5-1-2004

Study on Dynamical process in Boolean Network.

Masahiko Kimura

Study on Dynamical process in Boolean Network

A Thesis

Presented to the

Department of Mathematics

and the

Faculty of the Graduate College

University of Nebraska

In Partial Fulfillment

of the Requirements for the Degree

(Master of Art)

University of Nebraska at Omaha

by

Masahiko Kimura

May 2004

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

THESIS ACCEPTANCE

Acceptance for the faculty of the Graduate College, University of Nebraska, in partial fulfillment of the requirements for the degree (~~name the degree~~ MA), University of Nebraska at Omaha.

Committee

Valentin Matache — MATHEMATICS
Name — Department

Jarl Hedl — Mathematics
Name — Department

Hal Hartzer — Computer Science
Name — Department

Chairperson /Vyacheslav Rykov/

Date 05/06/04

ABSTRACT

Study on Dynamical process in Boolean Network

Masahiko Kimura, MA

University of Nebraska, 2004

Advisor: Vyacheslav Rykov

This thesis began with a question about how the human brain works and then two type of Boolean Networks , 1-D Cellular Automata and Associate Memory Model are chosen to explore this question. In the first part of this thesis the computability of 1-D Cellular Automata is studied by inventing a simple model, 3 colored balls model and several logical gates are implemented in its space. Although this model is so simple, by implementing three important logical gates, "NOT", "AND" and "OR", 3 colored balls model is proved to be computational universal. Since a common property " annihilation" is seen in each gate for having computability, the property is also used to prove the computational universality of Elemental Cellular Automata. In the other part of thesis the probability of errors in the recurrent associate memory model is

studied. To study dynamics of the recurrent model, I use the same MDS code as an input and an output and found the probability distribution of the number of extra ones that may appear through associate map. By the probability distribution we found that the number of error is going to decrease when the larger MDS code is used and also by comparing to binomial distribution both results are close and when we choose the larger MDS code the difference of both is getting closer.

ACKNOWLEGEMENTS

For the opportunity to work on this project, I sincerely thank my advisor Dr. Rykov and Dr. Jack Heidel. Furthermore, I would like to extend my gratitude to Dr. Heidel for insuring 2 year length research job in the Mathematical Cellular Biology Research Group. My gratitude also goes out to Dr. Valentine Matache and Dr. Heifeng of my thesis committee.

Thank to everyone who was stuck in room 243 for 2002-2003, especially, Andrew S Buchan. Special thank goes out to all other people who help me to survive in United States.

This thesis dedicated to my wife, Mika, and my daughter, Rino. I have never succeeded without your help.

Contents

1. INTRODUCTION

Regarding to the recent scientific research, many researchers are trying to clarify the complex phenomena with computer simulation. We could say that the origin of studying on complex phenomena might have gotten started by Henri Poincare who showed that there is no analytical solution for the n-body problem ($n > 2$). After that, it gradually penetrated among researchers and now became a big research subject in Mathematics, Physics and many other fields.

One of the reasons for this research in complex systems is J. Von Neumann's concept, Cellular Automata. A cellular automaton is a discrete dynamical system, i.e., space, time, and the states of the system are discrete. Each point in a regular spatial lattice, called a cell, can have any one of a finite number of states. The states of the cells in the lattice are updated according to a local rule. That is, the state of a cell at a given time depends only on its own state one time step previously, and the states of its nearby neighbors at the previous time step. All cells on the lattice are updated synchronously. Thus the state of the entire lattice advances in discrete time steps. Although this model sounds like very simple, its dynamics is diverse and it is not as

simple as we can imagine.

Although Cellular Automata is studied mainly from a physical point of view I will discuss the computational universality of one-dimension elemental cellular automata, which S.Wolfram introduced. Since the elemental cellular automata is one of the simplest cellular automata, if this model could be the computational universal, this tells us that the computation, which we are doing inside of our brain, could be a simple and repetitive procedure of thousands of neurons like Cellular Automata does. Because of that, we try to prove computational universality for some Cellular Automata models.

On the order hand, the research on brain is also one of the biggest research topics on the complex phenomena. In the last decade of 20 century, there was a big project to clarify the function of human brain with biological, physical, mathematical and computational point of views in the U.S.A and also in Japan, the Ministry of Education, Science and Culture declared that the 21 century would be a Brain century and give high priority to brain research.

In 1970, Palm [5] introduced an associative memory model in which all input vectors are connected to output vectors through an associative map. Since Boolean sum is applied to the associative map, we can see that this network model is one of the Boolean network models. After Palm's introduction of his model V.V.Rykov, A.G.D'yachkov [6] have done a mathematical analysis using coding theory, i.e., superimposed code and got several theoretical results.

In another part of this Master thesis, we try to continue A.G.Dyachkov and V.V.Rykov's work on Palm's associative memory model. Since they have focused on the model with (s,t)-superimposed code for perfect recognition, however, in

common-sense we get sometimes failure recognitions. In terms of Palm's model, the failure of a simulation is based on the use of the (p,t)-superimposed code, $p > s$, as a question code. Hence, we figure out the behavior of Palm's model with (p,t)-superimposed code and we will introduce some formulas to calculate the probability of reproduction when we use an MDS code as an input and an output code .

2. PHYSICS LIKE COMPUTAION.

2.1 Review.

2.1.1 Cellular Automata.

a. Historical Review of Cellular Automata

Cellular Automata (CA for short) theory was first introduced by the Hungarian Mathematician J.Neumann. Before finding CA theory he was interested in Finite State Automaton (FSA) theory because he wanted to apply this theory as a mathematical basis for computer architecture design. However, the founder Neumann himself was unsure that this idea could be extend and applied to other fields. Nevertheless, it opened a new field, namely the study of so-called discrete dynamical systems.

From 1960 to 1970 FSA and CA were studied just for the design of a circuit in a computer or a tool of searching on Turing Machines. But the introduction of the "game of life" invented by J.H.Conway in a science magazine Oct. 1970 provided a turning point. In J.H.Conway's game of life there is a rule with which some

configuration is going to spread out on a cell space. What is so interesting for many people was to find an initial configuration leading to a final expected configuration while observing some cell patterns appearing and disappearing on the display. It was probably one of the first computer games in human history.

In the 1980's, when interest in the game of life was being realized, S.Wolfram pointed out the correspondence between CA and differential equation. In 1983, although at that time, the center of the CA research was 2-D CA such as game of life and others, he introduced new concept i.e., 1-D CA and furthermore, he insisted that CA could be one of the basic model of discrete dynamical system. In 1984 he published a famous paper among CA researchers in which he claims that there exists only four classes in CA; class 1 corresponded to linear system, class 2 corresponded to cyclic system, class 3 corresponded to chaos, class 4 corresponded to any other none-linear systems.

He also studied 2-D CA with N.Packard and together they conjectured that 2-D CA theory could be applied as a model of self-organization in many physical system. More importantly, he proved in 1986 that Lattice Gas Automaton (LGA) that is a branch from CA theory is equivalent to Navier-Stokes equation at macro level. Because of this, the application of CA theory to fluid dynamics was spotlighted.

b. Fundamental principle of CA theory

The fundamental features of CA theory are the followings.[2]

- Assume that we have some uniform cells in a space.
- Every cell can have two or more states.
- The state of each cell is decided by a local rule using states of its neighbors.
- The final pattern on a cell space is decided by the initial condition and the rule.

In 1-D CA we assume that we have just a line of cells and set up the initial condition and a rule for dynamics, then we can observe the dynamics of the cell space as time progresses. The general form of 1-D CA is described with the cell location i, time step t, and the state of each cell takes just one value out of $k \in Z^+$ numbers of the state value. Express a_t^i as the state of a cell with location i and the time step t. Determining state $t+1$ is achieved by considering the neighboring states a_t^{i-r} and a_t^{i+r} and a local rule for $0 < i$, $r < \infty$ where $i \in Z, r \in Z^+$

$$a_{t+1}^i = F(a_t^{i-r}, a_t^{i-r+1}, \cdots a_t^i, \cdots, a_t^{i+r-1}, a_t^{i+r}).$$

Consider hereby the case of $r = 1$ where each cell can have two states 0 or 1. In this case, there is 8 possible arrangements of cells at time t and for the subsequent time steps it could takes two possible states, then there exist $2^8 = 256$ possible rules. When $r = 1$, S.Wolfram calls it "Elemental Rule". Generally, there exists k^n possible rules ($n = k^{2n+1}$).

He found that although different initial configurations make space-time patterns which differ in detail of their appearance, the characteristics of a given pattern appear unchanged. That is all 1-D Cellular Automata rules evolving from disordered initial condition reach to one of only four basic behavioral classes. The followings are the four behavior classes.

Class c1: *All site eventually attain the same value.*

Class c2: *Simple stable states or periodic and separated structures emerge.*

Class c3: *Chaotic nonperiodic patterns are generated.*

Class c4: *Complex, localized, propagating structures are formed.*

The homogeneous final states occurring for all **c1** rules, for example, are essentially the same as a *fixed point* attracting final state. Similarly, the asymptotically periodic final states of all **c2** rules are analogous to continuous *limit cycles*, while the chaotic states generated by **c3** rules are analogous to the *strange attractors* appearing in continuous dynamical system. The more complicated localized structures emerging from **c4** class rules, on the other hand, do not appear to have any obvious continuous analogs.

2.1.2 Computing Capability of Cellular Automata.

Computational Universality implies the ability to compute any logical function. Any logical function above indicates Boolean function and the smallest set to be computational universal is $\{AND, NOT\}$ or $\{OR, NOT\}$. According to the above sentence, the personal computers we use have the ability to perform any digital computation, so they are computational universal. Instead of personal computer, there are some reports that the physical, chemical and biological systems in the real or imaginary world seem to have computing capability. This sort of computation being in nature is called " Natural computing" [3].

Computational universality is classified into two types. One is based on its structure and another is based on moving materials. The former type, such as an ordinary computer containing electrical circuits and neural network stays the same structure as time progresses. Compared with this, the latter type is based on moving materials and is called " Collision based computing" insofar as collisions of moving materials are used for computing logical function. This type is also called " structureless computation", actually it is not necessary to use the wire to transmit information from a logic gate to others.

