different cell contents.

In this study, we first performed statistical tests on 3- and 4-input CA to see how good it is to make an elimination basen on mutual information compared to entropy. Considering the test scores collected on 3B, 4C and 4D CA, mutual information is an effective criterion for discarding the rules with unsatisfactory performance on the statistical tests. An evaluation over the number of p-values implying success on the tests shows that the rules which have at least a success rate of 83% have zero mutual information. It should be noted that, the data giving these results are collected on 3B rules and a randomly chosen set of functions from 4C and 4D rules.

On the other hand, it is not possible to make a strict correlation between mutual information and randomness quality based on the data used here for two reasons:

1. To reach a reliable conclusion about using mutual information as an indicator of randomness quality, one option may be comparing it with entropy. To compute the entropy of a sequence, there are several different measures available. Here, we computed an 8-bit entropy on the first 2200 bits of the generated sequences. The tests are performed on a sequence of 10^6 bits. Both entropy and mutual information were useful to select the good rules as we observed. But it should also be considered how well a high entropy (in the form it is computed here) corresponds to a high randomness quality.

2. Another option to see if zero mutual information is a reliable criterion may be comparing mutual information values with statistical test scores. It is also controversial if a single test is enough to decide on the randomness quality. Note that, in this study we performed the statistical tests once for each rule.

Running the test suite with several initial states is not possible in our case due to the amount of rules to be tested. Though multiple testing is required to reach a conclusion about statistical quality, we believe that a single test gives considerable information. Because, generally a rule either passes most of the tests or fails in most of them. Also, the initial state we used[1] is commonly preferred for testing to see if a function is capable of producing randomness even starting from a non-random state. It is quite challenging

[1]The central bit is set, all the others are zero.

to pass all the tests under these conditions. Therefore the rules passing all statistical tests are worthy of interest for generating randomness.

After making some other eliminations on 5-input CA, the tests were performed on 82 millions of functions from each of the 5A, 5B and 5C neighborhoods which sum up to more than 248 millions of rules in total. Passing the frequency test was compulsory for the other tests. For each neighborhood type, 10 - 12 millions passed the frequency test and 14 - 18 thousands of rules passed all of the tests.

The relation between different neighborhood types seems trivial if cyclic boundary is used. As we included in Section 4.6.2, same group of rules score high in all neighborhood types and exceptions have rule-specific explanations. In the set of 5-input CA, the number of rules passing all the tests differs by nearly 2000 or 4000 between different neighborhoods. We cannot make any comment using the data we collected about whether this level of difference arises from a substantial generalizable reason. A more comprehensive and thorough research is required to reach a conclusion about this issue.

Leaving all questionable details behind, we ended up with 120 rules which pass all the tests in all neighborhood schemes 5A, 5B and 5C. We believe these are the ones that deserve attention on the subject of random sequence generation. Among them, we took up the one with the shortest Boolean formula and tested it with several times (500 times) with various random initial states, as NIST recommended. Its score is on a critical level in Approximate Entropy test but it easily passed all the other tests which returns one p-value. The tests returning multiple p-values are fulfilled if the target success rate is set as 80%. If 95% is aimed, there will be two failures.

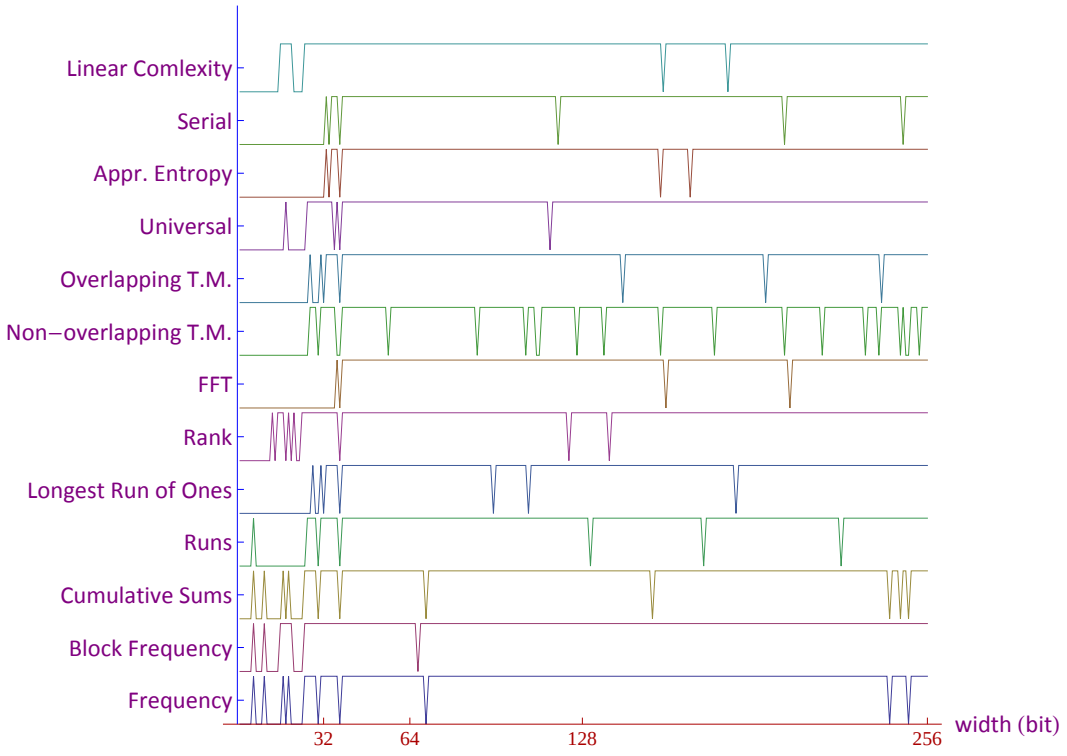
The testing process produced big data about 5-input CA rules. The part we presented here is quite limited. More visualized data could be presented. But commenting on the data is so hard since there is little information about these rules. Making a generalization or directly reaching a conclusion on the subject requires more research in depth and a strong dedication. Knowing this from the very beginning, we started such an empirical study to present our observations gained through the process.

