Pseudorandom Number Generator by Cellular Automata And Its Application to Cryptography

by

Siu Chi Sang Obadiah

Thesis
Submitted to the Faculty of the Graduate School of
The Chinese University of Hong Kong
(Division of Mathematics)

In partial fulfillment of the requirements
for the Degree of
Master of Philosophy

July, 1999

撮要

在自然科學的研究上已經廣泛使用 Cellular Automata。在一九八六年，Stephen Wolfram 發現可以將 Cellular Automata 應用在產生假任意數，而其中一項的數學軟件 Mathematica 也正使用它。

在七十年代，因著公開鍵密碼系統的創立，密碼學得以大力發展。在發展過程中，發現那些使用定函數的公開鍵密碼系統，因著已編碼的訊息內帶著原文或密碼鍵的資訊而不安全，所以我們想利用假任意數產生器去建立一個概率密碼系統，使那在已編碼的訊息內帶著原文或密碼鍵的資訊因這個系統而被隱藏。

在這篇論文中，我們會討論一個利用 Cellular Automata 制作的假任意數產生器。首先，我們會討論假任意數產生器的三個特性——擴張性、統計上不可分及計算複雜性。以後我們會介紹 Cellular Automata 的一些基本性質，同研討它的一些工具。接著，我們會看看 Cellular Automata 基本律 30 如何適用於制作假任意數產生器；因著我們限定產生器的輸出在{0,1}這個集內，因此我們會介紹一個如何利用在{0,1}內的假任意數去制作更大整數集上的假任意數。最後，我們會討論幾個利用假任意整數去加強保安的密碼系統。

Abstract

Cellular Automata have been widely used in research of nature science. In 1986, Stephen Wolfram found that Cellular Automata can be used in generating pseudorandom number and now is used by Mathematica to generate random integer.

Cryptography has been rapidly developed after the concept of public key cryptosystem was established in 70's. During the development, it was found that any public key cryptosystem which use deterministic function may have a risk that some partial information of the plaintext or the key could be found by the ciphertext, therefore we want to develop a probabilistic cryptosystem using pseudorandom number generator to hide this partial information.

In this thesis, we discuss a special kind of pesudorandom number generator which makes use of Cellular Automata. We begin by discuss in the pseudorandomness of the generator by three properties - expansiveness, statistical indistingushiblity and computational complexity. Then we introduce some basic properties of Cellular Automata, and some tools to help us to expound the detail about Cellular Automata. After that, we discuss a special kind of Cellular Automata - Elementary Automata Rule 30 and see how it suits to be a pesudorandom number generator. Since we restrict the set of output of the Cellular Automata on $\{0, 1\}$ we will introduce an algorithm to use the pseudorandom bits to generate pseudorandom integer. Finally, we discuss serveral algorithms in cryptography which use a pseudorandom integer to secure the cryptosystem.

ACKNOWLEDGMENTS

I wish to express my gratitude to my supervisor, Prof. K. M. Yeung, for his guidance and encouragement in these two years. His supervision has helped a lot during the preparation of this thesis. I would like to express my gratitude to the Department of Mathematics and the Science Faculty for their support.

Contents

Chapter 1

Pseudorandom Number Generator

1.1 Introduction

Random numbers have been widely used in numerical analysis, testing chips for defects and decision making.

To construct "random" number, it is equivalent to obtain independent samples from uniform probability distribution on sample space. Obtaining "random" number can be reduced to a problem that to obtain *random bits* from $\{0,1\}$ [2]. Therefore, in this thesis, we will mainly discuss the case that the sample space is $\{0,1\}$, and will discuss in the Section 4.2 when the sample space is the set of integers.

To discuss the "random" behaviour, we will mainly concern three properties of the generator: Is the generator expansive in the sense of a dynamical system? Is the output evenly distributed in the sample space if the input is? Is the computational complexity like that of a one-way function? We will discuss these three properties one by one below.

Expansiveness is a concept arised from dynamic system. A chaotic dynamical system has three properties, and one is expansiveness. We give the definition of expansiveness.

Definition A flow f_t on a metric space M is expansive at a point $x \in M$ if $|f_t(x) - f_t(y)| \geq \lambda^t |x - y|$ where $\lambda > 1$, provided that $|x - y| < \delta$ and $0 \leq t < t_0$ for some fixed t_0.

Expansiveness means that a small change of seed will result in a large change of the result.

For the distribution of output numbers, we need to introduce a concept called "statistical indistingushibility". We discuss it in the next section.

For the complexity of the generator, we need the concepts of NP-problem and one-way function in complexity theorem of computing.

The class P is the set of decision problems that the solutions can be solved in polynomial time, and the class NP is the set of decision problems that the solutions can be verified in polynomial time. The class NP-complete is the set of decision problems that they are NP and for all problems in NP, there exist a polynomial time reduction to the problems in NP-complete.

Since there are widely believe that P$\neq$NP, so a NP-complete problem must not in P, otherwise all NP problems will be in P.

Now, let us consider the following problem:

For a generator G, is there a x such that $G(x) = y$ for a given y?

That problem must be in NP-complete for generator G, since we do not want that problem can be solved in polynomial-time, otherwise the seed of the generator will be easily to be found from the random bits. In the other hand, we want that problem can be verify in polynomial-time since we want to generate random bits from any seed in polynomial-time, that means the generator is efficient.

1.2 Statistical Indistingushible and Entropy

A definition of pseudorandom number is always relative to the use to which pesudorandom numbers are to be put[3]. Our aim is to simulate a target probability distribution to within a degree of approximation.

We measure the degree of approximation using statistical tests.

Let P be the source probability.

$G(P)$ be the probability distribution generated by the PRG G,

Q be the target probability distribution approximated by G.

A statistic is any deterministic function $\sigma(x)$ of a sample x drawn from a distribution, and a statistical test σ consists of the computation of a statistic σ of sample drawn from $G(P)$.

Definition We say that distribution P_1 and P_2 on the sample space $\mathcal{S}$ are ϵ-indistinguishable using the statistic σ on $\mathcal{S}$ provided that the expected values of $\sigma(x)$ drawn from P_1 to P_2, respectively, agree to within the tolerance level ϵ; i.e.

$$|E[\sigma(x) : x \in P_1] - E[\sigma(x) : x \in P_2]| \leq \epsilon$$

Given a collection $\mathcal{T}$=$\{(\sigma_i, \epsilon_i)\}_i$ of statistical tests σ_i with corresponding tolerance levels ϵ_i we say that G is a $\mathcal{T}$-pseudorandom number generator from source

P to target Q provided that $G(P)$ is ϵ_i indistinguishable from Q for all statistical tests σ_i drawn from $\mathcal{T}$.

In most uses, we want $P = U_k$ (uniform distribution) on the set $\{0,1\}^k$ of binary strings of length k and for the target probability $Q = U_l$ for some l.

In this thesis, we will test the generator by some statistical tests from [21], and this will be discussed in Chapter 3.

In addition, the amount of randomness in a probability distribution that can be measured by its binary entropy or information, which for a discrete probability distribution P is

$$H(P) = -\sum_x p(x)\ \log_2 p(x)$$

where x runs over the atoms of P and p is the probability function. In particular

$$H(U_k) = k$$

The notion of randomness-increasing initially seems impossible, because any deterministic mapping G applied to a discrete probability distribution P never increase entropy, i.e.

$$H(G(P)) \leq H(P)$$

However, when computing power is limited, $G(P)$ may approximate a target distribution Q having a much higher entropy so well that, within the limits of computing power avaliable, one cannot tell the distribution $G(P)$ and Q apart, to a small tolerance level ϵ. If $H(Q)$ is much larger than $H(P)$, then we say G is indeed computationally randomness-increasing, as measured by the tolerance level to statistical tests $\mathcal{T}$=$\{(\sigma_i, \epsilon_i)\}_i$.

1.3 Example of PNG

Before discussing the cellular automata PNG, we will introduce some PNG to see the relationship between PNG and some cryptosystem.

Example (Multiplicative Congruential Generator)

$$x_{n+1} \equiv ax_n + b(modM) \text{ where } 0 \leq x_n \leq M-1$$

Here (a, b, n) are the parameters describing the generator and x_0 is the seed.

It is well known that this type of generator is similar to a type of private key cryptosystem called "Caesar Cipher".

Example (Power Generator)

$$x_{n+1} \equiv x_n^d(modN)$$

Here (d, N) are parameters describing the generator and x_0 is the seed.

If $N = p_1p_2$ is a product of two distinct primes and $(d, \phi(N)) = 1$, where ϕ is the Euler's totient function, defined by

$$\phi(N) = \#(\tfrac{\mathbb{Z}}{N\mathbb{Z}})^* = (p_1 - 1)(p_2 - 1)$$

Then the map $x \rightarrow x^d(modN)$ is one-to-one on $(\frac{\mathbb{Z}}{N\mathbb{Z}})^*$ and this operation is the encryption operation of the RSA public key cryptosystem. We call this kind of generators RSA generators.

For $d = 2$ and $N = p_1p_2$ with $p_1 \equiv p_2 \equiv 3(mod4)$; we call this kind of generators Square Generator. In this case the mapping

$$x_{n+1} \equiv (x_n)^2(modN)$$

is four-to-one on $(\frac{\mathbb{Z}}{N\mathbb{Z}})^*$.

One says that y is a quadratic residue (mod N) for some x, if $y = x^2$ (mod N) for some x. Now, any quadratic residue y with $(y, N) = 1$ has exactly four square roots. The assumption $p_1 \equiv p_2 \equiv 3$ (mod 4) guarantees that -1 is a quadratic nonresidue (mod p_1) and (mod p_2), and this fact implies that exactly one of these four square roots is itself a quadratic residue. We denote it by y and call it the principal square root of y.

If we restrict the generator to the domain

$$Q(N) = \{y \text{ (mod } N\text{)}:(y, N)=1 \text{ and } y \text{ is a quadratic residue. }\}$$

then it becomes a one-to-one mapping. The square generator on the domain $Q(N)$ is a pesudorandom number generator.

Example (Discrete Exponential Generator)

$$x_{n+1} \equiv g^{x_n}(modN)$$

Here (g, N) are parameters describing the generator and x_0 is the seed.

When N is a odd prime p and g is a primitive root (mod p). Then the problem of recoving x_n given (x_{n+1}, g, N) is the discrete logarithm problem, and is clearly a hard number-theoretic problem.

The discrete exponentiation operation (mod p) was suggested for cryptographic use in the key-exchange scheme. A key exchange scheme is a scheme for two parties to agree on a secret key used in an insecure channel.

Example (Kneading Map) Consider a bivariate transformation

$$(x_{n+1}, y_{n+1}) := (y_n, x_n + f(y_n, z_n))$$

where f is a fixed bivariate function, usually taken to be nonlinear. The function $f(\cdot, \cdot)$ determines the generator, while (x_0, y_0) and the family $\{z_n\}$ are the seed. One often takes all $z_n := K$ for a fixed K.

One can generalize this generator to take $\mathbf{x}$,$\mathbf{y}$,$\mathbf{f}(\cdot,\cdot)$ to be vector-valued. The Data Encryption Standard (DES) cryptosystem is composed of sixteen iteration of vector-valued maps of this type, where $f(x_0, y_0)$ is the plaintext, all $z_i = K$ compose the key, and f is a specific nonlinear function representable as a polynomial in several variables.

Example (Shift-register Sequences)

$$x_{n+1} := f(x_n, x_{n-1}, ..., x_{n-j})$$

for a fixed function f.

Such sequences are easy to compute by storing at each iteration the vector $(x_n,\ x_{n-1},\ ...\ ,x_{n-j})$ and using it to compute $(x_{n+1}, x_n, ..., x_{n-j+1})$. The seed is $(x_0, x_{-1}, ..., x_{-j})$

This generator is not much related to any cryptosystem but is a good generator that generates pseudorandom number in a efficient way.

Chapter 2

Basic Knowledge of Cellular Automata

2.1 Introduction

Cellular Automata are mathematical idealizations of physical systems in which space and time are discrete, and physical quantities take on a finite set of discrete values. They were originally introduced as a possible idealization of biological systems with the particular purpose of modeling biological self-reproduction. They have been applied and reintroduced for a variety purpose.

Physical systems containing many discrete elements with local interactions are often conveniently modeled as cellular automata. Any physical system satisfying differential equations may be approximated as a cellular automata by introducing finite differences and discrete variables.

Cellular Automata have also been used to study problems in number theory and their applications to tapestry design. In a typical case, successive differences in a sequence of numbers reduced with a small modulus are taken, and the geometry of zero regions is investigated.

Below, we will give a definition of Cellular Automata from [6]. That is not the general form of Cellular Automata but after this section, we will see a simply one.