E.Fredkin and T.Toffoli have done works on natural computing with their Billiards Ball Model (BBM). BBM is a model that expresses the dynamics of balls in a space. Computational power of this model could be seen when arbitrary logical gates are implemented in the system. They expressed two inputs with existence of balls (0 or 1) and showed its output as the result for collisions of two inputs balls. This idea was transformed to CA theory and studied by N.Margolus. Inventing Margolus neighbor

leaded him to a success of simulating BBM as a CA model, and he proved that BBMCA is computational universal.

J.H.Conway's game of life is also one of the computational universal systems. The designer by himself found the proof of its computational universality and published a book with the proof. Similarly, BBMCA, collisions were used in his proof, however, he used gliders which are one of the patterns in game of life instead of balls.

In the mid 80's, 1-D CA having a capability of computation appeared. Since the space of 1-D CA is just a line, 1-D CA's moving materials have just two directions to propagate i.e., right or left, in place of balls and gliders moving around in 2-D space, however, it has a variety of velocity to transmit and actually these various velocities make a 1-D CA's rule possible to implement several collisions for many CA models. The moving material in 1-D CA is called "soliton". R.K.Squier and K.Steiglits created a 1-D CA model called "particle model" and they successfully simulated a binary adder and multiplier with it. A.Adamatzky [3] discussed in his book information transition of soliton found in nature.

We can find the character of computing capability above mentioned in the set of moving DNA molecules. A.Adamatzky showed some of the logic gates based on the collision of DNA molecules and found a 1-D CA model having the same ability of computation.

S.wolfram's elemental rule is supposed to be one of the most studied CA models. He discussed the computability of elemental rules [1]. He mentioned that with an analogy of dynamical system space-time patterns of 1-D CA could be divided into four classes (*Fundamental principle of CA theory* in this thesis) and one class out

of four, i.e., class 4, seems to have a computing ability. He insisted that class 4 is too complex to define with dynamical system theory and also because there is a similarity between a space-time pattern of class 4 and a complex pattern of game of life, he conjectured that all of the rules belonging to this class are computational universal. However, proving this conjecture is very hard and there is no theory that does so as of yet. Consequently, current research aims at studying properties of class 4. As the character of class 4, it is well known that when a simulation with an arbitrary initial condition gets started, soliton appeared. Some researchers are calling it glider due to an analogy of J.H.Conway's game of life.

So far many models that are computational universal have been introduced, however, we can not see what is the key to be computational universal. Hence, in this chapter we introduced an original 1-D CA model that has four states, $r=1$ and try to show that the model may be computational universal with implementing collision gate on its space. With this result we would like to mention that it is enough to be computational universal with just one property which we can see in this new CA model. And also, we try to show that two rules of elemental CA might have a computing capability with the same method used above.

2.2 *Computability of 1D-CA.*

2.2.1 Computing Capability of 4states 1-D CA.

First, we introduce a CA model that has 4 states ($k = 4$) and $r = 1$. The function (local rule) of this model is

$$f(a_{i-1}, a_i, a_{i+1}) = \begin{cases} 1 & \text{if } a_{i-1} = 1, a_i = 0, a_{i+1} = 0 \\ 2 & \text{if } a_{i-1} = 0, a_i = 2, a_{i+1} = 0 \\ 3 & \text{if } a_{i-1} = 0, a_i = 0, a_{i+1} = 3 \\ 0 & \text{Otherwise} \end{cases}.$$

Since 3 colored balls appeared in the space-time pattern, I will call it 3 colored balls model (*3CBM*). The following figures represent 3 types of particles appearing in this model, though they are expressed numbers, rather than colors.

0 1 0 0 0 0 0	t	Right particle
0 0 1 0 0 0 0	$t+1$	
0 0 0 1 0 0 0	$t+2$	
0 0 0 0 1 0 0	$t+3$	
0 0 0 2 0 0 0	t	Stable particle
0 0 0 2 0 0 0	$t+1$	
0 0 0 2 0 0 0	$t+2$	
0 0 0 2 0 0 0	$t+3$	
0 0 0 0 0 3 0	t	Left particle
0 0 0 0 3 0 0	$t+1$	
0 0 0 3 0 0 0	$t+2$	
0 0 3 0 0 0 0	$t+3$	

Fig.2.1.1 Moving particles of this CA.

Fig. 2.1.2 depicts two types of collision implemented by two particles, right direction and left direction. Fig. 1.1.3 shows collisions between stable particle and angled particles.

0 1 0 0 0 3 0	t	Odd collision
0 0 1 0 3 0 0	$t+1$	
0 0 0 0 0 0 0	$t+2$	
0 0 0 0 0 0 0	$t+3$	
0 0 1 0 0 3 0	t	Even collision
0 0 0 1 3 0 0	$t+1$	
0 0 0 0 0 0 0	$t+2$	
0 0 0 0 0 0 0	$t+3$	

Fig. 2.1.2 two types of collision with two particles.

0	1	0	2	0	0	0	t	0	1	0	2	0	3	0	t
0	0	1	2	0	0	0	$t+1$	0	0	1	2	3	0	0	$t+1$
0	0	0	0	0	0	0	$t+2$	0	0	0	0	0	0	0	$t+2$
0	0	0	0	0	0	0	$t+3$	0	0	0	0	0	0	0	$t+3$

Fig. 2.1.3 two collisions between stable particle and two angled particles

Proposition 2.1.1. One can implement NOT gate ($u \to \bar{u}$; $u \in \{0,1\}$) with two moving particles of 3CBM.

One of the configurations to compute NOT is $\cdots 0Z00X000300\cdots, X \in \{1,0\}$, $Z = \{0,3\}$ ($Z = 0$ when $t = 0$) where X is an input and Z is an output. The transformations are the followings

$$u(X) = \begin{cases} 1, & \text{if } X > 0 \\ 0, & \text{Otherwise} \end{cases}$$

$$\bar{u}(Z) = \begin{cases} 1, & \text{if } Z > 0 \\ 0, & \text{Otherwise} \end{cases}$$

The result of this computation in seven steps is as follows

$u(X)$	$\bar{u}(Z)$
0	1
1	0

Table 2.1.1 NOT gate realized by collision gate.

$$
\begin{array}{cccccccccccccc}
0 & Z & 0 & 0 & X=1 & 0 & 0 & 0 & 3 & 0 & 0 & & t=0 \\
 & Z & & & & 1 & & 3 & & & & & t=1 \\
 & Z & & & & & 0 & & & & & & t=2 \\
 & Z & & & & 0 & & & & & & & t=3 \\
 & Z & & & 0 & & & & & & & & t=4 \\
 & Z & & 0 & & & & & & 0 & & & t=5 \\
 & Z & 0 & & & & & & & & & & t=6 \\
 & Z=0 & & & & & & & & & & & t=7
\end{array}
$$

Fig. 2.1.4 Example of NOT gate with $u(X)=1$, the result $\bar{u}(Z)=0$.

Proposition 2.1.2. One can implement OR gate ($(u,v)\to u\vee v$; $u\in\{0,1\}$, $v\in\{0,1\}$) with two moving particles of 3CBM.

One of the constructions to compute OR is $\cdots$01000X0002000Y0000Z0$\cdots$, $X\in\{0,1\}, Y\in\{0,3\}$ and $Z\in\{0,1\}$ ($Z=0$ when $t=0$) where X and Y are inputs and Z is an output. The transformations are

$$u(X)=\begin{cases}1, & \text{if } X>0\\ 0, & \text{Otherwise}\end{cases}$$

$$v(Y)=\begin{cases}1, & \text{if } Y>0\\ 0, & \text{Otherwise}\end{cases}$$

$$u\vee v(Z)=\begin{cases}1, & \text{if } Z>0\\ 0, & \text{Otherwise}\end{cases}$$

The result of this computation is

$u(X)$	$v(Y)$	$u \vee v(Z)$
0	0	0
0	1	1
1	0	1
1	1	1

Table 2.1.2 OR gate realized by collision gate.

Proposition 2.1.3. One can implement AND gate ($(u,v) \to u \wedge v$; $u \in \{0,1\}$, $v \in \{0,1\}$) with two moving particles of 3CBM.

One of the constructions to compute AND is$\cdots$01000X000Y0020003000Z0$\cdots$, $X \in \{0,1\}, Y \in \{0,1\}$ and $Z \in \{0,1\}$ ($Z = 0$ when $t = 0$) where X and Y are inputs and Z is an output. The transformations are

$$u(X) = \begin{cases} 1, & \text{if } X > 0 \\ 0, & \text{Otherwise} \end{cases}$$

$$v(Y) = \begin{cases} 1, & \text{if } Y > 0 \\ 0, & \text{Otherwise} \end{cases}$$

$$u \wedge v(Z) = \begin{cases} 1, & \text{if } Z > 0 \\ 0, & \text{Otherwise} \end{cases}$$

The result of this computation is

$u(X)$	$v(Y)$	$u \wedge v(Z)$
0	0	0
0	1	0
1	0	0
1	1	1

Table 2.1.3 AND gate realized by collision gate.

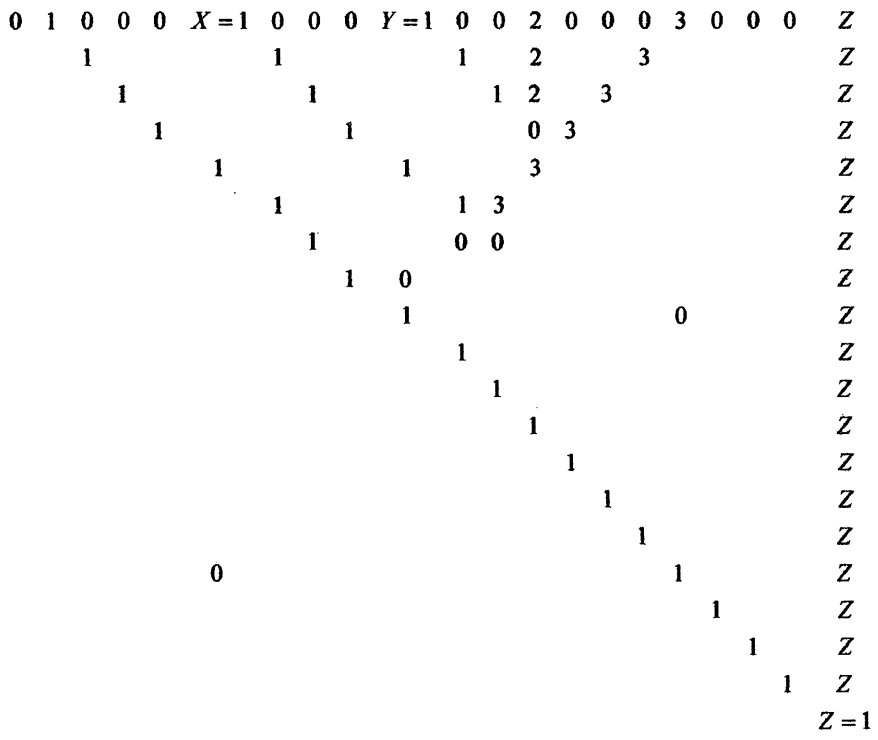


Fig. 2.1.5 Example of AND gate with $u(X)=1$ and $v(Y)=1$,the result $u \vee v(Z)=1$.