Considering the scope of this study, stating all rule numbers passing the tests or their test scores may create a redundant data. Therefore we only presented 120 rules. Their performance could be improved by efficient selection function combinations or hybrid designs. We hope that what is included in this study could be useful for further RNG studies.

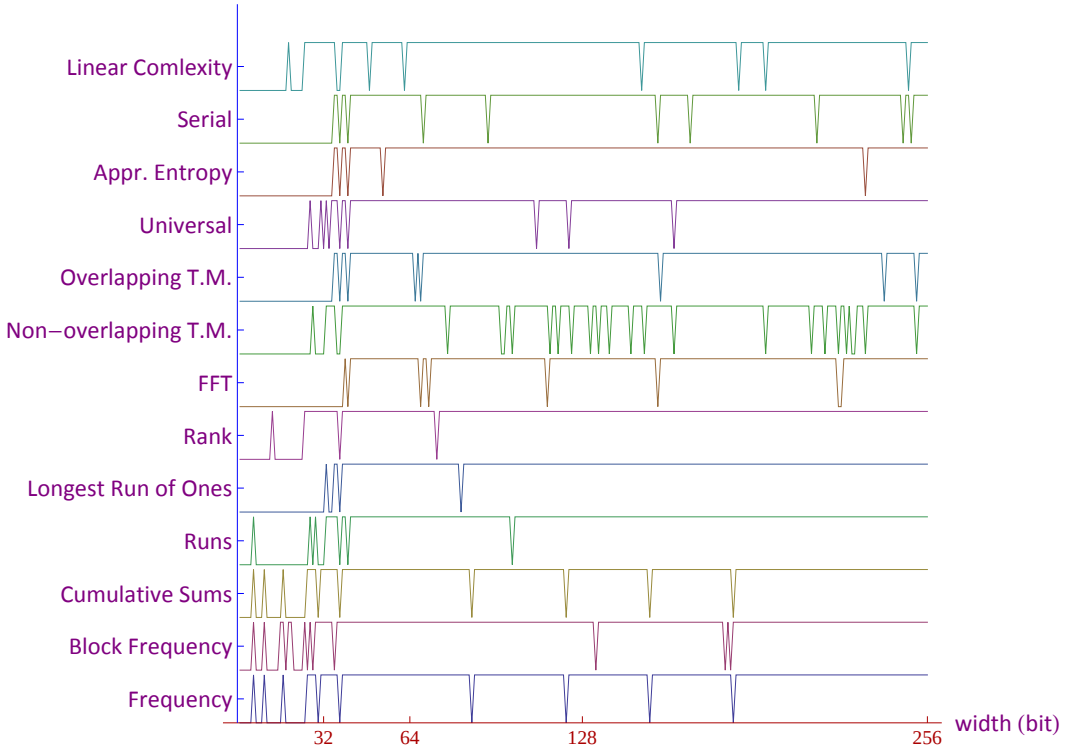
Appendix A

Test Results for Rule 30 and Rule 45

Figures A.1 and A.2 present the effect of state width in test results for Rules 30 and 45. These diagrams can be compared with Figures 4.1 and 4.2 in Chapter 4 to see the change in test results with varying sequence lengths.