Definition Let I be the set of integers. To obtain a cellular space we associate with the set $I \times I$:

1. The neighborhood function $g : I \times I \to 2^{I \times I}$, defined by

$$g(\alpha) = \{\alpha + \delta_1, \alpha + \delta_2, ...\alpha + \delta_n\} \quad \forall \alpha \in I \times I$$
$$\text{where } \delta_i(i = 1, 2, ..., n) \in I \times I \text{ is fixed.}$$

2. The finite automaton (V, v_0, f), where V is the set of *cellular states*, v_0 is a distinguished element of V called the *quiescent state*, and f is the local transition function from n-tuples of elements of V into V. The function f is subject to the restriction,

$$f(v_0, v_0, ..., v_0) = v_0.$$

We may think that cellular space is a space as a plane assemblage of a countable numbers of interconnected cells (or sites). The location of each cell is located by its Cartesian coordinates. Each cell contains an identical copy of the finite automaton (V, v_0, f) and the states $v^t(\alpha)$ of a cell α at time t is precisely the state of its associated automaton at time t. Each cell is connected to the n neighboring cells $\alpha + \delta_1, \alpha + \delta_2, ...\alpha + \delta_n$. In all that follows we shall assume the α is its neighbor itself, and in this assumption, $\delta_1 = 0$.

The neighborhood state function $h^t : I \times I \to V^n$ is defined by

$$h^t(\alpha) = (v^t(\alpha), v^t(\alpha + \delta_2), ..., v^t(\alpha + \delta_n)).$$

Now, we can relate the neighborhood states of a cell α at time t to the cellular state of that cell at time $t + 1$ by

$$f(h^t(\alpha)) = v^{t+1}(\alpha).$$

The above sees that two main restriction need to define Cellular Automata, the neighborhood function (tell you where is the neighborhood) and the state function (tell you how the state change with time).

2.2 Elementary and Totalistic Cellular Automata

$a_i^{(t)}$ is taken to denote the value of site i in one-dimensional cellular automaton at time step t. Each site value is specified as an integer in the range 0 through k-1. The site value evolve by iteration of the mapping

$$a_i^{(t)} = \mathbf{F}[a_{i-r}^{(t-1)}, a_{i-r+1}^{(t-1)}, \dots, a_i^{(t-1)}, \dots, a_{i+r}^{(t-1)}]. \quad (2.1)$$

F is an arbitrary function which specifies the cellular rules.

The parameter r in the equation above determines the range of the rule; the value of a given site depends on the last values of a neighborhood of at most $2r+1$ sites. The region affected by a given site grows by at most r sites in each direction at every time step; propagating features generated in cellular automaton evolution may therefore travel at most r sites per time step. After t time step, a region of at most $1+2rt$ site may therefore be affected by a given initial site value.

The elementary cellular automata have $k = 2$ and $r = 1$, corresponding to nearest-neighbor interactions.

An alternative form of eq. (2.1) is

$$a_i^{(t)} = \mathbf{f}[\sum_{j=-r}^{j=r} \alpha_j a_{i+j}^{(t-1)}] \quad (2,2)$$

where α_j are integral constants, and the function **f** takes a single integer argument. Rules specified according (2.1) may be reproduced directly by taking $\alpha_j = k^{r-j}$.

Totalistic rules are obtained by taking $\alpha_j = 1$. Such rules give equal weight to all sites in a neighborhood, and imply that the value of a site depends only on a total of all preceding neighborhood site values.

Cellular automaton rules may be combined by composition. The set of cellular rules is closed under composition, although composition increases the numbers of sites in the neighborhood. Composition of a rule with itself yields patterns corresponding to alternate time steps in time evolution according to the rule. If the composition of F_1F_2 of rules generates a sequence of configuration with period π, then the rule F_2F_1 must also allow a sequence of configuration with period π.

The form of the function F in time evolution rule (2.1) may be specified by a "rule number" [7]

$$\mathbf{R_F} = \sum_{\{a_{i-r}, a_{i+r}\}} \mathbf{F}[a_{i-r}, \ldots, a_{i+r}] k^{\sum_{j=-r}^{j=r} k^{r-j} a_{i+j}}$$

The function $\mathbf{f}$ in equation (2.2) also can be specified by a numerical "code"

$$\mathbf{C_f} = \sum_{n=0}^{(2r+1)(k-1)} k^n \mathbf{f}[n]$$

In general, there are a total of $k^{(k^{(2r+1)})}$ possible cellular rules of form (2.1) or (2.2).

A few cellular automaton rules are "reducible" in the sense that the evolution of sites with particular values, or on a particular grid of positions and times, are independent of other sites values.

Very little information on the behaviour of a cellular automaton can be deduced directly from simple properties of its rule. A few simple results are nevertheless clear.

For example, necessary conditions for a rule to yield unbounded growth are

$$\mathbf{F}[a_{i-r}, a_{i-r+1}, \ldots, a_i, 0, 0, \ldots, 0] \neq 0.$$
$$\mathbf{F}[0, \ldots, 0, 0, a_{i+1}, \ldots, a_{i+r}] \neq 0.$$

for some set of a_i.

If these conditions are not fulfilled then regions containing nonzero site surround by zero sites can never grow, and the cellular automaton must exhibit behaviour such that the pattern becomes homogeneous or degenerates into simple periodic structure (which are called class 1 and class 2 of cellular automata and are discussed in next section). For totalistic rules, the conditions becomes

$$\mathbf{f}[n] \neq 0$$

for some $n < r$.

One may consider cellular automata both finite and infinite in extent.

When finite cellular automata are discussed below, they are taken to consist of N sites arranged around a circle (periodic boundary conditions). Such cellular automata have a finite number k^N of possible states. Their evolution may be represented by a finite states transition diagram [8], in which nodes representing each possible configuration are joined by directed arcs, with single arc leading from a particular node to its successor after evolution for one time step. After a sufficiently long time (less than k^N), any finite cellular automaton must enter a cycle, in which a sequence of configuration is visited repeatedly. These cycles represented attractors for the cellular automaton evolution, and correspond to cycles in the state transition graph. At nodes in the cycle may be rooted in the tree have a single successor, but may have serveal predecessors. In the course of time evolution, all states corresponding to nodes in the tree to the cycles on which the root lies. Configurations corresponding to nodes on the periphery of the state diagram (terminals or leaves of the transient trees) are never reached in the evolution; they may occur only as initial states. The fraction of configuration which may be reached one time step in cellular automaton evolution, and which are therefore not on the periphery of the state transition diagram, give a simple measure of irreversibility.

The configuration of infinite automata are specified by infinite sequences of site values. Such sequences are identified as elements of a Cantor Set and discuss in [9].

Equation (2.1) and (2.2) may be generalized to several dimensions. For $r = 1$, there are at least two possible symmetric forms of neighborhood, containing 2d+1 (type I) and 3^d (type II) sites respectively; for larger r other "unit cells" are possible.

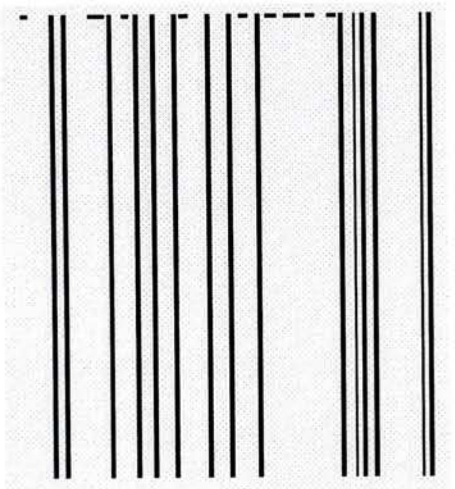
2.3 Four classes of Cellular Automata

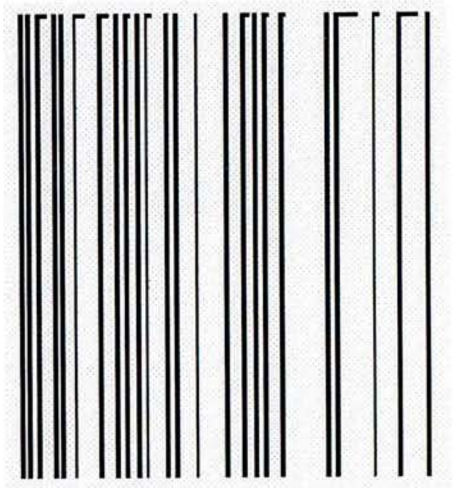
This section discusses some qualitative features of cellular automaton evolution.

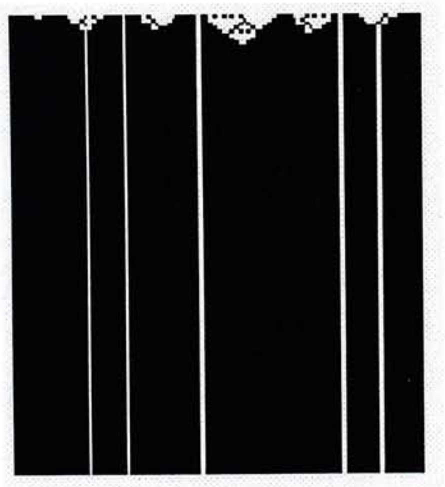
Despite the simplicity of their construction, cellular automata are founded to be capable of diverse and complex behaviour. To discuss its complexity, we will now see what happen when cellular automata evolute.

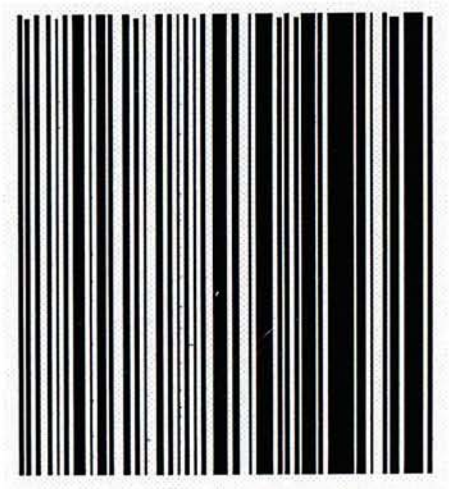
Now we will see the pattern of the cellular automata with initial condition that is randomly generated 100 site long. Most picture are elementary Cellular Automata Rule with $r = 1$ and $k = 2$, but there is no Class 4 in that kind of Cellular Automata, so we will find example in Totalistic Cellular Automata with $r = 2$ and $k = 2$.

The Elementary CA Rule 4,12, 218 and 205 are belonged to class 1, which

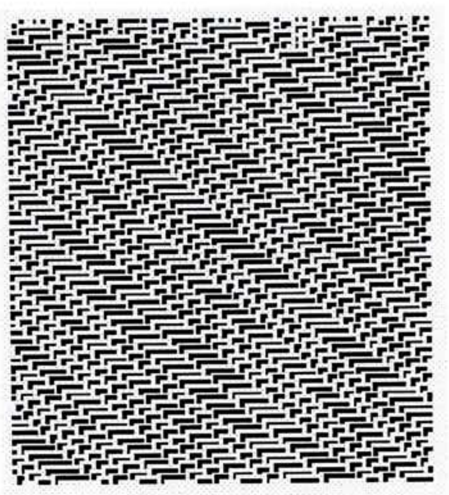

Rule 4

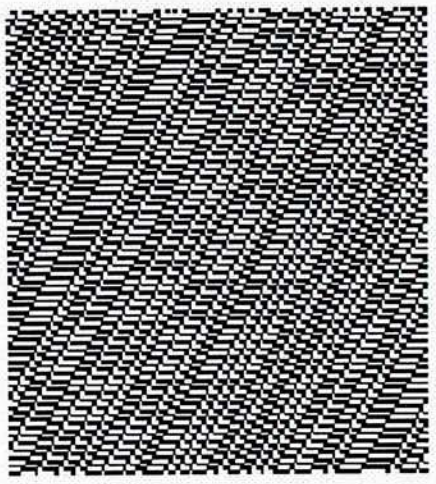

Rule 12


Rule 218

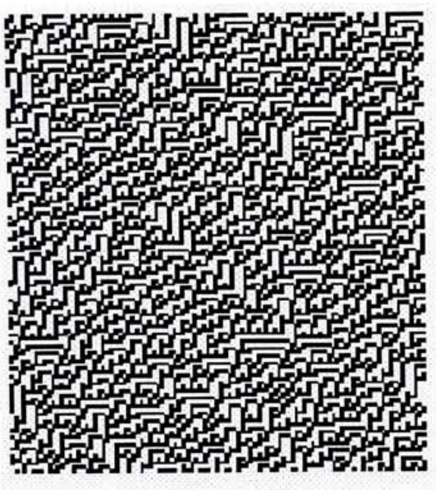

Rule 205

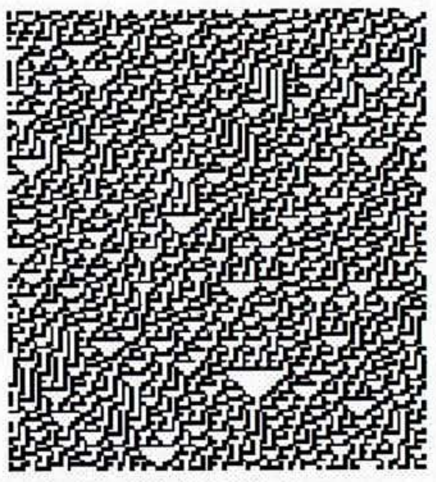
the pattern becomes homogeneous (fixed points).