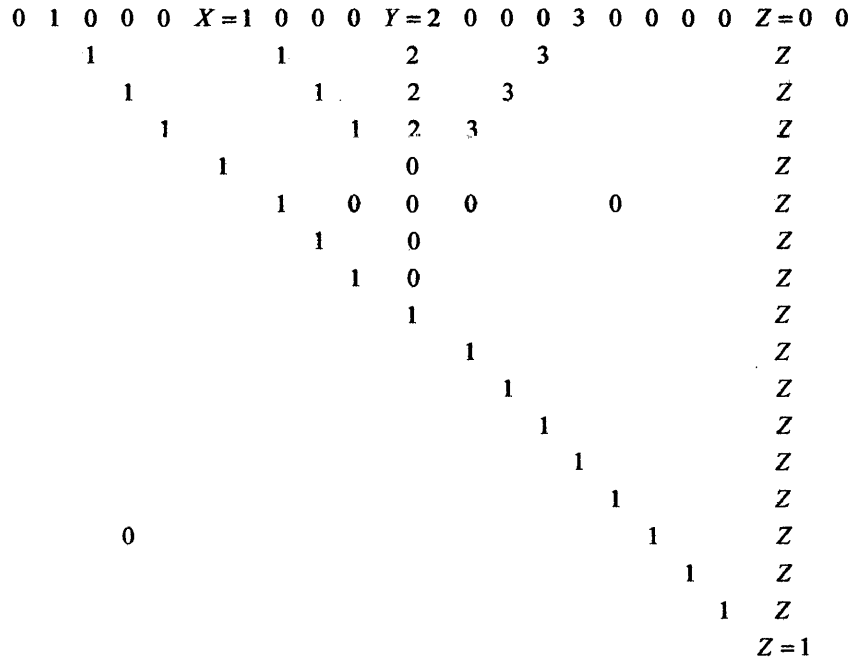


Fig. 2.1.6 Example of OR gate with $u(X)=1$ and $v(Y)=1$,the result $u \wedge v(Z)=1$.

Theorem 2.1.4. 3CBM is computational universal.

Proof: Since we can implement {NOT, AND, OR} gate with 3CBM, hence it is computational universal.

Example 2.1.5. To show computation of binary numbers with the CA model. I will give one example that is a binary adder $x_1x_0 + y_1y_0 = z_2z_1z_0$. Each binary number z_2, z_1, z_0 is described respectively by following equation ($\oplus$ expresses XOR, Exclusive OR)

$$w_0 = u_0 \oplus v_0,$$

$$w_1 = (u_1 \oplus v_1) \oplus (u_0 \wedge v_0),$$

$$w_2 = (u_1 \wedge v_1) \vee ((u_1 \oplus v_1) \wedge (u_0 \wedge v_0)),$$

with the following transformations

$$u_i(x_i) = \begin{cases} 1, & \text{if } x_i > 0 \\ 0, & \text{Otherwise} \end{cases}$$

$$v_i(y_i) = \begin{cases} 1, & \text{if } y_i > 0 \\ 0, & \text{Otherwise} \end{cases}$$

$$w_j(z_j) = \begin{cases} 1, & \text{if } z_j > 0 \\ 0, & \text{Otherwise} \end{cases}$$

where $i \in \{0,1\}$ and $j \in \{0,1,2\}$.

Since XOR is used in the above computation, I will show the construction of the XOR gate with CA Model before we introduce the construction for the computation.

Proposition 2.1.6. One can construct XOR gate ($(u,v) \to u \oplus v$; $u \in \{0,1\}$, $v \in \{0,1\}$) with two moving particles of this CA model.

A construction to compute XOR is $\cdots 0Z000X00Y00200030 \cdots$, $X \in \{0,1\}, Y \in \{0,1\}$ and $Z \in \{0,3\}$ ($Z = 0$ when $t = 0$) where X and Y are inputs and Z is an output. The transformations are

$$u(X) = \begin{cases} 1, & \text{if } X > 0 \\ 0, & \text{Otherwise} \end{cases}$$

$$v(Y)=\begin{cases}1, & \text{if } Y>0\\ 0, & \text{Otherwise}\end{cases}$$

$$u\oplus v(Z)=\begin{cases}1, & \text{if } Z=3\\ 0, & \text{Otherwise}\end{cases}$$

The result of this computation is below.

$u(X)$	$v(Y)$	$u\oplus v(Z)$
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.1.4 XOR gate realized by collision gate.

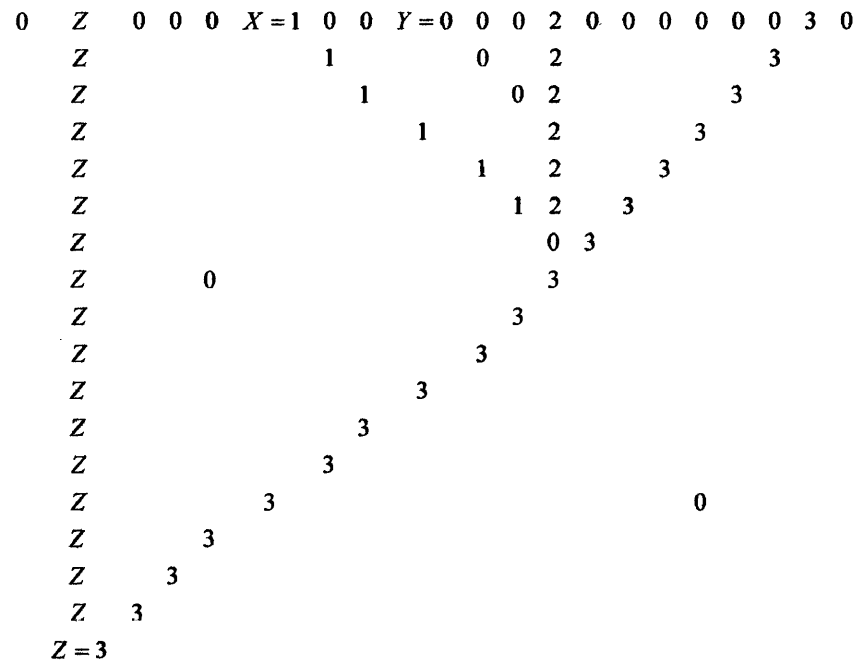


Fig. 2.1.7 Example of XOR gate with $u(X)=1$ and $v(Y)=0$,the result $u\oplus v(Z)=1$.

Now we can construct Binary adder with following configurations.

For $z_0 = x_0 \oplus y_0$, the construction is

$$\cdots 0z_0 00x_0 00y_0 00200030 \cdots .$$

which is the same as XOR gate.

For $z_1 = (x_1 \oplus y_1) \oplus (x_0 \wedge y_0)$, the construction is

$$\cdots 0z_1 100020x_1 0y_1 0100x_0 00y_0 00200030200000000000000000000000003 \cdots .$$

For $z_2 = (x_1 \wedge y_1) \vee ((x_1 \oplus y_1) \wedge (x_0 \wedge y_0))$, the construction is

$$\cdots 10\overbrace{10000000000000000000000x_1 00y_1 00200300020}^{x_1 \wedge y_1}\underbrace{1000200\overbrace{x_1 00y_1 00200003}^{x_1 \oplus y_1}01000200\overbrace{x_0 00y_0 0030030}^{x_0 \wedge y_0}}_{(x_1 \oplus y_1) \wedge (x_0 \wedge y_0)} z_2 \cdots$$

2.2.2 Computing Capability of Elemental CA.

It has been discussed whether there exists computational universal CA in Wolfram's elemental rule since he found 4 classes in 1-D CA . Especially, computational universal of R110 is the center of the argument because of its dynamics.

S.Wolfram [1] tried to show us that R110 is computational universal by showing similarity to cyclic tag system that has been shown to be universal. However, it is not clear and still, many researchers do not agree with his explanation. Therefore, in this section we try to apply the collision gate method to elemental CA.

As we saw in section 2.2.1, A CA model could be computational universal. If we want to use the same method for elemental CA, what kind of property is needed to be computational universal? The common property that appears in 3CBM is annihilation. Namely, as we see in Fig. 2.1.1 and 2.1.2, all particles annihilate after the collision. Hence, if we could find particles that annihilate after collision in a rule of elemental CA, the rule may be computational universal with constructing collision gates.

In R110 space-time pattern I will use a background to show moving materials. The background is 14 period cycle on which materials are moving. Fig. 2.2.2 shows that all materials transmitting in space-time pattern of R110. We found a material that is shown Fig. 2.2.3 that annihilates when two of them collide so then we will use it to construct collision gates.

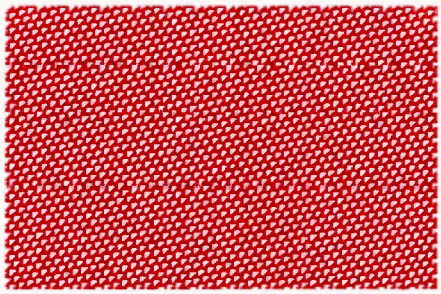


Fig. 2.2.1 Background of R110 that is 14 periods cycle of {11111000100110}.