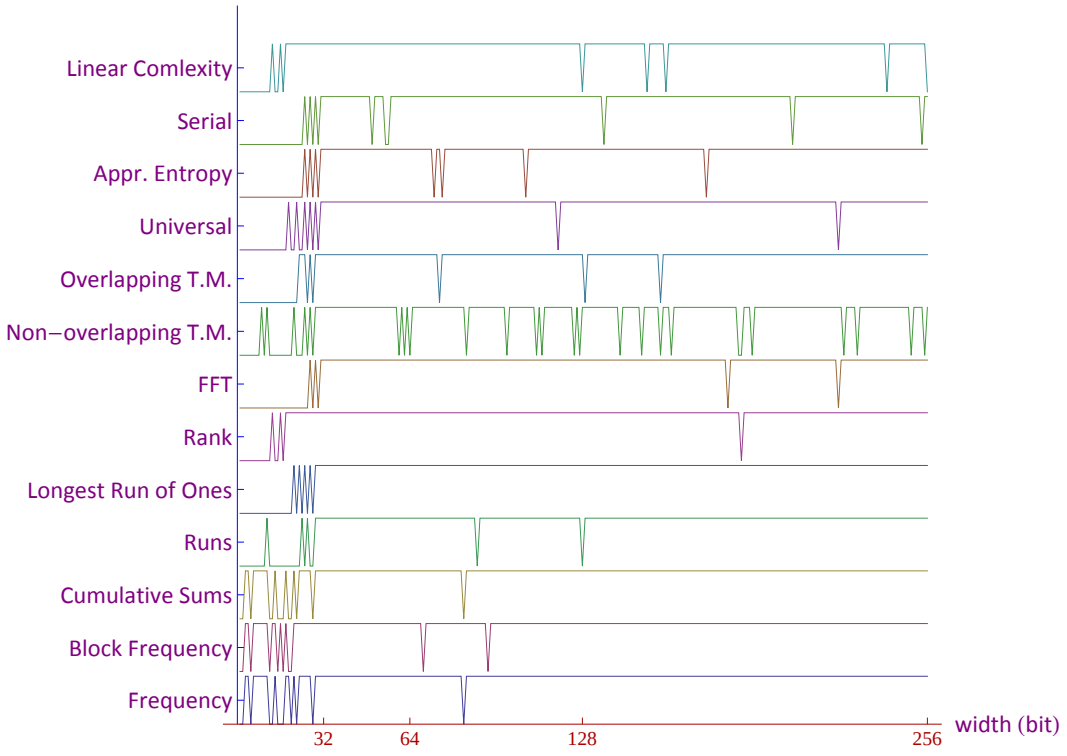


(a) Rule 30, length: 1,000,000 bits

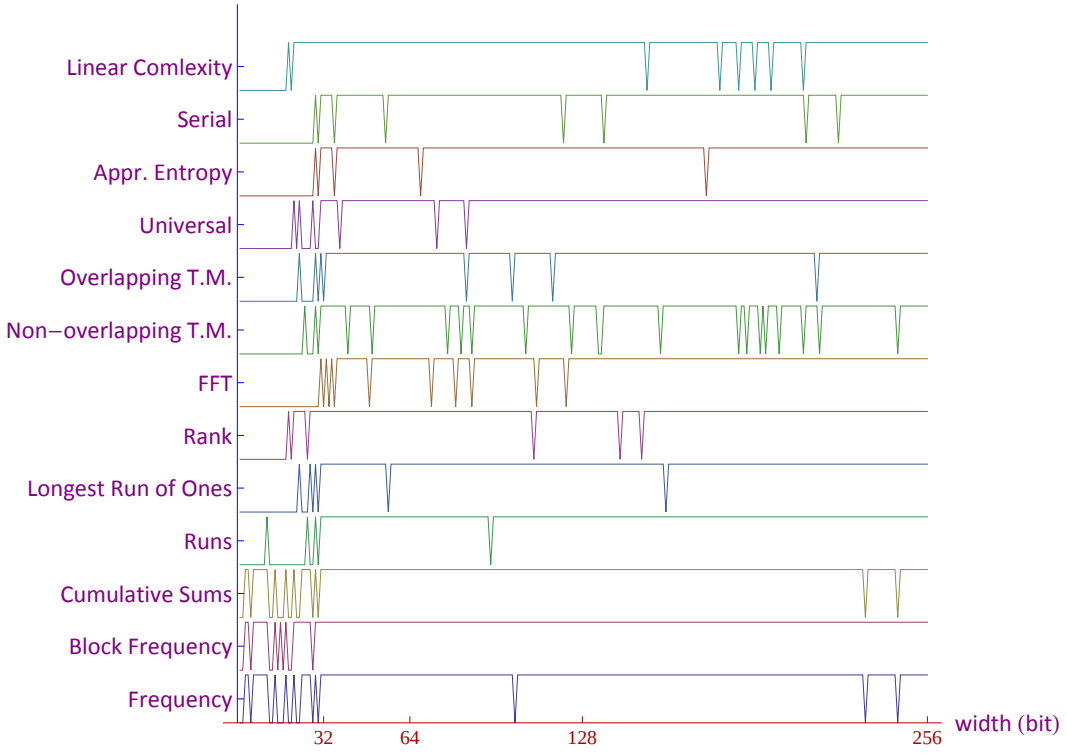


(b) Rule 30, length: 5,000,000 bits

FIGURE A.1: Test results of Rule 30 for varying state widths.



(a) Rule 45, length: 1,000,000 bits



(b) Rule 45, length: 5,000,000 bits

FIGURE A.2: Test results of Rule 45 for varying state widths.

Appendix B

120 Rules with their Shortest Boolean Formulae

Tables B.1, B.2, B.3 and B.4 show 120 5-input CA rules that are mentioned in Section 4.7 as passing all tests with all neighborhood types. Boolean formulae are written in terms of logical operations AND, OR and XOR, which are denoted as $\wedge$, $\vee$ and $\veebar$, respectively. Negation operator is denoted with $\neg$. These are the shortest formulae for each rule, as provided by [56].