Rule 9


Rule 83

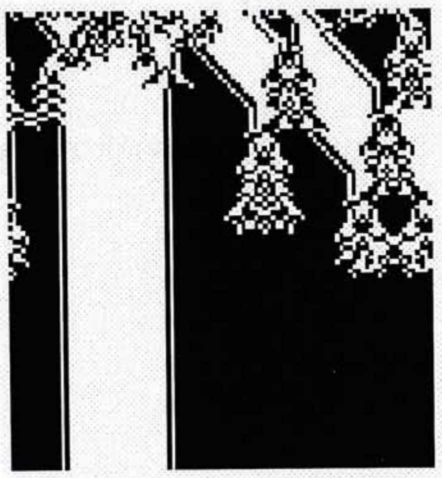
The Elementary CA Rule 9 and 83 are belonged to class 2, the pattern degenerates into simple periodic structure (limit cycles).

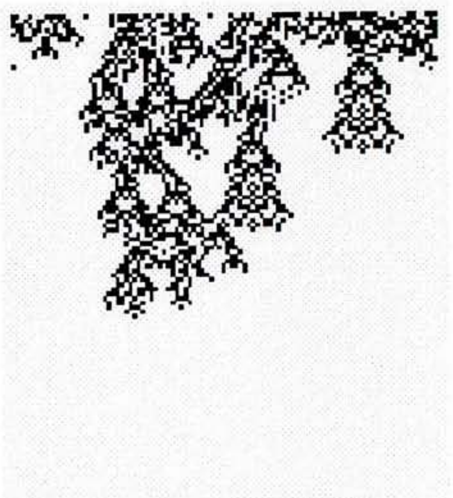

Rule 45


Rule 30

The Elementary CA Rule 45 and 30 are belonged to class 3, the pattern is aperiodic, and appears chaotic. Some patterns assigned to class 3 contain many triangular "clearing" and appear more regular than others. The regularity is related to the degree of irreversibility of the rules.

Most CA PNG are made by this class. The Rule 30 is discussed in next chapter.


Totalistic CA Rule 52


Totalistic CA Rule 20

The Totalistic CA Rule 52 and 20 are belonged to class 4, which complicated localized structures are produced by them.

According to [10], there is a relation between the class and the entropy. As the entropies study in that thesis is important to the analysis of CA Rule 30 PNG, it will be introduce in next section.

2.4 Entropy

This section describes quantitative statical measures of order and chaos in pattern generated by cellular automaton evolution. These measures may be used to distinguish the four classes of behaviour identified qualitatively above.

First, consider the statical properties of configurations generated at a particular time step in cellular automaton evolution. A disordered initial states, in which each site takes on its k possible values with equal independent probabilities, is statically random. Irreversible cellular automaton evolution generates deviations from statistical randomness. In a random sequence, all k^X possible subsequences of length X must occur with equal probabilities. With probabilities $p_i^{(x)}$ for the k^X possible sequence of site value in a length X block, one may define:

Definition (spatial set entropy)

$$s^{(x)}(X) = \frac{1}{X}\,\log_k(\sum_{j=1}^{k^X}\theta(p_j^{(x)})) \text{ where } \theta(p) = 1 \text{ for } p > 0 \text{ and } \theta(p) = 0 \text{ for } p = 0.$$

and

Definition (spatial measure entropy)

$$s_\mu^{(x)}(X) = -\frac{1}{X}\sum_{j=1}^{k^X} p_j^{(x)}\log_k p_j^{(x)}.$$

In both cases, the superscript (X) indicates the "spatial" sequences (obtained at a particular time step) are considered. The "set entropy" is determined directly by the total number $N^{(X)}(X)$ of length X blocks generated (with any nonzero probability) in cellular automaton evolution, according to

$$s^{(x)}(X) = \frac{1}{X}\,\log_k N^{(X)}(X).$$

In the "measure entropy" each block is weighted with its probability, so that the result depends explicitly on the probability measure for different cellular automaton configuration, as indicated by the subscript μ. Set entropy is often called "topological entropy"; measure entropy is sometimes referred as "metric entropy".

The definitions above yield immediately

$$s_\mu^{(x)}(X) \leq s^{(x)}(X) \leq 1.$$

The first equality holds only for "equidistributed" systems, in which all nonzero block probabilities $p_i^{(x)}$ are equal. The second equality holds if all possible length X blocks of site occurs, but perhaps with unequal probabilities. $s_\mu^{(x)}(X) = 1$ only for "X-random" sequences, in which all k^X possible sequences of X site values occur with equal probabilities. In addition,

$$0 \leq s_\mu^{(x)}(X) \leq s^{(x)}(X)$$

$s_\mu^{(x)}(X) = 0$ if and only if just one length X block occurs with nonzero probability, so $s^{(x)}(X) = 0$ also. As discuss above, the equality holds for class 1 cellular automata.

The entropies $s^{(x)}$ and $s_\mu^{(x)}$ may be obtained either for many blocks in a single cellular automaton configuration, or for blocks in an ensemble of different configuration. For smooth probability measures on the ensemble of possible initial configurations, the results obtained in these two ways are almost always the same. (A probability measure will be considered "smooth" if changes in the values of a few site in an infinite configuration lead only to very little change in the probability for the configuration) The set entropy $s^{(x)}$ is typically independent of the probability measure on the ensemble, for any smooth measure. The measure entropy $s_\mu^{(x)}$ in general depends on the probability measure for initial configuration, although for class 3 cellular automata, it is typically the same for at least a large class of smooth measures.

The entropies $s^{(x)}$ and $s_\mu^{(x)}$ are defined for infinite cellular automata. A corresponding definition may be given for finite cellular automata, with a maximum block length given by the total number of sites N in the cellular automaton. The entropies $s^{(x)}(N)$ and $s_\mu^{(x)}(N)$ are related to the global properties of the state transition diagram for the finite cellular automaton. The value of $s^{(x)}(N)$ at a particular time is determined by the fraction of possible configurations which may be reached at that time by evolution from any initial configuration. The limiting value of $s^{(x)}(N)$ at large times is determined by the fraction of configuration on cycles in the state transition graph.

The spatial sequences entropies were defined in terms of the sequences of sites values in a cellular automaton configuration at a particular time step. One may also define temporal entropies which characterize the sequences of values taken on by particular site through many time step of cellular automaton evolution. With probabilities $p_i^{(t)}$ for the k^T possible sequences of value for a site at T successive time steps, one may define

Definition (temporal set entropy)

$$s^{(t)}(T) = \tfrac{1}{T}\log_k(\sum_{j=1}^{k^T}\theta(p_j^{(t)})) \text{ where } \theta(p) = 1 \text{ for } p > 0 \text{ and } \theta(p) = 0 \text{ for } p = 0,$$

and

Definition (temporal measure entropy)

$$s_\mu^{(t)}(T) = -\tfrac{1}{T}\sum_{j=1}^{k^T} p_j^{(t)}\log_k p_j^{(t)}.$$

These entropies satisfy relations directly analogous to those defined on spatial sequences.

As a generalization of the spatial and temporal entropies introduced above, one may consider entropies associated with space-time patches in the patterns generated by cellular automaton evolution. With probabilities $p_i^{(t,x)}$ for k^{XT} possible patches of spatial width X and temporal extent T, one may define

Definition (set entropy)

$$s^{(t,x)}(T:X) = \tfrac{1}{T}\log_k\Big(\sum_{j=1}^{k^{XT}} \theta(p_j^{(t,x)})\Big) \text{ where } \theta(p)=1 \text{ for } p>0 \text{ and } \theta(p)=0 \text{ for } p=0,$$

and

Definition (measure entropy)

$$s_\mu^{(t,x)}(T:X) = -\tfrac{1}{T}\sum_{j=1}^{k^{XT}} p_j^{(t,x)}\log_k p_j^{(t,x)}.$$

Clearly

$$s_\mu^{(t)}(T) = s_\mu^{(t,x)}(T:1),$$

and

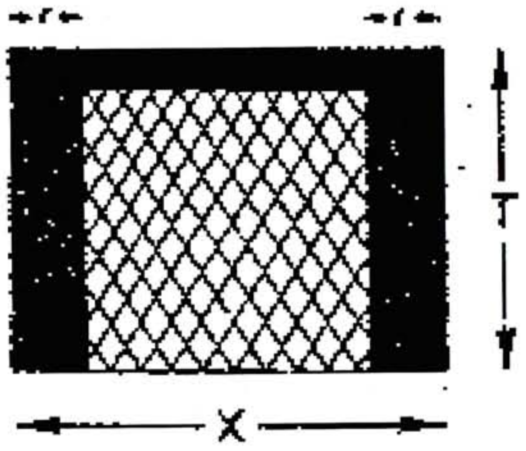
$$s_\mu^{(x)}(X) = s_\mu^{(t,x)}(1:X).$$

If no relation existed between configuration at successive time steps, then

$$s_\mu^{(t,x)}(T:X) \le s^{(t,x)}(T:X) \le X.$$

The cellular automaton rules introduce definite relations between successive configuration and tighten the bound of set entropy and measure entropy. In fact, the values of all sites in a $T \times X$ space-time patch are determined according to the cellular automaton rules by the values in the "rind" of the pitch. The rind contains only $X+2r(T-1)$ sites (where r is the "range" of the cellular automaton rule), so that

$$s_\mu^{(t,x)}(T:X) \le s^{(t,x)}(T:X) \le [X+2r(T-1)]/T.$$



The rind of the spacetime pattern

For large T and fixed X,

$$s_\mu^{(t,x)}(T:X) \leq s^{(t,x)}(T:X) \leq 2r.$$

If both X and T tend to infinity with X/T fixed, then the "information per site" $s_\mu^{(t,x)}(T:X)/X$ in $T \times X$ patch must tend to zero. The evolution of cellular automata can therefore never generate random space-time patterns.

Chapter 3

Theoretical analysis of the CA PNG

3.1 The Generator

There are a total of 256 cellular automaton rules for $k = 2$ and $r = 1$. Among these there are two rules that seem best as pseudorandom number generator proposed in [8].

In this thesis, we will mainly focus on Rule 30, which can be simply represented as

$$a_i^{(t)} = a_i^{(t-1)} \text{ XOR } (a_{i-1}^{(t-1)} \text{ OR } a_{i+1}^{(t-1)}) \tag{3.1}$$

Here XOR stands for exclusive or and OR stands for inclusive disjunction. From the figure in Chapter 2 about Rule 30, we can see that such complexity can arise in systems of simple construction. From the apparent randomness of the center vertical, it may show the potential for pseudorandom number generation.

The CA Rule 30 is essentially nonlinear. Nevertheless, its dependence on $a_{i-1}^{(t-1)}$ is in fact linear. This feature is the basis for many of its properties. In the Rule 30, the rule gives $a_i^{(t)}$ in term of $a_{i-1}^{(t-1)}$, $a_i^{(t-1)}$ and $a_{i+1}^{(t-1)}$. But the linear dependence on $a_{i-1}^{(t-1)}$ allows the rule to be rewritten as

$$a_{i-1}^{(t-1)} = a_i^{(t)} \text{ XOR } (\, a_i^{(t-1)} \text{ OR } a_{i+1}^{(t-1)} \,) \tag{3.2}$$

gives $a_{i-1}^{(t-1)}$ in terms of $a_i^{(t)}$,$a_i^{(t-1)}$ and $a_{i+1}^{(t-1)}$. This relation implies that the spacetime patterns shown can be found not only by direct time evolution by equation (3.1) from a given initial configuration, but also by extending spatially according to equation (3.2), starting with temporal sequences of values of two adjacent sites.

Random sequences are obtained from Rule 30 by sampling the values that a particular site attains as a function of time. In practical implementations, a finite number of sites are considered, and are typically arranged in a circular register. Given almost any initial "seed" configuration for the sites in the register, a long and seemingly random sequence can apparently be obtained.

3.2 Global Properties

This section will discuss about the behaviour of cellular automata starting from all possible initial states. The basic approach is to count the possible sequences and patterns that can occur and to characterize them. Most section in this chapter will discuss infinite size limit and section 3.6 will discuss the finite size effect.