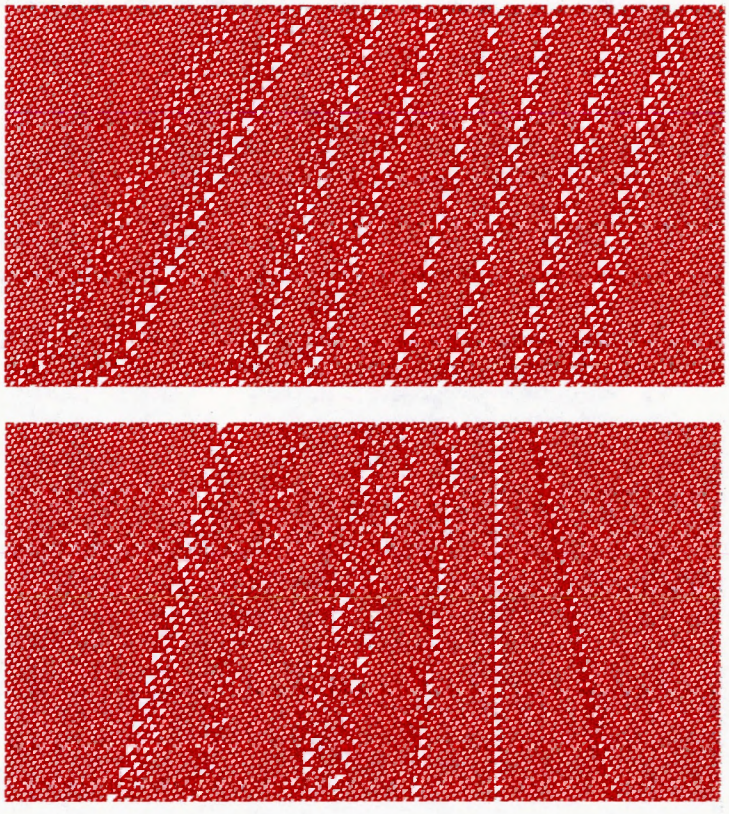


Fig. 2.2.2 All moving material in R110 space-time pattern.

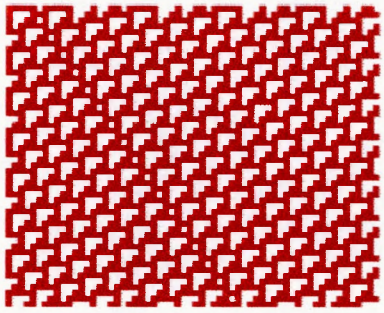


Fig. 2.2.3 A moving material in R110 space-time pattern that is annihilate when two of them collide.

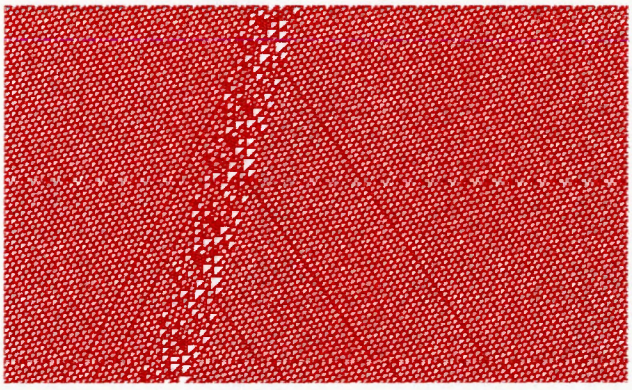


Fig. 2.2.4 A generator of moving material.

Along with moving materials we can easily construct " NOT gate" that is shown in Fig. 2.2.5, however, when we make " OR gate" a problem arises, i.e., "a presence of a stopper". In the Game of life, if one follows the way Rennard [3] did, one can

construct " AND " and "OR" with a stopper that dose not move at all. The stopper works when there exist some useless moving particles. In the case of R110, if one tries to construct "OR gate", it is necessity to use a stopper because with inputs A=1 and B=1, either of the two inputs goes to be useless moving particles. This tells us that for more complicated computation one need to use several stoppers. Besides, if we allow using it, we must know the position of it in the initial configuration, which means we must know all computational procedures before it runs. Can we say that a 1-D rule is universal with using such a stopper?

We considered that the universality with using stopper is weaker than that of 3CBM, however, we may say that it is conditionally computational universal.

In addition to that, we realized that if we admit the conditional universality with stoppers, we can say that all rules that have one or two defects moving to right and also left have the conditional power of computation, For example, R184 that is sometime used for "traffic model".

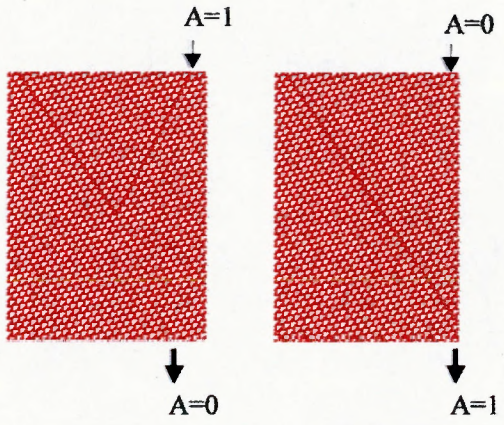


Fig. 2.2.6 NOT gate of R110

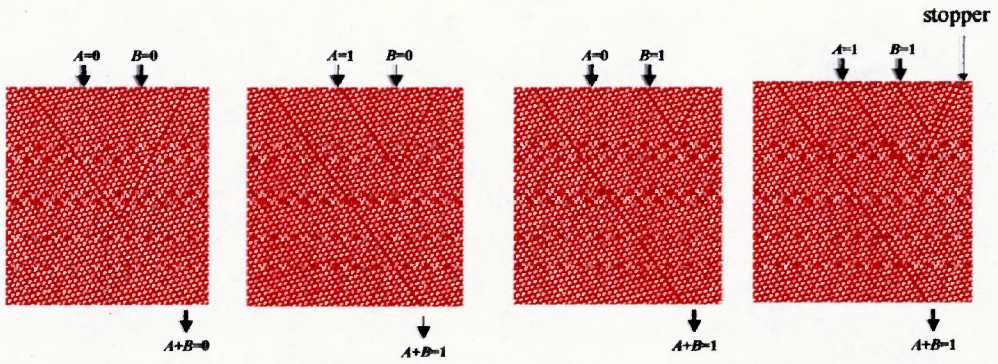


Fig. 2.2.7 OR gate of R110

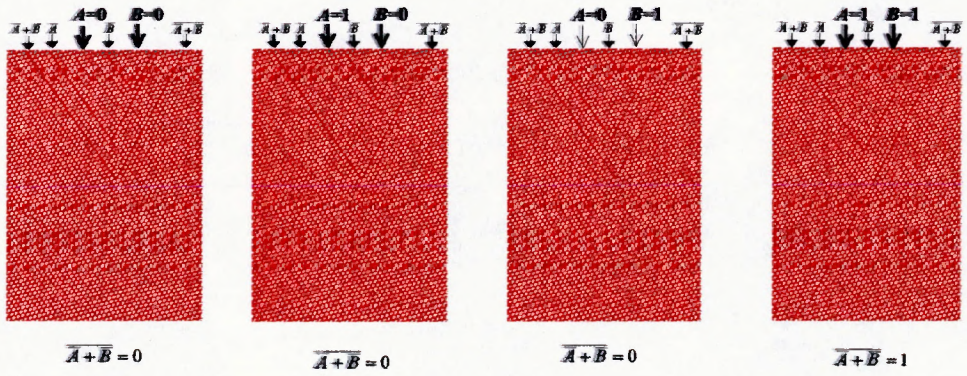
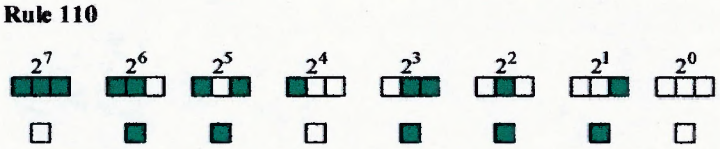


Fig. 2.2.8 AND gate of R110 constructed by the combination of NOT and OR gate.



$$2^6 + 2^5 + 2^3 + 2^2 + 2^1 = 110$$

Fig. 2.2.9 Elemental Rule 110.

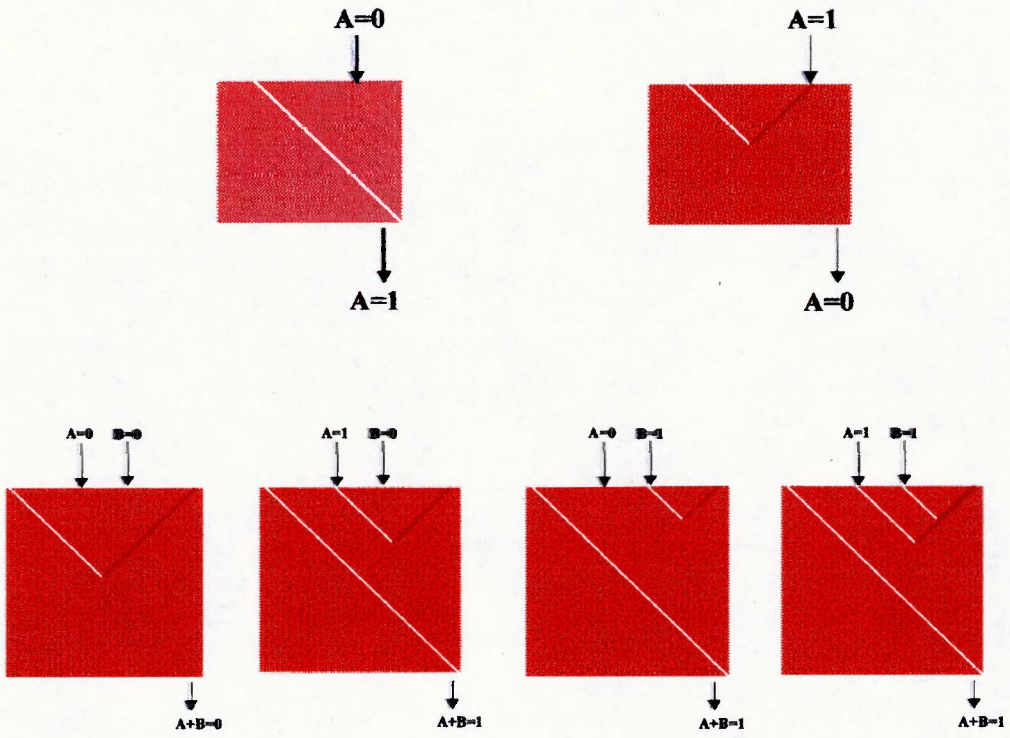


Fig. 2.2.10 NOT and OR gate of R184.