Rule Number	**Shortest Boolean Formula**
2845202858	$(\neg y \vee (\neg x \wedge w)) \veebar v \veebar (\neg z \vee (x \veebar (y \vee \neg w)))$
2778306202	$\neg v \veebar \neg x \veebar ((w \vee z) \wedge (y \vee (\neg x \veebar w)))$
2774161754	$x \veebar \neg v \veebar (\neg y \vee (\neg z \wedge (x \vee w)))$
2774112602	$v \veebar ((\neg w \wedge (y \veebar z)) \vee (y \veebar (x \vee (y \wedge \neg z))))$
2594597274	$((x \wedge y) \vee (w \wedge z)) \veebar \neg v \veebar ((w \veebar z) \vee (\neg x \wedge \neg y))$
2594593114	$v \veebar ((\neg z \wedge x) \vee ((y \veebar w) \wedge (x \veebar \neg w \veebar z)))$
2593548698	$(y \wedge \neg z) \veebar \neg w \veebar \neg v \veebar (x \vee (w \wedge (\neg y \veebar z)))$
2589354406	$\neg z \veebar (x \vee (\neg w \veebar z)) \veebar \neg v \veebar (\neg y \vee (\neg w \wedge (z \vee x)))$
2589349290	$\neg y \veebar v \veebar (\neg z \vee (\neg w \wedge (\neg x \veebar y)))$
2577033626	$(\neg x \vee \neg y \vee z) \veebar v \veebar (w \vee (\neg x \wedge \neg y))$
2576771686	$w \veebar \neg v \veebar ((\neg x \wedge \neg y) \vee (\neg z \wedge (x \vee \neg w)))$
2573883046	$y \veebar \neg z \veebar v \veebar ((w \vee \neg z) \wedge (x \vee (\neg y \veebar w)))$
2573817258	$\neg v \veebar (\neg w \vee \neg z) \veebar (y \vee (z \wedge \neg x))$
2527685286	$\neg x \veebar y \veebar \neg v \veebar ((w \veebar z) \vee (y \veebar (x \vee (y \wedge w))))$
2527423078	$\neg v \veebar \neg x \veebar ((\neg y \vee (\neg w \veebar z)) \wedge (z \vee (x \veebar w)))$
2526696794	$v \veebar ((x \vee y) \wedge (\neg z \vee (x \veebar w)))$
2523568486	$w \veebar \neg y \veebar v \veebar (\neg z \vee x \vee (\neg y \wedge \neg w))$
2523309670	$w \veebar \neg v \veebar (\neg x \vee \neg y \vee (\neg w \wedge \neg z))$
2523305382	$(x \vee \neg y) \veebar v \veebar ((\neg y \veebar w) \vee (\neg z \wedge (\neg x \veebar w)))$
2522191206	$y \veebar v \veebar ((w \vee z) \wedge (\neg y \vee (\neg x \veebar w)))$
2509858198	$(\neg w \vee (\neg x \wedge y)) \veebar \neg v \veebar (z \vee (x \veebar y))$
2509679014	$\neg v \veebar ((\neg z \vee (\neg x \veebar w)) \wedge (x \vee (\neg y \veebar w)))$
2509658474	$v \veebar (\neg w \vee (\neg z \wedge \neg x)) \veebar ((\neg y \vee w) \wedge (\neg z \vee \neg x))$
2509592166	$v \veebar ((w \veebar z) \vee ((x \veebar w) \wedge (y \vee z)))$
2506726038	$(x \vee (y \wedge w)) \veebar w \veebar \neg v \veebar (\neg z \vee (x \wedge y))$
2506725722	$v \veebar \neg x \veebar ((\neg z \vee (y \veebar w)) \wedge (\neg y \vee (x \veebar w)))$
2506708390	$(\neg x \vee \neg w \vee \neg z) \veebar \neg v \veebar (y \vee (\neg x \wedge (w \veebar z)))$
2506529430	$(\neg y \vee (\neg x \wedge w)) \veebar \neg w \veebar \neg v \veebar (z \vee (\neg x \veebar y))$
2506462554	$x \veebar v \veebar ((y \vee z) \wedge (\neg x \vee (\neg y \veebar w)))$
2476304742	$\neg v \veebar z \veebar w \veebar (x \vee \neg y \vee (\neg v \wedge z))$