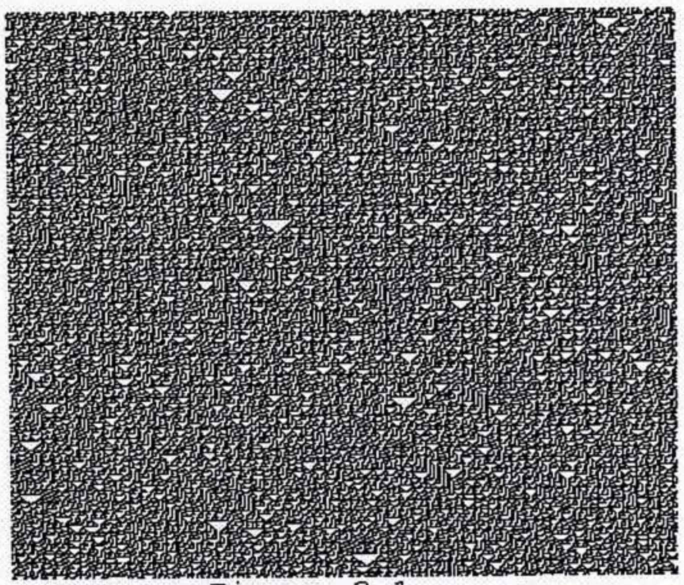


Figure 3.1

The figure above is a spacetime pattern produced by the Rule 30 starting from a disordered initial states (in which the value of each site is randomly chosen to be zero or one). Now we want to show that: given an appropriate initial condition, any sequences can be generated in an infinite cellular automaton with the Rule 30.

The Rule 30 can be considered as a mapping from one cellular automaton configuration to another. An important property of this mapping is surjective.

Any configuration $A^{(t)}$ can be obtained as the image of some $A^{(t-1)}$, according to $A^{(t)}=\mathbf{F}(A^{(t-1)})$. A possible configuration $A^{(t-1)}$ (may be not unique) can be found by starting with the candidate pair of site values, then extends to the left by equation (3.2). So, if all possible initial configurations are considered, then any configuration can be generated at some time step. Therefore with suitable initial conditions, any spatial sequences of site values can be reproduced.

Every length X spatial sequences of site values that occurs is determined by a length $X+2$ spatial sequences of site values on the previous time step. Since CA Rule 30 is surjective, so predecessors exists for any length X spatial sequences of site values.

For the Rule is surjective, there are exactly four predecessors for any sequences. Given values $a_i^{(t)}$, $a_{i-1}^{(t)}$ and so on, in one sequences, the values $a_{i+1}^{(t-1)}$ and $a_i^{(t-1)}$ in its predecessor can be chosen in all four possible ways; in each case the remaining $a_{i-j}^{(t)}$ are uniquely determined by equation (3.2).

Therefore, starting from an ensemble that contains all possible cellular automaton configurations with equal probabilities, each configuration will be generated with equal probabilities, throughout the evolution of the cellular automaton, so every possible spatial sequences of a particular length will occur with equal frequency.

One may also consider sequences of values attained by a single site as a function of time. Starting from an initial ensemble which contains all configurations with equal probabilities, all such sequences again occur with equal frequencies. For, given any temporal sequence, iteration of Equation (3.2) yields an equal number of initial configurations which evolve to it. The same is true for sequences of site values on lines at any angle in the spacetime pattern.

Entropies introduced in last chapter provided characterizations of the number of possible sequences that occur. Now, I repeat the definition with little change. Before the definition, I have to say that n is the length of the sequences, x is the specified sequences (the sequences may be spatial or temporal).

Definition (set entropy)

$$s = \lim_{n\to\infty} \frac{1}{n} \log_2 \sum_{j=1}^{2^n} \theta(p_j^{(x)})$$

where $\theta(p) = 1$ for $p > 0$ and $\theta(p) = 0$ for $p = 0$.

Definition (measure entropy)

$$s_\mu = \lim_{n\to\infty} \frac{-1}{n} \sum_{j=1}^{2^n} p_j^{(x)} \log_k p_j^{(x)}$$

For the cellular automaton of Equation (3.1), all possible sequences occur with equal probabilities (given an equal probability initial ensemble) so both entropies are maximal:

$$s_\mu = s = 1. \qquad (3.3)$$

Any reduction in entropies would reveal redundancy in the sequences, and would imply a lack of randomness. Equation (3.3) is a must for randomness.

Though Equation (3.3) implies that all possible sequences of value for single sites can occur along any spacetime pattern direction, the deterministic nature of the cellular automaton Rule 30 implies that only certain spacetime patches of values can occur. In fact, all the site values in a particular patch are completely determined by the values that appears on its upper on its upper, left and right boundaries. When these boundaries are specified, the values of remaining sites in the patch are redundant, and can be found simply by applying equation (3.1) and (3.2).

In general the degree of redundancy in such spacetime patterns can be characterized by the invariant of set and measure entropies for the cellular automaton mapping, given by

$$\mathbf{h} = \lim_{x\to\infty}\lim_{t\to\infty} s^{(t,x)}(T:X)$$

$$\mathbf{h}_\mu = \lim_{x\to\infty}\lim_{t\to\infty} s_\mu^{(t,x)}(T:X)$$

where $s^{(t,x)}(T:X)$ and $s_\mu^{(t,x)}(T:X)$ defined in last chapter.

By the result that

$$s_\mu^{(t,x)}(T:X) \le s^{(t,x)}(T:X) \le 2r$$

we have

$$\mathbf{h}_\mu \le \mathbf{h} \le 2$$

A calculation based on the method of [11] in fact shows that

$$\mathbf{h}_\mu \le 1.2$$

Therefore a knowledge of the time sequences of values of about 1.2 sites suffice in principle to determine the values of all other sites. In particular however the function which gives the initial configuration in terms of these temporal sequences seems to become intractably complicated, as discussed in Section 3.5.

3.3 Stability Properties

In this section, we will consider the change in the patterns produced by small perturbations in the initial states. Figure 3.2 shows the differences resulting from reversal of a single site value in a typical disordered initial configuration. The region affected increases in size with time, reflecting the instability of pattern generated.

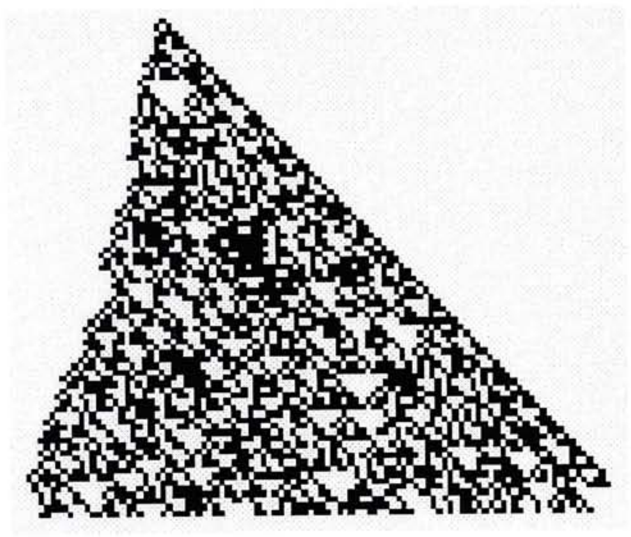


Figure 3.2 - The difference in pattern produced by the CA Rule 30 from two initial condition which difference only by one site.

The instability implies that information on localized changes eventually propagates throughout the cellular automaton. The rates of information transmission to the left and right are determined by the slope of the difference pattern in Figure 3.2 These in turn give left and right Lyapunov exponents λ_L and λ_R for the cellular automaton evolution.

The form of the cellular automaton Rule 30 immediately implies that

$$\lambda_R=1.$$

For consider a configuration in which the difference pattern has reach the site that $i = -1$. Whatever the current values of site that $i = 1$ and $i = 0$, the XOR in equation (3.1) leads to change in the new value of site that $i = 0$. Therefore $\lambda_R=1$ is the maximum allowed by the locality of the Rule 30.

Empirical measurements suggests that the left-hand side of the difference pattern expands at an asymptotically linear rate, with the slope [15]

$$\lambda_L=0.2428 \pm 0.0003$$

the above gives the average speed of the left-hand side of the difference pattern. In general, one can construct the analog of a Green's function[10], giving the probability that a site at a particular position and time will be affected by an initial perturbation.

Lyapunov exponents measure the rate of information transmission in cellular automaton, and provide upper bounds on entropies, which measure the information content of patterns generated by cellular automaton evolution. For surjective cellular automata it can be shown [10],

$$\mathbf{h}_\mu \leq (\lambda_L + \lambda_R)$$

consistent with Equation $\mathbf{h}_\mu \leq 2$. The existence of positive Lyapunov exponents is a characteristic feature of Class 3 cellular automata.

The difference pattern of Figure 3.2, and the related Green's function, measure the effect of initial perturbation on the values of individual sites. In studying random sequences generation, one must also consider the effect of such perturbation on time sequences of site values, say of length T. These sequences are always completely determined from the initial values of $2T + 1$ sites. However not all these initial values necessarily affect the time sequences. A change in any of the $T + 1$ left-hand initial sites necessarily leads to a change in at least one element of the time sequence. In the other hand, some change in the T right-hand initial sites have no effect on any element of the time sequence. It seems that the probability for a particular initial site to affect the time sequences decreases exponentially with distance to the right. The average number of the sites on the right which affect the time sequence is found to be approximately $0.26 + 1.9T$ [10]. Thus the total number of initial sites on which a length T time sequence depends on average approximately $1.91 + 1.19T$ [10]. This result is presumably related to the entropy.

3.4 Particular Initial States

This section considers evolution from particular special initial configuration.

Figure 3.4.1 shows on two scales the pattern produced by evolution from a configuration containing a single nonzero site.

There are some definite regularities. For example, diagonal sequences of sites on the left-hand side of the pattern are periodic, with small periods. In general, the value of a site at depth N from the edge of the pattern depends only on the sites at depths N or less; all the other sites on which it could depend always have value 0 because of the initial conditions given. As a consequence, the site down to depth N are independent of those deeper in the pattern, and in fact follow a shifted version of the cellular automaton Rule 30, with boundary conditions that constrain two sites at one end to have values zero. Since such finite cellular automaton has a total of 2^N possible states, any time sequence of values in it must have a period of at most 2^N. The corresponding diagonal sequences in the

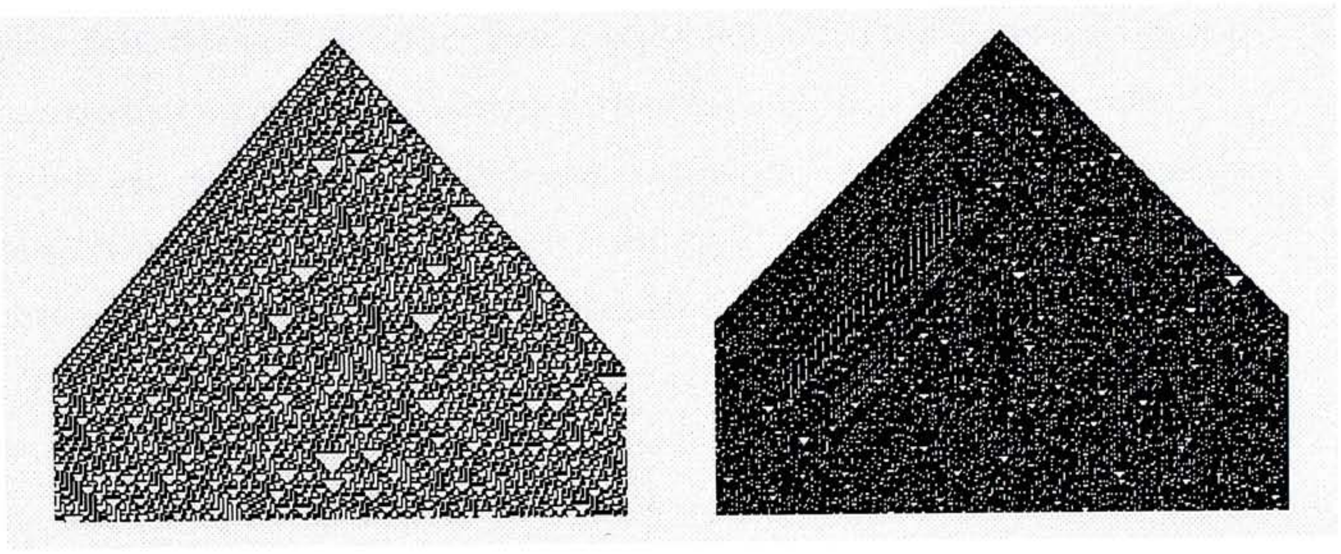


Figure 3.4.1

pattern Figure 3.4.1 must therefore also have periods not greater 2^N.