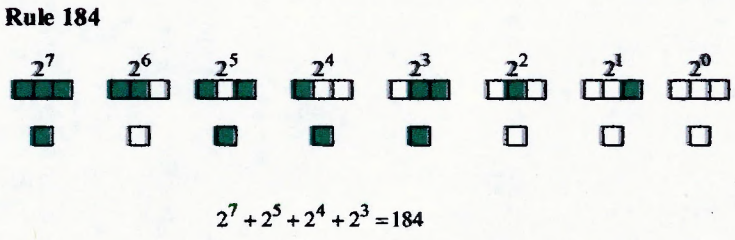


Fig. 2.2.11 Elemental Rule 184.

3. MATHEMATICAL ANALYSIS ON NEURAL NEWORK WITH CODING THEORY

3.1 Review.

In this chapter we study a neural network model with superimposed code theory. First, we will introduce definitions and how to construct Superimposed Code with some examples. Secondly, an explanation of Associative Memory model is given.

3.1.1 Superimposed Code.

a. Notation and Definition.

We use the terminology of combinatorial coding theory and the following collection of notations [7],[8]. Let

- $1 < s < t$, $1 \le k < t, N > 1$ be integers;
- t – code size, N – code length;
- X is $N \times t$ matrix of a code;

- $\mathbf{x}(u) = (x_1(u), x_2(u), ..., x_N(u))$, $u = 1,2,...,t$, be columns of X (codewords), and $\mathbf{x}_i = (x_i(1), x_i(2), ..., x_i(t))$, $i = 1,2,...,N$, be rows;
- $w = \min_u \sum_{i=1}^{N} x_i(u)$ be the minimal weight of codewords, $x_i(u) \in \{0,1\}$;
- $\lambda = \max_{u,v} \sum_{i=1}^{N} x_i(u)x_i(v)$ be the maximal dot product of codewords, $x_i(u) \in \{0,1\}$;
- $k = \max_i \sum_{u=1}^{t} x_i(u)$ be the maximal weight of rows, $x_i(u) \in \{0,1\}$;
- $\lceil a \rceil$ denote the least integer $\geq a$ and $\lfloor b \rfloor$ denote the largest integer $\leq b$;
- $\stackrel{\text{def}}{=}$ denote the equation by definition.

Let $\mathbf{x}(j) = (x_1(j), x_2(j), ..., x_N(j))$, $j = 1,2,...,s$ denote the binary columns of length N. The Boolean sum

$$\mathbf{x} = \bigvee_{j=1}^{s} \mathbf{x}(j) = \mathbf{x}(1) \vee \mathbf{x}(2) \vee ... \vee \mathbf{x}(s)$$

of column $\mathbf{x}(1), \mathbf{x}(2), ..., \mathbf{x}(s)$ is the binary column $\mathbf{x} = (x_1, x_2, ...x_N)$ with components

$$\mathbf{x}_i = \begin{cases} 0, & \textit{if} \quad x_i(1) = x_i(2) = \cdots x_i(s) = 0 \\ 1 & \textit{otherwise.} \end{cases}$$

Let us say that column $\mathbf{x}$ covers column $\mathbf{y}$ if $\mathbf{x} \vee \mathbf{y} = \mathbf{x}$.

The following is the definition of superimposed code in general.

Definition 3.1.1. [11] An $N \times t$ – matrix X is called a *superimposed code* (SC) *of length* N, *size* t, *strength* s if the Boolean sum of any s – subset of codewords X cannot cover codewords that are not components of the s – subset. This code also can be called an (s,t,N) – superimposed code.

Definition 1 is equivalent to the following condition. The Boolean sum of any s – subset of columns X covers those and only those columns that are the components of given Boolean sum.

The next definition is for a subset of superimposed code that is restricted with the constant weight k in a row and it is used for analyzing the dynamics of a neural network.

Definition 3.1.2. [8] An $N \times t$ – matrix X is called a *superimposed* (s,t,k) – *code of length* N, *size* t, *strength* s and constant k if code X is a superimposed (s,t) – code whose maximal row weight is equal to k.

b. Lower Bound of Superimposed Code.

We introduce a result to determine the size of superimposed code [8].

Proposition 3.1.3. [8] Let $t > k \geq s \geq 1$ and $N > 1$ be integers. For any superimposed $(s,t,k)-$code of length N, the following inequality holds:

$$N \geq \left\lceil \frac{(s+1)t}{k} \right\rceil .$$

Proof: Let $s \geq 1, 1 \leq k < t$ be fixed integers. Consider an arbitrary superimposed $(s,t,k)-$code X of length N. Let n, $0 \leq n \leq t$, be the number of codewords of X having a weight s. From the definition of superimposed $(s,t)-$code it follows that $n \leq N$ and, for each codeword of weight $\leq s$, there exists a row in which all the remaining elements, except of this codeword, are 0's. We delete these n rows from X together with n codewords of weight $\leq s$. Consider the remaining $(N-n)\times(t-n)$ matrix X'. Obviously, each column of X' has a weight $\geq s$ and each its row contains $\leq k$ 1's. Since $k \geq s$, we have

$$(s+1)(t-n) \leq k(N-n), \quad t(s+1) \leq kN - n(k-s) \leq kN .$$

□

c. Construction of Superimposed Code.

Kautz-Singleton–Superimposed Codes.

Kautz and Singleton [11] introduced this superimposed codes and A.G.Dyachkov, V.V.Rykov [10] formulated the important sufficient condition below as theorem 1.

Theorem 3.1.4. [11] Let X be a constant-weight code, i.e., X is a binary $N \times t$ matrix, whose columns (codewords) $\mathbf{x}(j)$ have the same number of 1's

$$w = \sum_{i=1}^{N} x_i(j), \quad j = 1,2,...,t, \qquad \lambda \overset{\text{def}}{=} \max_{k \neq j} \sum_{i=1}^{N} x_i(k) x_i(j)$$

be the maximal correlation of codewords. Then the matrix X is $(s, N, t)-$code for any s, satisfying the inequality

$$s \leq \left\lfloor \frac{w-1}{\lambda} \right\rfloor.$$

This theorem is easy to understand because if $w \geq s\lambda + 1$, then the ith codeword cannot possible be contained in the sum of any s other codeword, since it overlaps each of these other codewords in no more than λ positions. Thus, this code X satisfies above condition.

Definition 3.1.5. [10] Let $1 \leq \lambda \leq w \leq N$ be given integers, and let X be a code of size t, length N with parameters w and λ. A code X will be called a Kautz-Singleton –superimposed code ($KS-$*superimposed* code) of length N, size

t and strength s, if inequality $s \le \left\lceil \frac{w-1}{\lambda} \right\rceil$ holds. This code also will be called an (s, N, t) – KS-code.

c-1 KS-Superimposed Code Based on Latin Square.

Next we introduce a way to construct KS-superimposed code from well-known Latin square. The definition of Latin square is following [4].

Definition 3.1.6. A Latin square of order q is a $q \times q$ array whose entries are from a set of q distinct symbols such that each row and each column of the array contains each symbol exactly once.

A Procedure for constructing a code from a Latin square is simple. The code matrix consist of q^2 number of columns containing three elements i.e., row, column and its number in the Latin square.

Example 3.1.7. Let $q = 3$. We produce $(2,9,9)$ – KS superimposed code. Since a codeword in this code has at most one element that is the same element in other codewords, so $s = \left\lceil \frac{3-1}{1} \right\rceil = 2$.

$$\begin{matrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{matrix} \quad \Leftrightarrow \quad \begin{pmatrix} 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 \\ 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 \\ 1 & 2 & 3 & 2 & 3 & 1 & 3 & 1 & 2 \end{pmatrix}$$

so

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

c-2 KS-Superimposed Code Based on MDS Code.

A q-ary code C is a given set of sequence of symbols where each symbol is chosen from a set $F_q = \{0,1,...,q-1\}$ of q distinct elements. The set F_q is called the alphabet and is often taken to be the set Z_q of integers *mod* q if $q = p$ (p is a prime number).

Distance of Code.

Definition 3.1.8. [4] The Hamming distance between two vectors $x(u)$ and $x(v)$ is the number of places in which they differ.

It is denoted by $d(x(u),x(u))$. An important parameter of the code C is the minimum distance, denoted $d(c)$, which is defined to be the smallest of the

distances between distinct codewords. That is,

$$d(c) = \min\{d(x(u), x(v)) : u \neq v\} \ .$$

An $(n,t,d)-code$ is a code of length n, containing t codewords and having minimum distance d.

Linear Code.

Let alphabet F_q is the Galois field $GF(q)$ (set of elements with two operation + (addition) and $\bullet$ (*multiplication*), with 0 and 1, where any equation of the form $ax+b=c$ has a solution.), q is a prime power, and we regard $(F_q)^n$ as the vector space $V(n,q)$.

A linear code over $GF(q)$ is just a subspace of $V(n,q)$, for some positive integer n. If C is an h-dimensional subspace of $V(n,q)$ then the linear code C is called an $[n,h,d]-$code.

A generating set of C which is also linearly independent is called a *basis* of C and An $h \times n$ matrix G whose rows form a basis of a linear $[n,h,d]-$code is called a generator matrix of the code.

A parity –check matrix H for an $[n,h,d]-$code C is an $(n-h)\times n$ matrix satisfying $GH^T = 0$, where H^T denotes the transpose of H and 0 is the null matrix.

$$C = \{x \in V(n,q) : xH^T = 0\}$$

Definition 3.1.9. [4] An $[n,h,n-h+1]-$ code (i.e., a linear code of minimum distance $d=n-k+1$) is called *maximum distance separable* code or MDS code for short.

Before introducing the theorem for the parity-check matrix of MDS code, we will give three important results from theory of linear code [4].

Theorem 3.1.10. [4] Suppose C is a linear $[n,k]-$ code over $GF(q)$ with parity-check matrix H. Then the minimum distance of C is d if and only if any $d-1$ columns of H are linearly independent but some d columns are linearly dependent.