Table B.1: List of 120 rules, part 1

Rule Number	**Shortest Boolean Formula**
2099319166	$(\neg w \wedge (v \veebar (y \vee (z \veebar x)))) \vee ((\neg y \veebar z) \wedge (\neg w \veebar (\neg v \vee \neg x)))$
2033260910	$(\neg y \wedge \neg z) \veebar w \veebar ((v \veebar (\neg z \vee \neg x)) \vee (v \veebar (\neg w \vee (x \veebar y))))$
1972442478	$y \veebar \neg v \veebar ((y \wedge (\neg x \vee (v \veebar z))) \vee (z \veebar (\neg w \vee (v \wedge \neg x))))$
1966151022	$(w \wedge \neg y \wedge \neg v) \veebar \neg z \veebar ((x \wedge (w \veebar z)) \vee (v \veebar (\neg y \vee z)))$
1964324958	$((z \wedge \neg x) \vee (y \wedge (w \veebar v))) \veebar x \veebar (v \vee (w \wedge (\neg z \veebar x)))$
1789499990	$x \veebar (\neg z \vee (\neg x \wedge y)) \veebar v \veebar (\neg w \vee (x \veebar y))$
1788450214	$\neg v \veebar z \veebar (\neg x \vee \neg w) \veebar (y \vee (w \wedge (\neg z \vee \neg x)))$
1788258646	$y \veebar v \veebar ((x \veebar w) \vee ((w \veebar z) \wedge (x \vee y)))$
1785096554	$\neg v \veebar ((w \veebar (z \vee x)) \vee (w \veebar (x \vee \neg y)))$
1784060310	$w \veebar (\neg z \vee (\neg y \wedge \neg w)) \veebar \neg v \veebar (x \vee (\neg y \veebar (\neg w \vee \neg z)))$
1784046998	$(x \wedge \neg w) \veebar v \veebar ((y \vee w) \wedge ((\neg z \veebar x) \vee (\neg x \veebar y)))$
1771460006	$y \veebar \neg v \veebar ((\neg x \veebar w) \vee (\neg z \wedge (y \vee \neg w)))$
1722116713	$\neg v \veebar \neg x \veebar ((\neg w \vee z) \wedge (\neg y \vee (x \veebar w)))$
1721390438	$y \veebar \neg v \veebar ((\neg w \veebar z) \vee ((y \veebar z) \wedge (\neg x \vee w)))$
1721341545	$(z \vee \neg x) \veebar \neg w \veebar \neg y \veebar v$
1721326233	$\neg w \veebar \neg v \veebar (\neg y \vee (\neg z \wedge (x \veebar w)))$
1718199958	$w \veebar \neg v \veebar ((\neg z \veebar x) \vee (y \wedge (\neg x \vee \neg w)))$
1718183274	$\neg v \veebar (x \vee y) \veebar (w \vee (\neg z \wedge \neg x))$
1717937754	$(w \wedge (z \vee \neg x)) \veebar \neg v \veebar (y \vee (\neg z \veebar (x \vee w)))$
1717918038	$\neg z \veebar \neg v \veebar ((\neg w \vee \neg z) \wedge (x \vee (y \veebar w)))$
1717212566	$v \veebar \neg x \veebar ((\neg w \vee z) \wedge (\neg y \vee (\neg x \veebar \neg w \veebar z)))$
1716873897	$(z \vee (\neg x \wedge \neg w)) \veebar \neg v \veebar (\neg y \vee (\neg w \veebar (z \vee \neg x)))$
1705666138	$\neg v \veebar ((\neg y \vee \neg w) \wedge ((\neg z \veebar x) \vee (\neg x \veebar y)))$
1705421401	$v \veebar (y \vee (\neg x \wedge w)) \veebar (\neg z \vee (x \wedge y \wedge \neg w))$
1705404073	$\neg v \veebar ((\neg y \vee \neg w) \wedge (x \vee (\neg y \veebar \neg w \veebar z)))$
1704614297	$v \veebar (w \vee (z \wedge \neg x)) \veebar (\neg y \vee (\neg z \wedge (\neg x \vee \neg w)))$
1701488230	$\neg v \veebar ((\neg w \wedge (\neg z \vee x)) \vee ((z \veebar x) \wedge (y \veebar z)))$
1701472617	$w \veebar \neg v \veebar (x \vee (z \wedge (\neg y \veebar w)))$
1701402217	$(\neg x \wedge (\neg y \vee \neg w)) \veebar v \veebar (w \vee (y \wedge \neg z))$
1701210534	$(x \vee \neg y \vee w) \veebar v \veebar (\neg w \vee (\neg z \wedge (x \veebar y)))$