Depth	π_R	π_L
0	1	1
1	2	1
2	2	1
3	4	2
4	8	1
5	8	2
6	16	2
7	32	1
8	32	4
9	64	1
10	64	4
11	64	4
12	64	4
13	64	4
14	64	4
15	128	4
16	256	4

Table 3.4.1 above gives the actual periods of diagonal sequences found at various depths on left- and right-hand sides of the pattern in Figure 3.4.1. which π_R and π_L signify respectively periods for diagonal sequences on the right and left of the patterns, at the specify depth.

The short periods on the left-side of the pattern in Fig 3.4.1 are related to high degree of irreversibility in the effective cellular automaton rule for diagonal sequences in this case. Starting with any possible initial configuration, this cellular automaton always yields cycles with periods 2^j. The maximum value of j increases very slowly with N, yielding maximum cycle lengths which increase in jumps, on average slower than linearly with N. The actual sequences that occur near the left-hand boundary of the pattern in Fig 3.4.1 correspond to a particular set of those possible in this effective cellular automaton. In a first approximation, they can be considered uniformly distributed among possible N-site configurations, and their periods increase very slowly with N.

The effective rule for the right-hand side diagonal pattern in Fig 3.4.1 is a shifted version of Rule 30.

$$a_i^{(t)} = a_i^{(t-1)} XOR(a_{i+1}^{(t-1)} OR a_{i+2}^{(t-1)})$$

with boundary condition

$$a_{N-1}^{(t)} = a_{N-1}^{(t-1)} XOR a_N^{(t-1)}$$
$$a_N^{(t)} = a_N^{(t-1)}$$

This system is exactly reversible; all of its 2^N possible configurations have unique predecessors. All the configuration thus lie in cycles, and again the cycles have periods of the form 2^j. As the length of the longest cycles is a function of N, one [5] may yield

$$\log_2 \Pi_N \simeq 0.5(N+1)$$

This length is small compared to the total number of states 2^N; few states in fact lie on such longest cycles. Nevertheless, the periods of the right-hand diagonal sequences in Fig 3.4.1 do seem to increase roughly exponentially with depth, as suggested by Table 3.4.1

The boundary in Fig 3.4.1 between regular behaviour on the left and irregular behavior on the right seems to be asymptotically linear, and move to the left with speed 0.25. A statistical argument for this result can be given in analogy with that λ_L=0.2428 $\pm$ 0.0003. Each site at depth d on the left-hand side of the pattern could in principle be affected by sites down to depth d arbitrarily far up in the pattern. In practice, however, it is unaffected by changes in sites outside a cone whose boundary propagates at speed $\lambda_L \simeq 0.25$. Thus the irregularity on the right spreads to the left only at this speed.

With diagonal sequences at angles ± 1 in Figure 3.4.1 must ultimately become periodic, sequences closer to the vertical need not. In fact, no periodic has been found in any such sequences. The central vertical (temporal) sequences has, for example, been tested up to length 2^{19} [5], and no periodicity is seen. For if two sequences were both periodic, then it would fellow that all sequences to their right must also be, which would lead to a contradiction at the edge of the pattern.

Not only no periodicity has been detected in the center vertical sequences of Figure 3.4.1; the sequences has also passed all other statical tests of randomness applied to it, as discussed in section 3.8.

While individual sequences seem random, there are local regularities in the overall pattern of Fig 3.4.1. Example are the triangular regions of zero sites. Such regularities are associated with invariants of the CA Rule.

The particular configuration in which all sites have value 0 is invariant under the cellular automaton Rule 30. As a consequence, any string of zeros that appears can be corrupted only by effects that propagate in from its ends. Thus each string of zeros that is produced leads to a uniform triangular region.

Period	Element
1	0
	01
3	000011111001
4	0000001
	0000111
	0010011
	0111111

Table 3.4.2 above and Fig 3.4.2 give other configuration which are periodic under the Rule 30. Again, any string that contains just the sequences in these configuration can be corrupted only through end effects, and leads to a regular region in spacetime pattern generated by Rule 30.

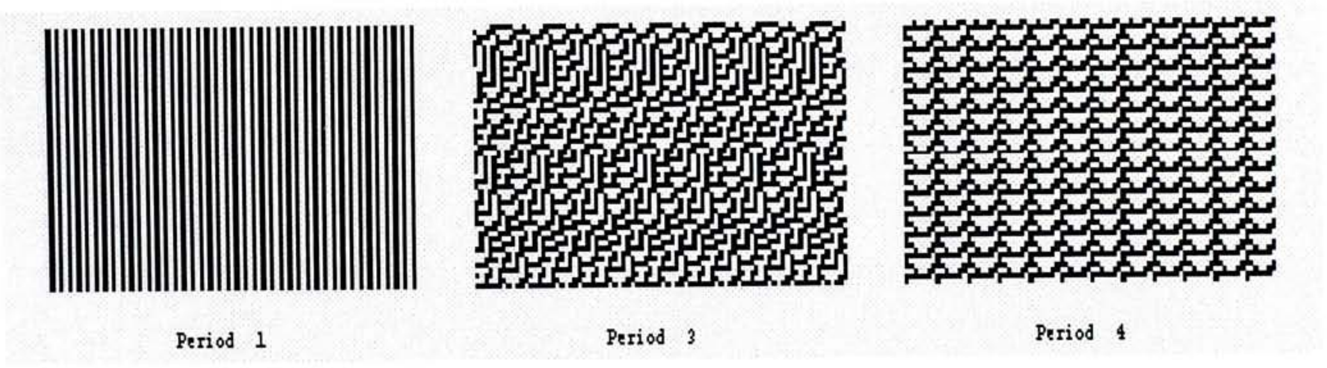


Figure 3.4.2

In general, there are finite set of configurations with any particular period p under a permutive cellular automaton rule such as Rule 30. The configuration may be found by starting with a string of length $2p$, then testing whether this and the string yields through Equation (3.2) on the left are in fact invariant. The string to be tested need never longer than 2^{2p} , since such a string can contain all possible length 2p string. Therefore the periodic configuration consists of repetitions of blocks containing 2^{2p} or less site values.

3.5 Functional Properties

Cellular automaton rule such as Rule 30 can be considered as a function f which map three Boolean values to one. Iteration of these values for say t steps correspond to function of $2t + 1$ Boolean values. The complexity of these functions reflects the complexity of cellular automaton evolution.

The complexity of a Boolean function can be characterized by the number of logic gates that would be needed to evaluate it with a particular kind of circuit, or the number of terms that it would have in particular symbolic representation. Explicit evolution according to the cellular automaton Rule 30 corresponds to a circuit with $O(t^2)$ components and depth t, but for purposes of comparison, it is convenient to consider fixed depth representation. One such representation is disjunctive normal form (DNF), in which the function is written as a disjunction of conjunctions. A two-level circuit can be constructed in direct correspondence with this form.

For Rule 30, the DNF of Equation (3.1) can be written as:

$$f(a_{-1}, a_0, a_1) = (\overline{a}_{-1}a_0) + (a_{-1}\overline{a}_0\overline{a}_1) + (\overline{a}_{-1}a_1)$$

where + stand for OR, concatenation for AND and bar for NOT.

The general problem of finding the absolute shortest representation for any arbitrary Boolean function, even in DNF, is NP-complete [12] , and so presumably requires an exponential time computation. However a definite approximation can be found in terms of "prime implicants"[13]. Each prime implicant can be used as a term in a DNF for the function. The number of prime implicants required gives a measure of the total number of terms in the DNF and thus of the complexity of the function.

The minimal DNF obtained with prime implicants for the function corresponding to two iterations of the cellular automaton Rule 30 is

$$\begin{aligned} f^2(a_{-2},a_{-1},a_0,a_1,a_2) =& (\overline{a}_{-2},\overline{a}_{-1},\overline{a}_0,a_1,\overline{a}_2) + (\overline{a}_{-2},a_{-1},a_0,a_1,\overline{a}_2) \\ &+(a_{-2},\overline{a}_{-1},a_0,a_1,\overline{a}_2) + (a_{-2},a_{-1},a_0,\overline{a}_1,\overline{a}_2) \\ &+(a_{-2},\overline{a}_{-1},\overline{a}_1,\overline{a}_2) + (\overline{a}_{-2},\overline{a}_{-1},\overline{a}_0,a_2) \\ &+(a_{-2},\overline{a}_{-1},a_0,a_2) + (a_{-2},a_{-1},a_0,a_2) + (a_{-2},a_{-1},\overline{a}_0) \end{aligned}$$

Table 3.5.1 below gives the number of prime implicants for successive iterations of the Rule 30. These results are plotted in Fig 3.5.1. For arbitrary Boolean functions of 2t+1 variables, the number of prime implicants could increase like 4^t. In practice, however, a least square fit to the data of Table 3.5.1 suggests growth like $4^{0.77t}$.

t	P.I.	Min
1	3	3
2	9	7
3	23	17
4	76	41
5	185	105
6	666	272

Various efficient methods are known to find DNF that are somewhat simpler than those obtained using prime implicants. With one such method [14], the DNF can be reduced to

$$\begin{aligned} f^2(a_{-2},a_{-1},a_0,a_1,a_2) =& (\overline{a}_{-2},\overline{a}_{-1},\overline{a}_0,a_1) + (\overline{a}_{-2},a_{-1},a_0,a_1) \\ &+(\overline{a}_{-2},\overline{a}_{-1},\overline{a}_0,a_2) + (\overline{a}_{-2},a_{-1},a_0,a_2) \\ &+(a_{-2},\overline{a}_1,\overline{a}_2) + (a_{-2},\overline{a}_{-1},a_0) + (a_{-2},a_{-1},\overline{a}_0) \end{aligned}$$

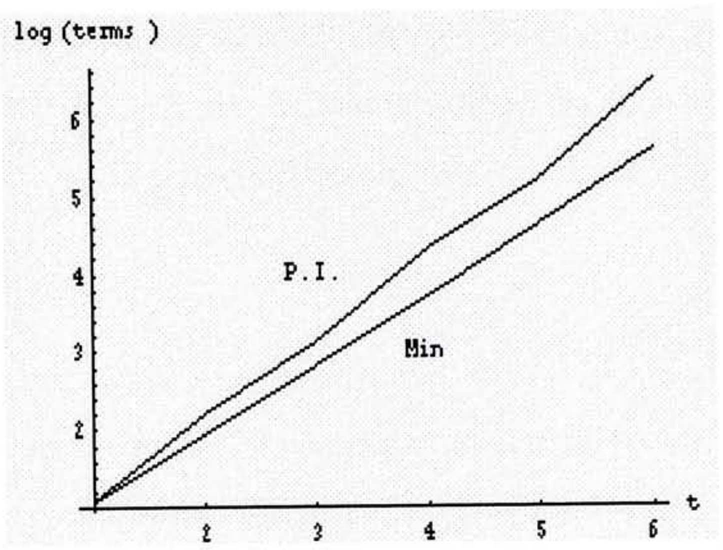


Figure 3.5.1

The sizes of the minimal DNF obtained by this method for iteration of the Rule 30 are shown also in Table 3.7.1 and Figure 3.7.1. They are seen to grow more slowly than those obtained with prime implicants; the data given are again fit by exponential growth like $4^{0.65t}$.

The rapid increase in the size of the minimal DNF found for the iteration of the Rule 30 indicates the increasing computational complexity of determining the result of evolution according to Rule 30, and supports the conjecture of its computational irreducibility.

The results of Table 3.5.1 and Figure 3.5.1 concern the difficulty of finding the outcome of cellular automaton evolution according to the Rule 30 from a given initial state. One may also consider the problem of deducing the initial state from time sequences of site values produced in the evolution. Given say t steps in the time sequences of values for two adjacent sites, the initial configuration up to t sites to the left can be deduced directly by the iteration of Equation (3.2). The combinatorial results of section 3.2 indicate in fact that only about 1.2 such temporal sequences should on average be required, and in principle from a single sufficiently long temporal sequences, it should be possible to deduce a complete initial configuration for a finite cellular automaton. In practice, the necessary computation seems to become increasing intractable as the size of systems increases.