Theorem 3.1.11. [4] Suppose $a_1,a_2,...,a_r$ are distinct non-zero elements of a field. Then the so-called Vandermonde matrix

$$A=\begin{bmatrix} 1 & 1 & \cdots & 1 \\ a_1 & a_2 & \cdots & a_r \\ a_1^2 & a_2^2 & \cdots & a_r^2 \\ \vdots & \vdots & & \vdots \\ a_1^{r-1} & a_2^{r-1} & \cdots & a_r^{r-1} \end{bmatrix}$$

has a non-zero determinant.

Theorem 3.1.12. If A is a $(r\times r)$ matrix having a non-zero determinant, then the r columns of A are linearly independent.

The next theorem is for parity-check matrix of an MDS code[4].

Theorem 3.1.13. Suppose $1 \le h < n \le q+1$. Let $a_1, a_2, ..., a_{q-1}$ be the non-zero elements of $GF(q)$. Then the matrix

$$H = \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 & 0 \\ a_1 & a_2 & \cdots & a_{q-1} & 0 & 0 \\ a_1^2 & a_2^2 & \cdots & a_{q-1}^2 & \vdots & \vdots \\ \vdots & \vdots & & \vdots & 0 & 0 \\ a_1^{n-h-1} & a_2^{n-h-1} & \cdots & a_{q-1}^{n-h-1} & 0 & 1 \end{bmatrix}$$

is the parity-check matrix of an MDS $[n, h, n-h+1]-$code, where $n = q+1$.

Example 3.1.14. Let $q=3$, $h=2$, then $F_3 = \{0,1,2\}$. Hence we get parity-check matrix of $[3,2,2]-$MDS code which has $3^2 = 9$ codewords.

$$H = [1 \quad 1 \quad 1]$$

Hence, all codewords of his $[3,2,2]-$MDS code are

$$C = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\ 0 & 2 & 1 & 2 & 1 & 0 & 1 & 0 & 2 \end{pmatrix}.$$

To change MDS code to superimposed code, each symbol of the q-ary alphabet $\{0,1,...,q-1\}$ is substituted for the corresponding binary column of length q and the

weight 1, namely:

$$0 \Leftrightarrow \underbrace{\{1,0,0,\ldots,0\}}_{q},\ 1 \Leftrightarrow \underbrace{\{0,1,0,\ldots,0\}}_{q},\ 2 \Leftrightarrow \underbrace{\{0,0,1,\ldots,0\}}_{q},\ \cdots, q \Leftrightarrow \underbrace{\{0,0,0,\ldots,1\}}_{q}$$

Hence, we have

$$\widetilde{C} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Example 3.1.15. Let $q=7$, $h=3$, then $F_7 = \{0,1,...,6\}$. So we get parity-check matrix of $[7,3,5]-$MDS code which has $7^3 = 343$ codewords.

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & 1^2 & 2^2 & 3^2 & 4^2 & 5^2 & 6^2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & 1 & 4 & 2 & 2 & 4 & 1 \end{bmatrix}.$$

3.1.2 Combinatorial Model of Associative Memory.

a. Palm's Model and Superimposed Code.

Definition 3.1.16. [6] The *question code* of length n for memory of size t is defined by a $n \times t$ binary (0 and 1) matrix $\mathbf{X} = \|x_j(u)\|, j = \overline{1,n}, u = \overline{1,t}$. The column $\mathbf{x}(u) = (x_1(u), x_2(u), ..., x_n(u))$ of $\mathbf{X}$ is called the codeword of question u. All the codewords are assumed to be distinct.

Definition 3.1.17. [6] The *answer code* $\mathbf{Y} = \|y_i(u)\|, i = \overline{1,m}, u = \overline{1,t}$ is a $m \times t$ $0-1$ matrix. The codeword for answer u is the column $\mathbf{y}(u) = (y_1(u), y_2(u), ..., y_m(u))$. As in the question code, the answer codewords are all distinct.

Let $[t]$ be the set of integers from 1 to t, and $\mathbf{p} = (p_1, p_2, ..., p_t), p_u \in [t], p_u \neq p_v$, be a given permutation of the element of $[t]$ (there exist $t!$ such permutation). In what follows, we interpret $\mathbf{p}$ as a 1-1 mapping of the question code $\mathbf{X}$ to the answer code $\mathbf{Y}$, namely:

$$\mathbf{x}(u) \overset{\mathbf{p}}{\Rightarrow} \mathbf{y}(p_u), \quad u = \overline{1,t}.$$

To the mapping $\mathbf{p}$ we associate the $n \times m$ 0-1 matrix $\mathbf{A}^{\mathbf{p}} = \|a_j^{\mathbf{p}}(i)\|$, $j = \overline{1,n}$, $i = \overline{1,m}$, with elements

$$a_j^p(i) = \bigvee_{u=1}^{t}(x_j(u)y_i(p_u)),$$

i.e., $a_j^p(i) = 1$ if and only if at least for one $u = \overline{1,t}$ we have $x_j(u) = y_i(p_u) = 1$. The matrix $\mathbf{A}^p$ is called the *associative net* [6] for storing the mapping $\mathbf{p}$.

Let $\mathbf{a}^p(i) = (a_1^p(i), a_2^p(i),..., a_n^p(i))$ be the $i-$th column of the matrix $\mathbf{A}^p$ and $\mathbf{x} = (x_1, x_2,..., x_n)$, $\mathbf{y} = (y_1, y_2,..., y_n)$ binary columns of length n and m respectively. Using the symbol $\vee$ to denote the componentwise Boolean sum of 0-1 columns, we define in terms of the matrix $\mathbf{A}^p$ the mapping

$$\mathbf{y} = \mathbf{A}^p\mathbf{x}, \quad y_i = \mathbf{A}_i^p\mathbf{x},$$

where

$$\mathbf{A}_i^p(\mathbf{x}) = \begin{cases} \mathbf{1}, & \textit{if} \quad \mathbf{x} \vee \mathbf{a}^p(i) = \mathbf{a}^p(i), \\ \mathbf{0}, & \textit{if} \quad \mathbf{x} \vee \mathbf{a}^p(i) \neq \mathbf{a}^p(i). \end{cases}$$

Definition 3.1.18. [6] The pair of codes $(\mathbf{X}, \mathbf{Y})$ is said to have the retrieving property of stored information of size t if for any mapping $\mathbf{p}$ of the code $\mathbf{X}$ to the code $\mathbf{Y}$ is representable in the form

$$\mathbf{y}(p_u) = \mathbf{A}^p\mathbf{x}(u),$$

$$u = \overline{1,t}.$$

Theorem 3.1.19. [6] If s is the maximum number of ones in a row of the answer code $\mathbf{Y}$, then the code pair $(\mathbf{X},\mathbf{Y})$ has the reproduction property if and only if the question code $\mathbf{X}$ is a superimposed $(s,t)-$code.

Proof: $(\Rightarrow)$ Let $s_i = \sum_{u=1}^{t} y_i(u)$, $i = \overline{1,m}$ be the number of ones in the $i-$th row of the answer code $\mathbf{Y}$. Then $s = \max\ s_i$. For a given permutation $\mathbf{p}$, we interpret the code $\mathbf{Y}$ as the collection of columns $y(p_1), y(p_2),..., y(p_t)$, written in this specific order. For the $i-$th row of this collection, let $k^{\mathbf{p}}(1), k^{\mathbf{p}}(2),..., k^{\mathbf{p}}(s_i)$ be the indices of ones in this row. Suppose that the question code $\mathbf{X}$ is a superimposed $(s,t)-$code. By construction of the matrix $\mathbf{A}^{\mathbf{p}}$ it follows that its $i-$th column of the matrix $\mathbf{A}^{\mathbf{p}}$ is

$$\mathbf{a}^{\mathbf{p}}(i) = \mathbf{x}(k^{\mathbf{p}}(1)) \vee \mathbf{x}(k^{\mathbf{p}}(2)) \vee ... \vee \mathbf{x}(k^{\mathbf{p}}(s_i)),$$

i.e., it is the Boolean sum of the columns of the question code $\mathbf{X}$ with indices corresponding to ones in the $i-$ th row of the collection of columns $y(p_1), y(p_2),..., y(p_t)$. Since the question code is superimposed $(s,t)-$code, any summations of s-columns are distinct. Hence, $y_i(p_t) = 1$ when incoming codewords $\mathbf{x}$ are included in $\mathbf{a}^{\mathbf{p}}(i) = \mathbf{x}(k^{\mathbf{p}}(1)) \vee \mathbf{x}(k^{\mathbf{p}}(2)) \vee ... \vee \mathbf{x}(k^{\mathbf{p}}(s_i))$. Thus the code pair $(\mathbf{X},\mathbf{Y})$ has a reproduction property.

$(\Leftarrow)$ Suppose that the code pair $(\mathbf{X},\mathbf{Y})$ has a reproduction property. Since the matrix $\mathbf{A}^{\mathbf{p}}$ could recognize all input vectors, then all columns of Boolean sum of $s-$subsets in the matrix $\mathbf{A}^{\mathbf{p}}$ does not cover any other vectors which are not the

component of the columns. Thus, the question code **X** is a superimposed $(s,t)-$code □

b. The Functional Scheme of The Model.

The scheme functions in two modes – learning and recognizing. In the learning mode, n inputs successively receive t questions $x(1),x(2),...,x(t)$ and m inputs successively receive the answers $y(p_1),y(p_2),...,y(p_t)$, arriving synchronously with the questions. Moreover, the system has m outputs on which signals are generated only when information is recognized.

Learning occurs when the columns $\mathbf{x}(u)$ and $\mathbf{y}(p_u)$, $u=\overline{1,t}$, arrive simultaneously at the question and answer inputs. If $x_j(u)=y_i(p_u)=1$ for at least one $u=\overline{1,t}$, then the intersection of $i-$th and $j-$th buses are made conducting. Intersections not made conducting in the learning mode are blocking in the recognizing mode. In the matrix $\mathbf{A}^p$, $a_j^p=1$ if the intersection of buses i and j is conducting, and $a_j^p(i)=0$ if it is a blocking intersection.

In the recognizing mode, the signals $\mathbf{x}(u)$, $u=\overline{1,t}$, are applied only to the question inputs. The symbol 1 is produced on the $i-$th output if at least one 1 arrived at a conducting intersection of the $i-$th output and there were no 1's on the blocking intersections of this input.