Table B.2: List of 120 rules, part 2

Rule Number	**Shortest Boolean Formula**
1701144918	$\neg x \veebar \neg v \veebar ((\neg x \vee \neg w) \wedge (z \vee (y \veebar w)))$
1701139882	$\neg v \veebar (y \vee z) \veebar (\neg x \vee w \vee \neg z)$
1700440678	$\neg v \veebar ((\neg w \vee (y \veebar z)) \wedge (\neg z \vee (\neg x \veebar y)))$
1700439654	$(w \wedge (\neg x \vee \neg y)) \veebar \neg v \veebar (\neg z \vee (x \veebar \neg y \veebar w))$
1700439446	$\neg x \veebar \neg v \veebar ((w \wedge \neg z) \vee (y \wedge (\neg x \vee \neg w)))$
1700370790	$z \veebar \neg v \veebar ((\neg y \veebar w) \vee (z \wedge (\neg x \veebar w)))$
1700161178	$v \veebar ((z \vee (\neg y \veebar w)) \wedge (w \vee (x \veebar y)))$
1521117801	$\neg z \veebar \neg v \veebar ((\neg y \vee (z \veebar x)) \wedge (z \vee (\neg x \veebar w)))$
1521112665	$(\neg z \wedge (x \vee \neg w)) \veebar \neg v \veebar (\neg y \vee (z \veebar x))$
1520854614	$(w \vee (x \wedge \neg y)) \veebar \neg v \veebar (\neg z \vee (x \veebar (\neg y \vee \neg w)))$
1520802217	$\neg v \veebar ((w \wedge \neg z) \vee (y \veebar (x \vee (y \wedge \neg z))))$
1520019113	$(\neg x \wedge y) \veebar \neg v \veebar ((\neg y \vee \neg z) \wedge (w \vee (\neg z \wedge x)))$
1516935845	$(\neg y \wedge (\neg z \vee x)) \veebar \neg v \veebar (\neg x \vee (w \wedge (y \veebar z)))$
1516874390	$(\neg z \wedge (x \vee \neg y)) \veebar \neg v \veebar x \veebar (y \vee w)$
1516660054	$(\neg x \wedge y \wedge z) \veebar v \veebar (w \vee (\neg y \veebar (z \vee \neg x)))$
1515623833	$(\neg z \wedge (\neg x \vee w)) \veebar v \veebar (x \vee (\neg y \wedge w))$
1515608406	$v \veebar (x \vee (\neg w \veebar (\neg y \vee (\neg w \wedge z))))$
1504275045	$y \veebar \neg x \veebar \neg v \veebar ((\neg z \veebar x) \vee (\neg y \veebar w))$
1504269670	$(\neg w \wedge (\neg y \vee z)) \veebar v \veebar (\neg z \vee (x \wedge (\neg y \veebar w)))$
1503291750	$\neg z \veebar (w \vee (z \wedge \neg x)) \veebar v \veebar (\neg y \vee (\neg x \veebar (\neg w \vee \neg z)))$
1500162405	$z \veebar \neg x \veebar \neg v \veebar ((\neg x \veebar y) \vee (\neg w \veebar z))$
1499830618	$(w \wedge (\neg z \vee \neg x)) \veebar v \veebar (y \vee (\neg x \veebar \neg w \veebar z))$
1499092377	$(z \wedge \neg x \wedge (\neg y \vee w)) \veebar v \veebar (y \vee \neg w \vee z)$
1498782054	$v \veebar ((y \veebar w) \vee (z \wedge (x \vee \neg y)))$
1454024357	$(y \wedge (x \vee w)) \veebar \neg v \veebar (z \vee (x \veebar y))$
1454004838	$\neg v \veebar ((\neg x \vee \neg y) \wedge (\neg w \vee (y \veebar z)))$
1453955734	$\neg z \veebar (w \vee (\neg z \veebar x)) \veebar \neg v \veebar (y \vee (\neg z \wedge (x \vee \neg w)))$
1452975461	$z \veebar v \veebar ((\neg y \wedge w) \vee (\neg x \wedge (\neg w \vee \neg z)))$
1452907109	$\neg y \veebar v \veebar (\neg x \vee w) \veebar (\neg z \vee (\neg w \wedge (\neg x \vee y)))$
1452693930	$(y \vee (\neg w \wedge z)) \veebar \neg v \veebar (x \vee (z \veebar (\neg y \vee \neg w)))$

Table B.3: List of 120 rules, part 3

Rule Number	**Shortest Boolean Formula**
1452693094	$v \veebar ((x \wedge y) \vee (\neg w \veebar (\neg z \vee \neg x)))$
1452692889	$\neg w \veebar \neg v \veebar ((\neg y \vee \neg w) \wedge (\neg z \vee x))$
1449809497	$(y \vee w \vee \neg z) \veebar \neg v \veebar (x \vee (\neg y \veebar w))$
1449567658	$\neg v \veebar ((\neg w \vee \neg z) \wedge (\neg y \vee (z \veebar x)))$
1448765097	$v \veebar ((z \vee (\neg y \veebar w)) \wedge ((\neg z \veebar x) \vee (y \wedge w)))$
1448761945	$v \veebar (x \vee ((\neg y \veebar z) \wedge (\neg w \veebar z)))$
1448715622	$(z \vee (\neg y \wedge w)) \veebar \neg v \veebar (x \vee (y \veebar (\neg w \vee \neg z)))$
1448450393	$w \veebar v \veebar ((\neg w \wedge (x \vee \neg y)) \vee (\neg z \wedge (\neg x \vee y)))$
1437227606	$y \veebar \neg v \veebar (z \vee (\neg y \veebar (x \vee (\neg y \wedge w))))$
1436984665	$(\neg z \wedge (x \vee \neg y)) \veebar v \veebar (y \vee (\neg x \wedge w))$
1435937174	$v \veebar (y \vee (x \wedge w)) \veebar ((x \vee w) \wedge (\neg y \vee \neg z))$
1435932310	$v \veebar ((x \veebar w) \vee (y \wedge z))$
1435932262	$(\neg w \wedge (\neg y \vee \neg z)) \veebar v \veebar (\neg x \vee (\neg y \veebar z))$
1435932054	$v \veebar ((x \veebar w) \vee (y \wedge (z \vee \neg x)))$
1435919973	$\neg z \veebar v \veebar ((\neg w \vee z) \wedge (y \vee (x \veebar w)))$
1435916650	$(x \vee \neg w \vee z) \veebar \neg v \veebar (y \vee (\neg w \veebar (\neg z \vee \neg x)))$
1435855462	$v \veebar ((\neg z \veebar (\neg x \vee \neg w)) \vee (\neg z \veebar (y \vee \neg w)))$
1433033321	$y \veebar x \veebar v \veebar ((y \vee \neg w) \wedge (\neg z \vee x))$
1432983898	$(\neg x \wedge \neg y) \veebar v \veebar ((\neg y \veebar z) \vee (\neg w \veebar z))$
1432786266	$\neg z \veebar v \veebar ((w \vee (\neg y \veebar z)) \wedge (z \vee (\neg x \veebar y)))$
1432708453	$v \veebar (\neg x \vee w \vee (y \wedge z))$
1431999833	$v \veebar ((y \wedge z) \vee (x \veebar y) \vee (\neg w \wedge \neg z))$
1431923050	$(x \vee y \vee \neg w) \veebar \neg v \veebar (z \vee (\neg w \veebar (x \vee \neg y)))$
1431737766	$\neg x \veebar \neg v \veebar ((\neg z \vee \neg x) \wedge (w \vee (x \veebar y)))$
1431726490	$(x \vee y \vee w) \veebar \neg v \veebar (z \vee (w \veebar (\neg x \vee \neg y)))$
1431725401	$v \veebar ((y \wedge w) \vee (x \veebar y) \vee (x \veebar \neg w \veebar z))$
1431673493	$v \veebar ((\neg x \wedge \neg y) \vee z \vee (x \wedge \neg w))$
1431672169	$v \veebar ((x \veebar (y \vee \neg w)) \vee z \vee (y \wedge \neg w))$
1297700276	$v \veebar ((z \veebar x \veebar (w \vee v)) \vee ((w \vee \neg z) \veebar (\neg y \vee (v \wedge \neg x))))$
861551811	$(\neg x \wedge (\neg y \vee \neg z)) \veebar w \veebar (y \vee (z \wedge (\neg w \veebar v)))$