Given a particular temporal sequences, say at position 0, Equation (3.2) uniquely determines the values of all sites in a triangle to left as a function of values in the temporal sequences at position 1. The number of values in the position 1 temporal sequences on which a given site depends varies with the form of the position 0 sequences. For example, if the position 0 sequence consists solely of ones, then the whole triangle of sites is completely determined, entirely independent of the position 1 sequence. Table 3.5.2 gives some results from considering the dependence of the site value a_{-t} at position $-t$ on the position 1 sequence, for all 2^t possible position 0 sequences. The number of values in the position 1 sequences on which a^{-t} depends seems to be roughly Poisson distributed, with a mean that grows like $0.4t$ as shown in Figure 3.5.2. This is consistent with the combinatorial result.

n	Number of Variable	P.I.
2	0.5	1
3	1	2
4	1.375	3
5	1.125	3
6	2.281	12
7	2.828	17
8	3.164	26

Table 3.5.2 above also gives some properties of the prime implicants forms for a_{-t}. It is clear that the complexity of the function that determines a_{-t} from temporal sequences grows with t, probably at increasingly rapid rate. Again

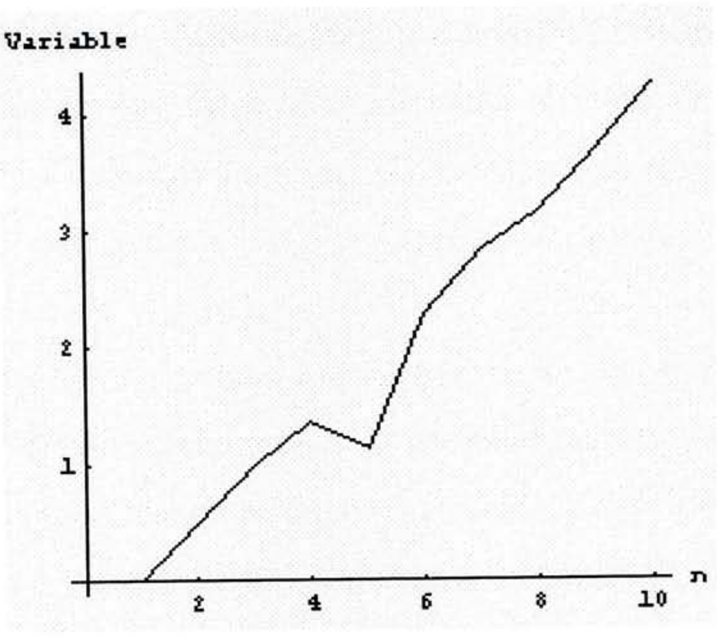


Figure 3.5.2

this suggests that the problem of deducing the initial sequences for evolution according to the Rule 30, while combinatorially possible, is computational complex.

3.6 Computational Theoretical Properties

The discussion of the previous section are considered as giving a characterization of the computational complexity of iterations of the cellular automaton Rule 30 in a particular simple model of computation. The results obtained suggest that at least in this model, there is no shortcut method for finding the outcome of the evolution; the computation required are not less than for a explicit simulation of each time step. As discussed above, one suspects in fact that the evolution is in general computationally irreducible, so that no possible computation could find its outcome more efficiently than by direct simulation.

This would be the case if the cellular automaton Rule 30 could act as an efficient universal computer [16], so that with an appropriate initial state, its evolution could mimic any possible computation. In particular, it could be that the problem of finding the value of a particular site after t steps, must take a time polynomial in t on any computer. (Direct simulation takes $O(t^2)$ time on a serial-processing computer, and $O(t)$ time with $O(t)$ parallel processors.)

In addition to studying cellular automaton evolution from given initial configuration, one may consider the problem of deducing configurations for the cellular automaton from partial information such as temporal sequences. In particular, one may study the computational complexity of finding the seed for a cellular automaton in a finite region from the temporal sequences it generators.

There are 2^N possible seeds for a size N cellular automaton, and one can always find which ones produce a particular sequence by trying each of them in turn. However, such a procedure would rapidly become impractical. The result in last section suggest a slightly more efficient method. If it were possible to find two adjacent temporal sequences, then the seed could be found easily using Equation (3.2). Given only one temporal sequences, some elements for the seed are initially undetermined. Nevertheless, in a finite size system, say with periodic boundary conditions, one can derive many distinct equations for a single site value. The site value can then be deduced by solving the resulting system of simultaneous Boolean equations. The equations will typically involve variable. As discussed in last section, the number of variables seems to be Poisson-distributed with a mean around $0.4N$.

The general problem of solving a Boolean equation in n variable is NP-complete, and so presumably cannot be solved in time polynomial in n. In addition, it seems likely that the average time to solve an arbitrary Boolean equation is correspondingly long. To relate the problem of deducing the seed discussed above to this would require a demonstration that the Boolean equations generated were in a sense uniformly distributed over all possibilities. Out of all 2^{2^n} n-variable equation, the problem here typically involves $O(2^n)$, but these seem to have no special simplifying features. At least with method discussed above, it is conceivable that the problem of deducing the seed is equivalent to the general problem of solving Boolean equation, which is NP-complete.

3.7 Finite Size Behaviour

Much of the above discussed have concerned the behaviour of the cellular automaton Rule 30 in the idealized limit of an infinite lattice of sites. But practical implementations must use finite size registers, and certain global properties can depend on the size and boundary conditions chosen.

The total number of possible states in a size N cellular automaton is 2^N. Evolution between these states can be expressed by a finite states transition diagram. Fig 3.7.1 give examples of such diagrams for the cellular automaton of the Rule 30 with $N = 9$ periodic boundary conditions, such that

$$\begin{aligned} a_1^{(t)} &= a_N^{(t-1)} \text{ XOR } (a_1^{(t-1)} \text{ OR } a_2^{(t-1)}) \\ a_N^{(t)} &= a_{N-1}^{(t-1)} \text{ XOR } (a_N^{(t-1)} \text{ OR } a_1^{(t-1)}) \end{aligned}$$

Table 3.7.1 summarizes some of their properties. Fraction of Longest Cycle is the total number of configuration involve in the longest cycle (including the configuration "attract" to the longest cycle) against total configuration. Cycle Fraction is the total number of configuration in cycles against the total number of configuration, and transient means that if one the configuration is not in cycle, average time step that configuration reach the cycles. We find that the results

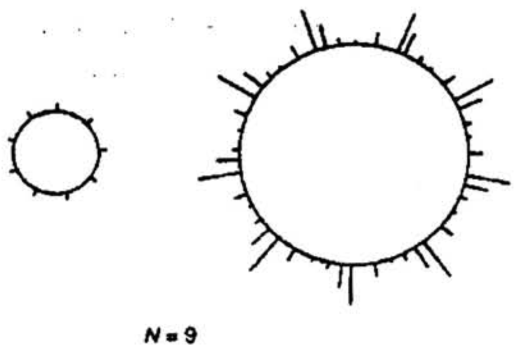


Figure 3.7.1

are seen to depend not only on the magnitude of N , but also presumably on its number theoretical properties.

N	No. of Cycle × Cycle Long	Fraction of Longest Cycle	Cycle Fraction	Transient
4	1×8 3×1	0.75	0.69	0.5
5	1×5 1×1	0.94	0.19	4.3
6	3×1	1.00	0.05	3.3
7	1×63 7×4 1×1	0.60	0.72	0.4
8	1×40 1×8	0.88	0.20	3.1
9	1×171 1×72 1×1	0.81	0.48	1.1
10	2×15 1×5 3×1	0.82	0.04	14.8
11	1×154 11×17 1×1	0.76	0.17	3.3
12	4×102 1×8 4×3 3×1	0.93	0.11	4.4
13	1×832 1×260 1×247	0.32	0.17	2.2
14	1×1428 2×133 1×112 2×84 1×63 1×14 3×1	0.84	0.13	2.7
15	1×1455 5×30 5×9 15×7 4×5 1×1	0.93	0.05	5.7

Table 3.7.1

Each site transition diagram contains a set of cycles, fed by trees representing transients. The cycle may be considered as "attractors" to which states in their "basins of attractions" irreversibly evolve.

There are many regularities in the structure of the state transition diagrams obtained from the Rule 30. The evolution is not well-approximated by a random mapping between 2^N states.

A first observation is that most configurations have unique predecessors under the Rule 30, so there is little branching in the state transition diagram. In fact, it can be shown that a configuration has a unique predecessor unless it contains a pair of value zero sites separated by a sequences of $3n+1$ value one sites (with $n \geq 0$) or unless N is divisible by 3, and all site have value 1. In the former case, the configuration has exactly zero or two predecessors; in the latter case, it has three. The numbers of configuration with zero or two predecessors are equal when N is not divisible by 3; there are two more with zero predecessors when $3|N$. For large N, the number of configurations with zero or two predecessors behaves as κ^N, where $\kappa \simeq 1.696$ is the real root of $4\kappa^3 - 2\kappa^2 - 1 = 0$. Since the total number of configurations grows like 2^N, the fraction of nodes in the state transition diagram that are branch points tends exponentially zero.

A second observation is that there are often many identical parts in the state transition diagrams of Table 3.7.1 and Figure 3.6.1. This is largely a consequence of shift invariance. States in a cellular automaton with periodic boundary conditions that are related by shift evolve equivalently. Therefore, for example, there are often several identical cycles, related by shifts in their configurations. In addition, the periods of the cycles are often divisible by N or its factors, since they obtain several sequences of configurations related by shifts. The transient tree that feed each of these sequences are then identical.

The evolution of finite cellular automaton with periodic boundary conditions is equivalent to the evolution of an infinite cellular automaton with periodic initial configuration. Therefore, the results on cycle length distributions in Table 3.7.1 can be considered as inverse to those Table 3.5.2 on configuration with given temporal periods. Cycles of lengths corresponding to these temporal periods occur whenever N is divisible by the spatial periods of these configurations. Such short cycles are absent if N has none of these factors.

For large N, the state transition diagrams for the Rule 30 appear to be increasingly dominated by a single cycle. This cycle is longer than the others, and its basin of attraction is large enough that most arbitrary chosen initial states evolve to it. The low degree of branching in the transient trees implies that the points reached from the arbitrary initial states should be roughly uniformly distributed around the cycle.

The shorter cycles in Table 3.7.1 can be considered as related to subsets of states invariant under cellular automaton rule. With N even, for example, configuration which consist of two identical $N/2$ subsequences can evolve only to configuration of the same type. Once such a configuration has been reached, the evolution is trapped within this subset of configurations, and must yield shorter cycles. In general, there may exist subsets of states within certain special symmetry properties that are preserved by the cellular automaton rule. Initial states with particular, symmetrical forms can be expected to have these properties, and thus to be trapped in subsets of state space, and to yield short cycles. For example, with $N = 36$, a configuration containing a single nonzero site evolves to a length 2844 cycle, while most initial evolve to the longest cycle, with 2237472 states.

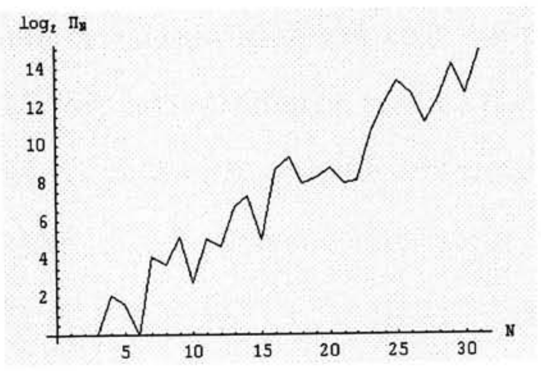


Figure 3.7.2

N	Longest Cycle Length : Π_N	N	Longest Cycle Length : Π_N
4	8	32	2002272
5	5	33	2038476
6	1	34	5656002
7	63	35	18480630
8	40	36	2237472
9	171	37	49276415
10	15	38	9329228
11	154	39	961272
12	102	40	19211080
13	832	41	51151354
14	1428	42	109603410
15	1455	43	93537212
16	6016	44	192218312
17	10845	45	75864495
18	2844	46	261598274
19	3705	47	811284813
20	6150	48	3035918676
21	2793	49	9937383652
22	3256	50	593487780
23	38429	51	3625711023
24	185040	52	20653434880
25	588425	53	40114679273
26	312156	54	7551779562
27	67554		
28	249165		
29	1466066		
30	306120		
31	2841150		

Table 3.7.2

In the infinite size limit, patterns such as that of Figure 3.4.1 generated by the cellular automaton of the Rule 30 never become periodic, but with a total of N sites, a cycle must occur after 2^N or less steps. Table 3.7.2 and Figure 3.7.2 give the actual maximal cycle length Π_N found. A roughly exponential increase of Π_N with N is seen, and a least square fit to the data of Table 3.7.2 yields

$$\log_2 \Pi_N \simeq 0.61(N+1) \qquad (3.7.1)$$

Note that if the state transition diagram correspond to an entirely random mapping between the 2^N cellular automaton states, then cycles of average length $2^{N/2}$ would be expected. The cycles actually obtained are significantly longer. The exponent in Equation (3.7.1) may be related to the entropy $\mathbf{h}_\mu \leq 2$ as a result of the expansivity or instability of the mapping discussed section 3.3.