In neural networks, the axons are the input buses and the exciting or retarding

synapses are produced in response to the signals from nerve fibers realizing the answer inputs.

Example 3.1.20. The following is an example of associative memory model with an input code and an output code.

In Learning process to make an associative memory.

output

$$\begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}$$

input

$$\begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Associative Memory

In recognition process, one can use an associative memory made in learning process to recognize an incoming input.

input **Associative Memory**

$$\begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}$$

Recognition result

3.2 Dynamics of Associate Memory with MDS code.

By theorem 3.1.16, we know that for perfect recognition the input code should be an $(s,t,N)-$superimposed code for input and $f(N,k)-$plan in which weights of a row is $k=s$ with the length of a codeword N for output. However, if we consider real neural systems, naturally we think that it seems to be much more complicate than the model with the combination between $(s,t,N)-$superimposed code and $f(N,k)-$plan. So then in this section we try to find out whether or not we can extend the condition for perfect recognition processes by using code based on MDS code.

3.2.1. Calculation of p-sets.

As long as we choose an $(s,t,N)-$superimposed code as question code, the associate memory model perfectly works. However, even if we choose an $(p,t,N)-$ superimposed code such that $p>s$ rather than an $(s,t,N)-$superimposed code, it is some time possible that associative model works properly. In this section we study the case with $p-$sets, $p>s$ and use the formula to find the average number of $p-$sets which cover any arbitrary codewords

of superimposed code based on given MDS code.

Consider an arbitrary MDS code $\mathbf{X}$ with parameters q, h, n of volume $t = q^h$, $h \le n-1 \le q$ and codewords $\mathbf{x}(i) = \{x_1(i), x_2(i), \ldots, x_n(i)\}, i = \overline{1,t}$. Because of MDS distance of this code is $d = n-h+1$. Denote by $A_w(n)$ the number of codewords in X of weight w then we shall introduce a theorem without proof to compute the number of codewords of weigh w [9]

Theorem 3.2.1: The number of codewords of weight w in an $[n, k, d = n-k+1]$ MDS code over GF(q) is

$$A_w(n) = \binom{n}{w}(q-1)\sum_{j=0}^{w-d}(-1)^j\binom{w-1}{j}q^{w-d-j}, \quad w = \overline{d,n}. \qquad (1)$$

Denote the zero codeword as

$$\mathbf{0} = (0,0,\ldots,0)$$

and a product of codewords $\mathbf{x}(i)$ and $\mathbf{x}(j)$ as

$$\mathbf{x}(i) \times \mathbf{x}(j) = \{x_1(i)x_1(j), x_2(i)x_2(j), \ldots, x_n(i)x_n(j)\}.$$

We will say that the set of codewords $\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(p)$ dose not cover $\mathbf{0}$ if $\mathbf{x}(1) \cdot \mathbf{x}(2) \cdot \ldots \cdot \mathbf{x}(p) \neq \mathbf{0}$. Let $C_0(p,n)$ be the number of $p-$sets which do not cover $\mathbf{0}$, then $L(p)$, the average number of the codewords covers by an arbitrary $p-$system, is [12]

$$L(p)=\frac{\left(\binom{q^h-1}{p}-C_0(p,n)\right)\cdot t}{\binom{q^h}{p}} \qquad (2)$$

Denote by $D(p,v)$ the number of $p-$ sets for which fixes v , arbitrary coordinates of $\mathbf{x}(1)\cdot\mathbf{x}(2)\cdot,\ldots,\cdot\mathbf{x}(p)\neq\mathbf{0}$, then [12]

$$D(p,v)=\begin{cases}\binom{q^{h-v}\cdot(q-1)^v}{p} & \text{if} \quad v\le h\\ \binom{A_v(v)}{p} & \text{if} \quad v>h\end{cases} \qquad (3)$$

where $A_v(v)=(q-1)\sum_{j=0}^{h-1}(-1)^j\binom{v-1}{j}q^{h-j-1}$ the number of codeword with weight v in $[n,h]$ MDS-code. The explanation of this formula is following. Since $p-$ sets of arbitrary codewords length v dose not cover $\mathbf{0}$, we need to chose some codewords which do not include the 0 element in itself. Hence, the number of vectors which have nonzero entrances on all v fixed coordinates is $q^{h-v}(q-1)^v$ for $v\le h$.

For $v>h$, we fix v arbitrary strings in our MDS code X and consider a new code Y which has the same volume and whose strings are those v strings from X, then the number of vectors with nonzero entrances on all v fixed position s equal $A_v(v)$. Now we can determinate $C_0(p,n)$ by the formula of inclusion and exclusion

$$C_0(p,n)=n\cdot D(p,1)-\binom{n}{2}D(p,2)+\cdots(-1)^{n+1}D(p,n)=\sum_{i=1}^{n}(-1)^{i+1}\binom{n}{i}D(p,i). \qquad (4)$$

Example 3.2.2: Consider $q=3$, $h=2$, [3,2,2]−MDS code which has $3^2=9$ codewords and find $C_0(p,n)$, the number of $p-$sets dose not cover **0** with $p=3, n=3$.

A codewords of his [3,2,2]−MDS code are

$$C=\begin{pmatrix}0&0&0&1&1&1&2&2&2\\0&1&2&0&1&2&0&1&2\\0&2&1&2&1&0&1&0&2\end{pmatrix}.$$

For $v=1$, $D(3,1)$ is

$$D(3,1)=\begin{pmatrix}3^{2-1}\cdot(2)^1\\3\end{pmatrix}=\begin{pmatrix}6\\3\end{pmatrix}=20.$$

$\begin{pmatrix}1&1&1&2&2&2\\0&1&2&0&1&2\end{pmatrix}$ **is the set of 6 columns from which $p=3$ columns should be** chosen.

For $v=2$, $D(3,2)$ is

$$D(3,2)=\begin{pmatrix}2^2\\3\end{pmatrix}=\begin{pmatrix}4\\3\end{pmatrix}=4$$

$\begin{pmatrix}1&1&2&2\\1&2&1&2\end{pmatrix}$ **is the set of 4 columns from which $p=3$ columns should be chosen.**

For $v=3$, because of $v>h$ $D(3,3)$ is

$$A_3(3)=(3-1)\sum_{j=0}^{2-1}(-1)^j\begin{pmatrix}3-1\\j\end{pmatrix}q^{2-j-1}=2.$$

This is obvious when we see the code in which there are only 2 columns with weight $w=3$, i.e., $\begin{pmatrix}1 & 2\\ 1 & 2\\ 1 & 2\end{pmatrix}$. However, we can not choose $p=3$ columns out of 2 columns.

Hence,

$$D(3,3)=\binom{A_3(v)}{3}=0.$$

$C_0(p,n)$, the number of $p-$sets dose not cover $\mathbf{0}$ is

$$C_0(3,3)=\binom{3}{1}D(3,1)-\binom{3}{2}D(3,2)=3\times 20-3\times 4=48.$$

Since this MDS code is relatively small, we can check this answer with the following calculation. There are 8 columns in the code except $\mathbf{0}$ and thus $\binom{8}{3}=56$ possible sets exist. However, there exist eight sets which cover $\mathbf{0}$ in $\binom{8}{3}=56$ possible sets. Thus $56-8=48$.

The average number of outsiders covered by an arbitrary $p-$sets of codewords is

$$L(3)=\frac{\left(\binom{3^2-1}{3}-C_0(3,3)\right)\cdot 3^2}{\binom{3^2}{3}}=\frac{6}{7}.$$

The following figures represent the results of the calculation of the average number of $p-$sets which cover any arbitrary codewords of superimposed.

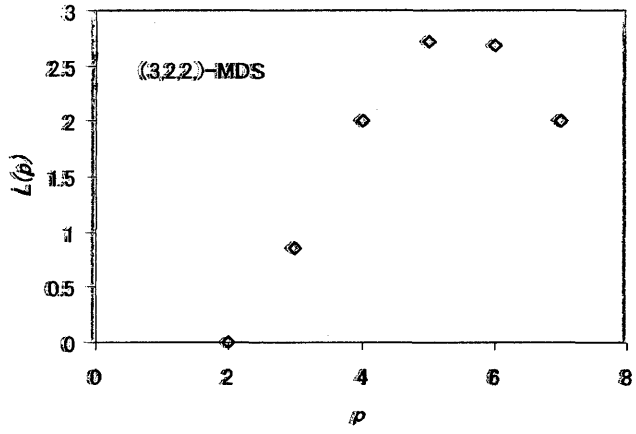


Fig. 3.2.1 Calculation result of $L(p)$ for (3,2,2)-MDS code.

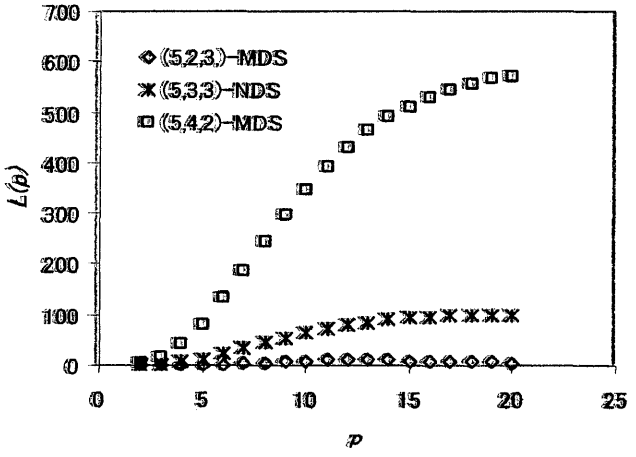


Fig. 3.2.2 Calculation result of $L(p)$ for (5,h,d)-MDS code.

	L(p)		
p	(5,2,4)-MDS	(5,3,3)-MDS	(5,4,2)-MDS
2	0.00	0.00	2.31
3	0.00	1.51	14.62
4	0.00	5.87	41.21
5	0.40	13.21	81.56
6	1.66	22.79	131.94
7	3.38	33.58	187.57
8	5.30	44.60	244.08
9	7.10	55.10	298.20
10	8.54	64.59	347.82
11	9.53	72.84	391.85
12	10.06	79.78	429.94
13	10.16	85.44	462.25
14	9.92	89.95	489.22
15	9.41	93.43	511.45
16	8.70	96.03	529.56
17	7.87	97.91	544.17
18	6.95	99.19	555.85
19	5.99	99.98	565.10
20	5.00	100.38	572.36

Table. 3.2.1 Calculation result of L(p) for (5,h,d)-MDS code.