Table B.4: List of 120 rules, part 4

Bibliography

[1] I. Goldberg and D. Wagner. Randomness and the Netscape Browser. *Dr. Dobb's Journal*, 21:66–106, January 1996.

[2] N. Minar. Breakable Session Keys in Kerberos v4. cypherpunks mailing list, 1996. message-ID: 199602200828.

[3] A. Biryukov, A. Shamir, and D. Wagner. Real Time Cryptanalysis of A5/1 on a PC. In *FSE: Fast Software Encryption*, pages 1–18. Springer-Verlag, 2000.

[4] P. Hellekalek. Good random number generators are (not so) easy to find. *Mathematics and Computers in Simulation*, 46(5-6):485–505, June 1998. ISSN 0378-4754.

[5] M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, January 1998.

[6] A. L. Rukhin, National Institute of Standards, and Technology U.S. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. U.S. Dept. of Commerce, Technology Administration, National Institute of Standards and Technology, rev. edition, 2001.

[7] G. Marsaglia. The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness. Web site at the Department of Statistics, Florida State University, Tallahassee, FL, USA., 1995.

[8] J. Walker. ENT: A Pseudorandom Number Sequence Test Program. `http://www.fourmilab.ch/random/`, 2008.

[9] S. Wolfram. Random Sequence Generation by Cellular Automata. *Advances in Applied Mathematics*, 7(2):123–169, June 1986.

[10] A. Ilachinski. *Cellular Automata: A Discrete Universe*. World Scientific, 2001.

[11] M. Serra, D. M. Miller, and J. C. Muzio. Linear Cellular Automata and LFSRs are Isomorphic. In *Proceedings of Third Technical Workshop on New Directions for IC Testing*, pages 213–223, 1988.

[12] P. D. Hortensius. *Parallel Computation of Non-Deterministic Algorithms in VLSI*. PhD thesis, University of Manibota, Winnipeg, Canada, 1987.

[13] P. D. Hortensius, R. D. McLeod, and H. C. Card. Parallel Random Number Generation for VLSI Systems Using Cellular Automata. *IEEE Transactions on Computers*, 38(10):1466–1473, 1989.

[14] M. Tomassini, M. Sipper, and M. Perrenoud. On the Generation of High-Quality Random Numbers by Two-Dimensional Cellular Automata. *IEEE Transactions on Computers*, 49(10):1146–1151, October 2000.

[15] M. Tomassini and M. Perrenoud. Cryptography with Cellular Automata. *Appl. Soft Comput.*, 1(2):151–160, 2001.

[16] M. Sipper and M. Tomassini. Generating Parallel Random Number Generators By Cellular Programming. *International Journal of Modern Physics C*, 7:181–190, 1996.

[17] M. Sipper and M. Tomassini. Co-evolving Parallel Random Number Generators. volume 1141 of *Lecture Notes in Computer Science*, pages 950–959. Springer, 1996.

[18] E. Borel. *Leqons sur la Théorie des Fonctions*. Gauthier-Villars, Paris, 1914.

[19] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, volume 17 of *Algorithms and Combinatorics*. Springer, 1998.

[20] A. Menezes, P. C. van Oorschot, and S A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[21] M. Haahr. True Random Number Service. `www.random.org`, 1998.

[22] J. Walker. HotBits: Genuine Random Numbers, Generated by Radioactive Decay. `https://www.fourmilab.ch/hotbits/`, 1996.

[23] Ü. Güler and S. Ergün. A high speed, fully digital IC random number generator. *AEU - International Journal of Electronics and Communications*, 66(2):143 – 149, 2012.