If there were very short cycles, the sequences produced by the cellular automaton would readily be predictable. Therefore if in fact no such prediction can be made by any polynomial time computation, the length of the cycles that occur should in general increase asymptotically faster than polynomial in N. This behaviour is supported by Equation (3.7.1)

If indeed the evolution of cellular automaton Rule 30 is computationally irreducible, then a complex computation may always be required to determine for example the lengths of cycles that appear. For in this case, there are effectively be no better way to find the succession of states that occur, except by explicit application of the Rule 30. One expects in fact that the problem of finding say whether two configurations lie on the cycle is PSPACE-complete, and so presumably cannot be solved in a time polynomial in N, but rather essentially requires a direct simulation of the cellular automaton evolution.

The cycle structure of finite cellular automata depends in detail on the boundary conditions chosen. Table 3.7.3 gives the maximal cycle length found for the Rule 30 with shift register boundary conditions such that

$$a_1^{(t)} = a_3^{(t)} \text{ XOR } (a_4^{(t)} \text{ OR } a_1^{(t-1)})$$
$$a_2^{(t)} = a_4^{(t)} \text{ XOR } (a_1^{(t-1)} \text{ OR } a_2^{(t-1)})$$
$$a_N \text{ and } a_{N-1} \text{ is similar to } a_1 \text{ and } a_2$$

The result differ substantially from those with boundary conditions given in Table 3.7.2.

N	Longest Cycle Length : Π_N
4	5
5	2
6	7
7	4
8	17
9	65
10	6
11	57
12	50
13	118
14	185
15	257
16	481
17	907
18	1681
19	707
20	2679
21	5630
22	1368
23	31241
24	3567
25	60503
26	4752
27	46519
28	35569
29	207197
30	149899
31	482717

Table 3.7.3

Other boundary conditions may also be considered. Among them are twisted ones, in which a_1 and a_N are negated in periodic boundary conditions. The maximum cycle length found with such boundary conditions seem typically shorter than in purely periodic case.

One may in addition consider boundary conditions in which the boundary site value are fixed, rather than being periodic identified. Different cycles are obtained in different cases; all those investigated nevertheless give maximal cycle length shorter than those in Table 3.6.2 found with periodic boundary conditions.

What has been discussed so far are cycles in complete finite cellular automaton configurations. But in obtaining random sequences one samples single sites. The sequences found could potentially have periods which sub-multiples of the periods for complete configuration. However, permutive rule such as Rule 30 cannot occur.

The state transition diagrams summarized in Table 3.7.1 give the number of complete N-site configurations that can occur at various stages in the evolution of the cellular automaton rule 30. One may also consider the number of single site temporal sequences that can occur. Table 3.7.4 gives the fraction of the 2^L possible length L temporal sequences that are actually generated from any of the 2^N possible initial states in a size N cellular automaton evolution according to the Rule 30 with periodic boundary conditions. Whenever $N \geq L+2$, all possible sequences seem to be generated. They appear with roughly equal frequencies.

L	3	4	5	6	7	8	9	10	11	12	13	14	15
3	0.500	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
4	0.250	0.625	0.875	0.938	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
5	0.125	0.313	0.656	0.844	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
6	0.063	0.156	0.344	0.594	0.906	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
7	0.031	0.078	0.180	0.352	0.609	0.891	1.000	1.000	1.000	1.000	1.000	1.000	1.000
8	0.016	0.039	0.094	0.188	0.328	0.633	0.949	0.992	1.000	1.000	1.000	1.000	1.000
9	0.008	0.020	0.047	0.094	0.168	0.361	0.668	0.895	0.996	1.000	1.000	1.000	1.000
10	0.004	0.010	0.023	0.047	0.085	0.195	0.386	0.644	0.917	0.989	1.000	1.000	1.000
11	0.002	0.005	0.012	0.023	0.042	0.102	0.204	0.377	0.666	0.897	0.995	1.000	1.000
12	0.001	0.002	0.006	0.012	0.021	0.052	0.105	0.209	0.385	0.669	0.913	0.995	1.000
13	0.000	0.001	0.003	0.006	0.011	0.026	0.054	0.105	0.209	0.385	0.669	0.913	0.995
14	0.000	0.001	0.001	0.003	0.005	0.013	0.027	0.053	0.109	0.209	0.397	0.671	0.906
15	0.000	0.000	0.001	0.001	0.003	0.007	0.013	0.027	0.055	0.109	0.215	0.399	0.668

Table 3.7.4

3.8 Statistical Properties

The sequences generated by the cellular automaton Rule 30 may be considered effectively random if no feasible procedure can identify a pattern in them, or allow their behavior to be predicted. Even though it may not be possible to prove that no such procedure can exist, circumstantial evidence can be accumulated by trying various statistical results on sequences generated by the Rule 30 with those calculated for sequences whose elements occur purely according to probabilities.

To establish the validity of the Rule 30 as a random sequence generator, one should apply a variety of statistical procedures, related to various different kinds of calculations. The choice of tests is necessarily as ad hoc as the choice of calculations done. At the end of the chapter lists those used here. While quite ad hoc, the tests seem to be sensitive, and reasonably independent.

As an example, consider the "equidistribution" or "frequency" test. If a sequence of zeros and ones is to be random, the digits zero and one must occur in it with equal frequency. In general, in fact, all 2^n possible length n blocks of digits must also occur with equal frequency. However, in a finite sample of length m, there are expected to be statistical fluctuations, which lead to slightly different numbers of zeros and ones. As a consequence, one can never definitely conclude by studying a finite sample that the complete sequence is not random. One can calculate the probabilities that truly random sequence would have the properties seen in the finite sample.

To do this, one evaluates χ^2, defined in terms of the observed and expected frequencies p_0 and p_e as

$$\chi^2=\sum_1^{\nu}(p_0 - p_e)^2/p_e$$

Here ν gives the number of degrees of freedom, or the number of distinct objects whose frequencies are included in the sum. If blocks of length n are studied then $\nu=2^n$. Now one must find the probability that a value of χ^2 larger than that observed would occur for a random sequence. This "confidence interval" is obtained immediately from the integral of the χ^2 distribution.

If the confidence interval is very close to zero or one, then the observed χ^2 is unlikely to be produced from a random sequence, and one may infer that the observed sequence is not random. Of course, if say a total of k tests are done, it is to be expected that the confidence interval for at least one of them will be less than $1/k$. Evidence for nonrandomness in a sequence must come from an excess of confidence interval values close to zero or one, over or above the number expected for a uniform distribution.

	N=17,L=8k	N=17,L=64k	N=23,L=64k	N=29,L=64k	N=37,L=64k	N=49,L=64k
A	0.0039	1.0000	0.0456	0.7375	0.3852	0.8003
B	0.0171	0.9944	0.3391	0.4888	0.1010	0.1494
C	0.4164	0.4783	0.7256	0.4847	0.4083	0.9407
D	0.3227	0.9998	0.1506	0.1434	0.1678	0.6074
E	0.4576	0.4484	0.6790	0.8492	0.5414	0.7991
F	0.4306	0.8644	0.8751	0.5590	0.6681	0.6606
G	0.2942	0.9944	0.1232	0.7359	0.4448	0.6961

Table 3.8.1 - With initial state as single nonzero site, and $k = 1024$. The number given are the probabilities for statistical averages of truly random sequences to exceed those of the sequences analyzed. The numbers should be uniformly distributed between 0 and 1 if sequences analyzed are indeed truly random. Accumulation close to 0 and 1 suggest derivations from randomness. Such accumulations are seen in this case only when the period of the cellular automaton is comparable to the length of the sequence sampled.

Table 3.8.1 gives results from the statistical tests described at the end of the section for sequences generated by the Cellular Automaton Rule 30 in a finite circular register. Except when the sample sequence is comparable in length to the period of the system, as given in Table 3.7.2, no significant deviations from randomness are found.

If deviations from randomness were detected by some statistical procedure, then this procedure could be used to make statistical predictions about the sequences. In addition, it could be used to obtained a compressed representation for the sequence, and would demonstrate that the sequence did not have maximal information content. The fact that deviations from randomness have not been found by any of the statistical procedures considered lends strong support to the belief that sequences produced by the Rule 30 with large N are indeed random for practical purposes.

3.8.1 statistical test used

The statistical test are taken from [21].

The sequences studied consists of strings of binary bits. In many of the test, these bits are grouped into blocks; either 8 or 4. The possible bit sequences in these blocks can be represented by integer value between 0 and 255 or 15, respectively.

n-blocks mean a blocks with n bits.

A Block Frequency Distribution

Each of the 2^n possible n-blocks should occur with equal frequency. (n=8 are used.)

B Gap Length Distribution

The lengths of runs of n-blocks whose value are all greater than i_2 or less than i_1 should follow a binomial distribution. (n=8, i_1=100 i_2=200 are used; runs longer than 16 blocks are lumped together.)

C Distinct Blocks Distribution

The frequencies with which p out of q successive m-blocks are distinct should follow a definite distribution. (m=4 q=4 are used.)

D Block Accumulation distribution

The number of successive n-block necessary for all possible m-blocks to appear in order as their first m elements should follow to a definite distribution.(n=8,m=3 are used; number greater than 40 are lumped together.)

E Permutation Frequency Distribution

The value of q successive n-blocks should occur in all $q!$ possible ordering with equal frequency. (n=8, q=5 are used.)

F Monotone Sequence Length Distribution

The lengths of sequences in which successive n-blocks have monotonically increasing value should follow a definite distribution. (n=8 is used; length greater than 6 are lumped together; elements immediately following each run are discarded to make successive runs statistically independent.)

G Maxima distribution

The maximum values of n-blocks in sequences of successive of q n-blocks should follow a power law distribution. (n=8,q=8 are used.)

Chapter 4

Practical Implementation of the CA PNG

4.1 The implementation of the CA PNG

The simplicity and intrinsic parallelism of the Cellular Automaton Rule 30 makes possible efficient implementation on many kinds of computers.

On a serial-processing computer, each site could be updated in turn according to and updated to the Rule 30, but in practical, site values can be repersented by single bits in say a 32-bit word, and updated in parallel using standard word-wise Boolean operations.

On a synchronous parallel-processing computer, different sites or groups of sites in the cellular automaton can be assigned to different processors. They can then be updated independently, using the same instruction, and with only local communications.

Very efficient hardware implementation of the Rule 30 should be possible. For short registers, explicit circuitry can be included for each site. And for long registers, a pinelined approach analogous to a feedback shift register can be used.

The evidence presented above suggests that the Cellular Automaton Rule 30 can serve as a practical random sequence (pseudorandom number on $\{0, 1\}$) generator. The most appropriate detailed choices of parameters depends on the application intended. The most obvious constraint isone of cycle length. To obtain a cycle length larger than $2^{32} \simeq 4 \times 10^9$, Table 3.7.2 shows that a circular registor of length of $N = 49$ can be used. Cycle lengths tends to increase with N, but Table 3.7.2 shows some irregularities. Therefore it is not clear, for example, how large N need to be obtain a cycle length larger than $2^{64} \simeq 10^{19}$, but base on Equation 3.7.1, a value $N = 127$ should certainly suffice.

Random sequences can be obtained by sampling the sequence of values of a particular site in a register updated according to the Rule 30. The theoretical and statistical studies described above support the contention that such sequences show no regularities.

Sequences could potentially be obtained more quickly by extracting the values of several sites in the register at each time step, but $\mathbf{h}_\mu \leq 2$ implies that some statistical correlations must exist between these values. The correlations are probably minimized if the sites sampled are equally spaced around the register.

The random sequences obtained from the Rule 30 have an equal fraction of 0 and 1. Many applications, however, involve random binary choices with unequal probabilities. There is a simple algorithm to obtain digits with arbitrary probabilities. [5]. First write the probabilities p for outcome 1 as a binary number. Then generate a random binary sequence s with length equal to this number. The output is obtained by an iterative procedure. Begin with the "current result" of 1. Then starting from the least significant digit in p, successively find a new result by combining the old result with corresponding digit of s, using a function AND or OR, depending on whether the digit in p is 0 or 1, respectively. The final result thus obtained is equal to 1 with probability exactly p.

Configurations in two length N registers with slightly different seeds should become progressively less correlated under the Rule 30 as a result of the instability discussed in the section 3.3. The characteristic time for this process is governed by the left and right Lyapunov exponents λ_L and λ_R, and should be $0.8N$. Therefore, if several sequences are to be generated with seeds that differ only slightly, then the Rule 30 should be applicated at least $O(N)$ times to the seeds before beginning to extract random sequences.

4.2 Applied to the set of integers

To generate a random integer on a set of integers size is equal to n, for example: $\{0, 1, 2, ..., n-1\}$, just choose a N that the "width" of the CA PNG, satisfies

$$n \leq 0.61(N+1)$$

then select a site to capture its temporal sequences with length equal to $\log_2 n + 1$. After capture the sequences, simply change it to decimal from binary. If generate a number that greater then $n-1$, discard it and generate again.