[24] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM J. Comput.*, 13(4):850–864, 1984.

[25] M. Geisler, M. Krøigård, and A. Danielsen. About Random Bits. 2004.

[26] A. C. Yao. Theory and Application of Trapdoor Functions. In *Foundations of Computer Science, 1982. SFCS '08. 23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, November 1982.

[27] Ç. K. Koç. *Cryptographic Engineering*. Springer Publishing Company, Incorporated, 2008. ISBN 0387718168, 9780387718163.

[28] S. W. Golomb. *Shift Register Sequences*. Aegean Park Press, Laguna Hills, CA, USA, 1981. ISBN 0894120484.

[29] D. E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms*. Addison-Wesley, 1969.

[30] U. M. Maurer. A universal statistical test for random bit generators. *Journal of Cryptology*, 5(2):89–105, 1992.

[31] H. V. McIntosh. What has and What hasn't Been Done with Cellular Automata. 1990.

[32] W. A. Beyer, P. H. Sellers, and M. S. Waterman. Stanislaw M. Ulam's Contributions to Theoretical Theory. *Letters in Mathematical Physics*, (10):231–242, 1985.

[33] Conway's Game of Life. `http://www.kongregate.com/games/shaman4d/conways-game-of-life`.

[34] S. Wolfram. *A New Kind of Science*. Wolfram Media, January 2002.

[35] S. Wolfram. Statistical Mechanics of Cellular Automata. *Reviews of Modern Physics*, 55(3):601–644, 1983.

[36] 3D Moore Neighbourhood. `http://cell-auto.com/neighbourhood/`.

[37] Two-Dimensional Cellular Automata. `http://radicalart.info/AlgorithmicArt/grid/cellular/2D/`.

[38] M. Serra, T. Slater, J. C. Muzio, and D. M. Miller. The Analysis of One-Dimensional Linear Cellular Automata and Their Aliasing Properties. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on CAD*, 9(7):767–778, July 1990.

[39] S. Uei Guan and S. K. Tan. Pseudorandom Number Generation with Self-Programmable Cellular Automata. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 23(7):1095–1101, 2004.

[40] S. Wolfram. Cryptography with Cellular Automata. In *Lecture Notes in Computer Sciences on Advances in Cryptology—CRYPTO 85*, volume 218, pages 429–432. Springer-Verlag New York, Inc., 1986.

[41] W. Meier and O. Staffelbach. Analysis of Pseudo Random Sequence Generated by Cellular Automata. In *EUROCRYPT*, pages 186–199, 1991.

[42] Ç. K. Koç and A. M. Apohan. Inversion of Cellular Automata Iterations, 1997.

[43] B. Martin. A Walsh Exploration of Elementary CA Rules. *Journal of Cellular Automata*, 3(2):145–156, 2008.

[44] B. Martin and P. Solé. Pseudo-random Sequences Generated by Cellular Automata. *CoRR*, abs/0807.3865, 2008.

[45] P.D. Hortensius, R.D. McLeod, and H.C. Card. Parallel Random Number Generation for VLSI Systems Using Cellular Automata. *Computers, IEEE Transactions on*, 38(10):1466–1473, October 1989.

[46] S. H. Shin and K. Y. Yoo. Analysis of 2-State, 3-Neighborhood Cellular Automata Rules for Cryptographic Pseudorandom Number Generation. pages 399–404. IEEE Computer Society, 2009.

[47] S. Nandi, B. K. Kar, and P. P. Chaudhuri. Theory and Applications of Cellular Automata in Cryptography. *IEEE Trans. Comput.*, 43(12):1346–1357, December 1994.

[48] B. Murphy, S. R. Blackburn, and S. Murphy. Comments on "Theory and Applications of Cellular Automata in Cryptography", 1997.

[49] M. J. Mihaljev'c. Security Examination of a Cellular Automata Based Pseudorandom Bit Generator Using an Algebraic Replica Approach. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 1255 of *Lecture Notes in Computer Science*, pages 250–262. Springer Berlin Heidelberg, 1997.

[50] N. H. Packard. Adaptation toward the edge of chaos. In *Dynamic Patterns in Complex Systems*, pages 293–301. World Scientific, 1988.

[51] M. Sipper and M. Tomassini. Generating Parallel Random Number Generators By Cellular Programming. *International Journal of Modern Physics C*, 7:181–190, 1996.

[52] M. Tomassini, M. Sipper, M. Zolla, and M. Perrenoud. Generating High-quality Random Numbers in Parallel by Cellular Automata. *Future Gener. Comput. Syst.*, 16(2-3):291–305, December 1999. ISSN 0167-739X.

[53] T. E. Tkacik. A Hardware Random Number Generator. In *CHES*, pages 450–453, 2002.

[54] C. G. Langton. Computation at the Edge of Chaos: Phase Transitions and Emergent Computation. *Phys. D*, 42(1-3):12–37, June 1990. ISSN 0167-2789.

[55] C. K. Yuen. Testing Random Number Generators by Walsh Transform. *IEEE Transactions on Computers*, 26(4):329–333, 1977. ISSN 0018-9340.

[56] Boolean Oracle. `http://boolean-oracle.swtch.com/`.