This method will have a expected run time $(2n - \frac{4}{n+2})\lceil \text{Log}_2 n \rceil$ for the worst case that the size of the set is $2^m + 1$ for some integer m and $\lceil \text{Log}\ _2 n \rceil$ for the best case that the size of the set is the power of 2.

From [2], we can found an algorithm called DDG-tree alogorithms (Discrete distribution generateing tree algorithms) that can generate any distribution of

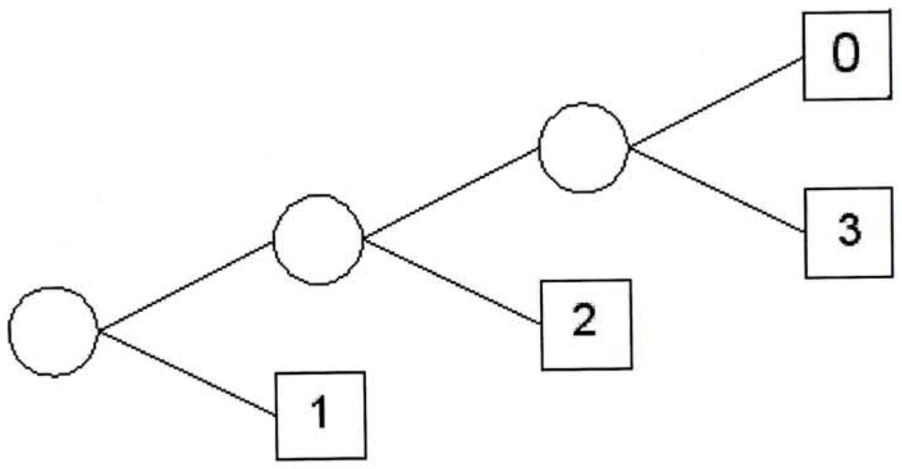


Figure 4.2.1

random numbers, and the above is the typical one that the distribution is uniform.

The analysis of the algorithm have been in [2] and there will be an introducion of that algorithm.

For example, if we want to generate the set of integers $\{0, 1, 2, 3\}$ with probabilities $1/8, 1/2, 1/4, 1/8$, we write the probabilities in binary form that

$$1/8 = (0.001)_2$$
$$1/2 = (0.1)_2$$
$$1/4 = (0.01)_2$$

Then construct a tree that for generate 0, we have a termial node on level 3 from the root, according to $(0.001)_2$. As same as for generate 1, we have a termial node on level 1 from the root. In the figure 4.2.1, we have shown the DDG-tree of the algorithm, which go to top-right from the left-node means get 1 from the CA PNG, and to bottom-right means get 0 from the CA PNG. Therefore, from the figure 4.2.1, we know that we generate 0 for 000, 1 for 1, 2 for 01 and 3 for 001.

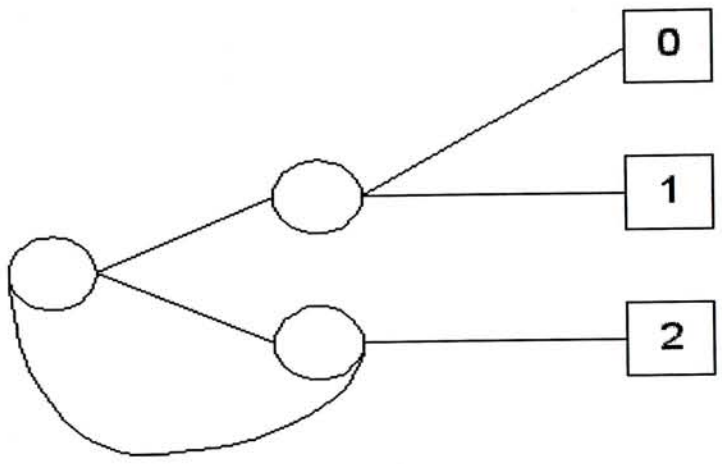


Figure 4.2.2

There [2] have been shown that for any dicrete distribution, with probabilities are rational, this algorithm will be halted in expected finite time, and there [2] is an algorithm to generate that tree.

The figure 4.2.2 shown the DDG-tree for generate a pesudorandom number from the set of integers { 0,1,2 } with equal probabilities.

Chapter 5

Application to Cryptography

5.1 Stream Cipher

Using the generator discuss in this thesis, we can construct a cryptosystem to encrypt stream cipher[17]. The initial state of the register (i.e. the seed) is used as a key. The value $a^{(t)}$ attained by a particular site through time then serve as a random sequence. Ciphertext C can be obtained from binary plaintext P according to $C_i = P_i$ XOR $a^{(i)}$; the plaintext can be recovered by repeating the same operation, but only if the sequence $a^{(i)}$ is known.

The security of this cryptosystem relies on the difficulty of finding the seed from time sequence of cell values. This problem is in the class NP. No systematic algorithm for its solution is currently known that takes a time less than exponential in N. No statistical regularities have been found in sequences shorter than the cycle length.

One approach to the problem of finding the key uses the near linearity of the Rule 30. The Rule 30 can be written in the alternative form $a_{i-1}^{(t-1)}$=$a_i^{(t)}$ XOR ($a_i^{(t-1)}$ OR $a_{i+1}^{(t-1)}$). Given the values of sites in two adjacent columns, this allows the values of all cells in a triangle to the left to be reconstructed, but the sequence provided gives only one column. Values in other column can be guessed, and then determined from the consistency of Boolean equations for the seed. However in disjunctive normal form the number of terms in these equations increase linearly with N, presumably making their solution take a time more than polynomial in N.

5.2 One Time Pad

Pseudorandom have been used in private key cryptosystem. A private key cryptosystem uses a key for two users exchange some secure which enables them both to encrypt and decrypt message sent between them. Ciphertext should be unreadable to anyone else and should appear "random" to unauthorized receiver, and ideally no statistical information can be extract from the ciphertext. Most private key cryptosystem do not achieve this, and statistical method is a main method of cryptanalysis.

Absolute security can always be achieved by the one time pad, which uses a key of same length as the totality of ciphertext to be exchanged. How much security is possible when the key is to be shorter than a messages to be encrypted? In this situation an analogy between pseudorandom number generation and private key cryptosystem is apparent: the key of the one time pad supplies seed for the pseudorandom number generator to generate longer one time pad.

5.3 Probabilistic Encryption

Probabilistic Encryption of single bits have been proposed to replace deterministic block encryption, since adversary cannot distinguish between a random encryption of "0" and "1".

There is a example that proposed in [18].

To send a message M to Alice using a probabilistic scheme, Bob proceeds as follows. Let $M{=}m_1 \cdot m_2 \cdots m_t$ in binary notation. For $i = 1, \cdots t$:

1. Bob randomly choose an integer r_i from $\mathbb{Z}_n$.

2. If $m_i{=}0$, Bob sends $c_i{=}r_i^2 \bmod n$ to Alice; if $m_i{=}1$, Bob sends $c_i{=}y \cdots r_i^2 \bmod n$ to Alice.

When $m_i{=}0$, Bob send a random square to Alice, whereas when $m_i{=}1$, Bob send a random pseudosquare. (Alice need to include y in her public key just so that Bob will be able to generate random pseudosquares.)

Since distinguishing squares from pseudosquare modulo n is easy if the factorization of n is known, Alice can decode the message.

However, for an adversary, the problem of distinguishing whether a given piece of cipher text c_i represents a 0 or a 1 is precisely the problem of distinguishing square from pseudosquare, which was assumed to be hard.

5.4 Probabilistic Encryption with RSA

RSA cryptosystem is a public cryptosystem that has its security depended on the computational complexity of factorizing large integers. However, for any deterministic encryption, partial information of plaintext can always be computed from ciphertext.

The following scheme have been proposed to hide all partial information bit-by bit.

Let N be a Blum integer which is the product of two primes each congruent to 3 (mod 4) and $\phi(N)$ be the Euler-phi function.

For Alice, choosing a suitable Blum integer, hides the two primes and publics the key N.

When Bob sends a message x with length n to Alice, first he chooses s_0 uniformly from $\{1, ..., N\}$. Next, for $i = 1, ..., n+1$, computes $s_i = s_{i-1}^2 \pmod N$ and $\sigma_i =$lsb(s_i). Finally, computes y=x XOR $\sigma_1\sigma_2 \cdots\cdots\cdots \sigma_n$. The ciphertext will be (s_{n+1}, y) .

After Alice receives the ciphertext, first computes $d = 2^{-n} \bmod N$. Next using this d to compute a_1 such that a_1=$s_{n+1}^d \bmod N$. Then, for $i = 1, ..., n$, computes $\varsigma_i =$lsb(a_i) and a_{i+1}=$a_i^2 \pmod N$. Finally, the plaintext is y XOR $\varsigma_1\varsigma_2 \cdots\cdots\cdots \varsigma_n$

The scheme security will depend on the length of N.

5.5 Prove yourself

The following is a scheme for a person to prove himself.[20]

Assume that Alice wants to prove that she is Alice to Bob, first Alice publics a Blum integer, and select a x_A from the set $\{1, ..., N\}$ and public $y_A{=}x_A^2$ (mod N).

Alice publics a Blum integer her public key. When Alice wants to prove herself, she uniformly select a r from the set $\{1, ..., N\}$ and sends $s{=}r^2$ mod N to Bob.

Bob uniformly select a challenge σ from $\{0, 1\}$, and sends it to the Alice.

Alice replies with $z{=}r \cdots x_A^\sigma$ (mod N) to Bob.

Bob finally accepts if and only if $z^2 \equiv s \cdots y_A^\sigma$ (mod N).

The above scheme can be repeated to maintain reliability.

The protocol depends on that Alice is the only party to know the square root of y_A.

Bibliography

[1] J. C. Lagarias, *Pseudorandom Number Generators in Cryptography and Number Theory*, Proceeding of Symposia in Applied Mathematics, Vol. **42** (1990), P.115 – P.143.

[2] D. Knuth and A. C. Yao, *The Complexity of Nonuniform Random Number Generation*, Algorithms and Complexity (J. F. Trabu,Ed.), Academic Press, P.357-428.

[3] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudorandom bits*, SIAM J. Comp. **13** (1984), P.850-864.

[4] G. Marsaglia, *A currect view of random number generators*, Proc. Computer Sci amd Statistics, 16th Sympos. on the Interface, P.1-10

[5] S. Wolfram, *Random Sequence generated by cellular automata*, Adv. Appl. Math. **7**, P.123-169.

[6] E. F. Codd, *Cellular Automata*, ACM Monograph Series.

[7] S. Wolfram, *Statistical mechanics of cellular automata*, Reviews of Modern Physics **55** (1983), P.601-644.

[8] O. Martin and Andrew M. Odlyzko, *Algebraic Properties of Cellular Automata*, Communications in Mathematical Physics. **93** (1984), P.219-258.

[9] D. Lind *Application of ergodic theory and sofic systems to cellular automata*, Physics 10D (1984) P.36-.

[10] S. Wolfram, *Universality and Complexity in Cellular Automata,* Physics 10D (1984) P.115-157.

[11] Ya. Sinai, *An answer to a question by J. Milnor*, Comment. Math. Helv. **60** (1985), P.173.

[12] M. Garey and D. Johnson, *Computer and Intractability: A Guide to the Theory of NP-completeness* W. H. Freeman, San Francisco(1979).

[13] R. Brayton, G. Hachu, c. McMullen and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis"* Kluwer, Boston(1984).

[14] R. Rudell *Espresso Software Program* Computer Science Department, University of California. Berkeley(1985).

[15] P. Grassberger, *Towards a quantitative theory of self-generated complexity* Wuppertal preprint (1986).

[16] M. Minsky, *Computation: Finite and Infinite Machines* Prentice-Hall, Eaglewood Cliffs, N.J., 1967.

[17] S. Wolfram, *Cryptography with Cellular Automata* CRYPTO'85 Proceedings: Advances in Cryptography.

[18] S. Goldwasser, *The Search for Provably Secure Cryptosystems* Proceeding of Symposia in Applied Mathematics, Vol. **42** (1990), P.89 - P.114.

[19] M. Blum and S. Goldwasser, *An Efficient Probabilistic Public-key Encryption Scheme Which Hides All Partial Information* CRYPTO'84 Proceedings: Advances in Cryptography.

[20] A. Fiat and A. Shamir, *How to Prove yourself: Practical solutions to Identification and Signature Problems* CRYPTO'86 Proceedings: Advances in Cryptography.

[21] D. Kruth, *Seminumerical Algorithms* Addison-Wesley, Reading, Mass., 1981.