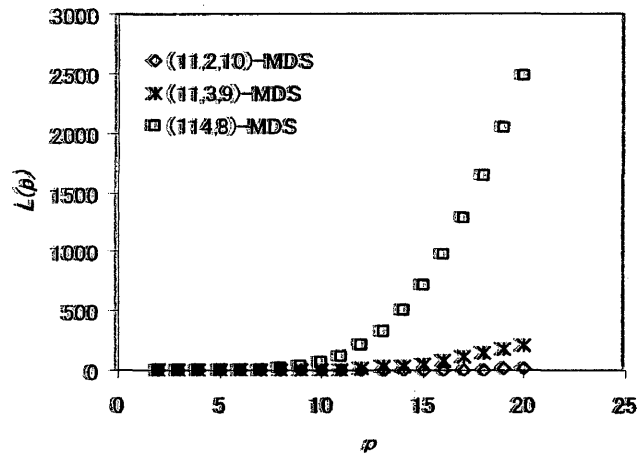


Fig. 3.2.3 Calculation result of $L(p)$ for (11,h,d)-MDS code.

	L(p)		
p	(11,2,10)-MDS	(11,3,9)-MDS	(11,4,8)-MDS
2	0.00	0.00	0.00
3	0.00	0.00	0.00
4	0.00	0.00	0.01
5	0.00	0.00	0.16
6	0.00	0.01	1.03
7	0.00	0.22	4.16
8	0.00	0.50	12.46
9	0.00	1.53	30.48
10	0.00	3.77	64.23
11	0.01	7.87	120.74
12	0.06	14.57	207.35
13	0.22	24.57	330.91
14	0.58	38.46	497.16
15	1.23	56.68	710.13
16	2.27	89.47	971.88
17	3.78	106.84	1282.42
18	5.89	138.64	1639.83
19	8.30	174.54	2040.51
20	11.28	214.07	2479.55

Table. 3.2.2 Calculation result of $L(p)$ for (11,h,d)-MDS code.

Remark 3.2.3: $L(p)=0$ means that there is no such $p-$set covering arbitrary codewords in an MDS code, hence, if we choose the max p such that $L(p)=0$,then we get (p,t,N)-superimposed code.

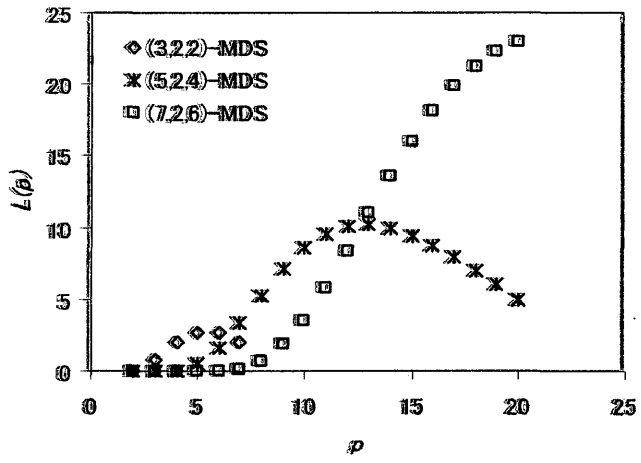


Fig. 3.2.3 Comparing calculation results of three MDS codes.

3.2.2. The Probability of Extra Ones in The Recurrent Associate Map.

In an MDS code the number of ones in a row is calculated by the formula

$$p = q^{h-1}.$$

Hence, as an input and an output code for the recurrent model of associative map, we can choose an MDS code with $p = q^{h-1}$ one's. However if we choose $h > 2$, when p goes to relatively large number we always get $L(p) > 1$, i.e., any $p-$sets covers some codeword in a given MDS. So then, the following discussion is given for $h = 2$ MDS code, i.e., $p = q$. In this case we always get $L(p) < 1$.

Since we know the number of $p-$sets that cover a codeword in given MDS code, then the probability that a $p-$sets covers a codeword satisfies following equation.

$$P(p,q,2) = \frac{\binom{q^2-1}{p} - C_0(p,n)}{\binom{q^2-1}{p}}. \qquad (5)$$

Let $p = q$, then equation above can be simplified and the function has only one variable p.

$$P(p) = \frac{\binom{p^2-1}{p} - C_0(p,n)}{\binom{p^2-1}{p}} \qquad (6)$$

	(p,2,p−1)-MDS code
p	P(p)
3	0.1428571429
4	0.0593406593
5	0.0240918502
7	0.0038019765
8	0.0014884993
9	0.0005787734
11	0.0000861563
13	0.0000126313
16	0.0000006952
17	0.0000002634
19	0.0000000376
23	0.0000000008
25	0.0000000001

Table 3.2.3. Calculation result of the probability for (p,2,p-1)-MDS code.

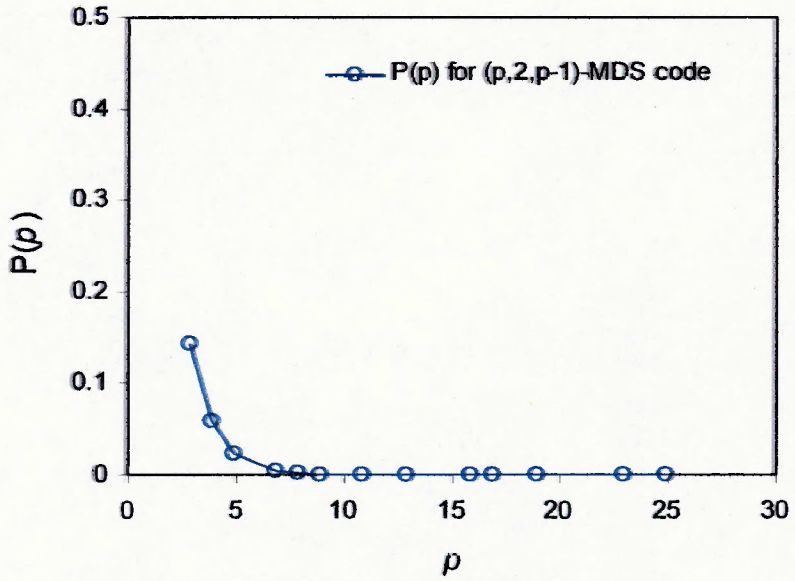


Fig 2.3.4. Calculation result of the probability for (p,2,p-1)-MDS code.

Now we talk about associate map. The map has $N = n \cdot q$ sums of $p-$sets . According to the rule of Palm's model, an associated response has ones when an

input vector is included in sum of $p-$sets, i.e., a column of the map. If we choose a vector from an input MDS code for recognition process, it is obvious that p positions out of N positions have positively ones, because p number of Boolean sums of $p-$sets must consist of the incoming vector itself. The problem is how many extra ones may appear in $N-p$ positions of an output, namely, how many errors will occur with an incoming vector?

Since there are $N=p^2$ codewords, we have $\binom{p^2}{p}$ number of $p-$sets in which there exist $\binom{p^2-1}{p}-C_0(p)$ number of $p-$sets which cover a codeword and $C_0(p)$ number of $p-$sets which do not cover any codeword. Furthermore, we can regard the associate map as a set of $p-$sets which has N number of elements and all elements come from $\binom{p^2}{p}$ number of $p-$sets. Since p out of N positions have always ones, this indicates us that p number of $p-$sets which covers an incoming vector is already used and they never appear in other $N-p$ positions because each row in an MDS code is distinct. Hence, in order to calculate the probability of extra ones in other $N-p$ positions, we need to subtract a codeword out of p^2 number of codewords.

Suppose we have x extra ones in $N-p$ positions, then we can choose $\binom{\binom{p^2-1}{p}-C_0(p,n)}{x}$ number of $p-$ sets which cover a vector and also

$\begin{pmatrix} C_0(p,n) \\ N-p-x \end{pmatrix}$ number of $p-$sets which does not cover a vector. Hence, for the probability distribution of extra ones, we multiply the both numbers and divide it by the total possible combination of $p-$sets in $N-p$ positions, i.e., $\begin{pmatrix} \begin{pmatrix} p^2-1 \\ p \end{pmatrix} \\ N-p \end{pmatrix}$.

Thus, define X as a random variable of the number of extra ones in an output, then the probability distribution of X , $B(x,p)$, is calculated by the equation.

$$B(x,p)=\frac{\begin{pmatrix} \begin{pmatrix} p^2-1 \\ p \end{pmatrix}-C_0(p) \\ x \end{pmatrix}\cdot\begin{pmatrix} C_0(p) \\ N-p-x \end{pmatrix}}{\begin{pmatrix} \begin{pmatrix} p^2-1 \\ p \end{pmatrix} \\ N-p \end{pmatrix}} \quad \text{, for } x=0,1,2,\ldots,G(p) \qquad (7)$$

And also the probability after some iterations is calculated by the following equation;

$$B^y(x,p)=\left(\frac{\begin{pmatrix} \begin{pmatrix} p^2-1 \\ p \end{pmatrix}-C_0(p) \\ x \end{pmatrix}\cdot\begin{pmatrix} C_0(p) \\ N-p-x \end{pmatrix}}{\begin{pmatrix} \begin{pmatrix} p^2-1 \\ p \end{pmatrix} \\ N-p \end{pmatrix}}\right)^y . \qquad (8)$$

where y is the number of iteration. However, if we choose relatively large MDS code, we can assume that the error of probability is going to decrease so we can get a

very simple equation known as binomial distribution.

$$b(x, N-p, P(p)) = \binom{N-p}{x} \cdot P^x(p) \cdot (1-P(p))^{N-p-x} \quad \text{for } x = 0,1,2,\ldots,N-p. \quad (9)$$

And the expected value $E(X)$, the variance $V(X)$ and the standard deviation $\sigma(X)$ are found by followings.

$$E(X) = P(p) \cdot N - p, \quad (10)$$

$$V(X) = (N-p) \cdot P(p) \cdot (1-P(p)), \quad (11)$$

$$\sigma(X) = \sqrt{(N-p) \cdot P(p) \cdot (1-P(p))} \quad (12)$$

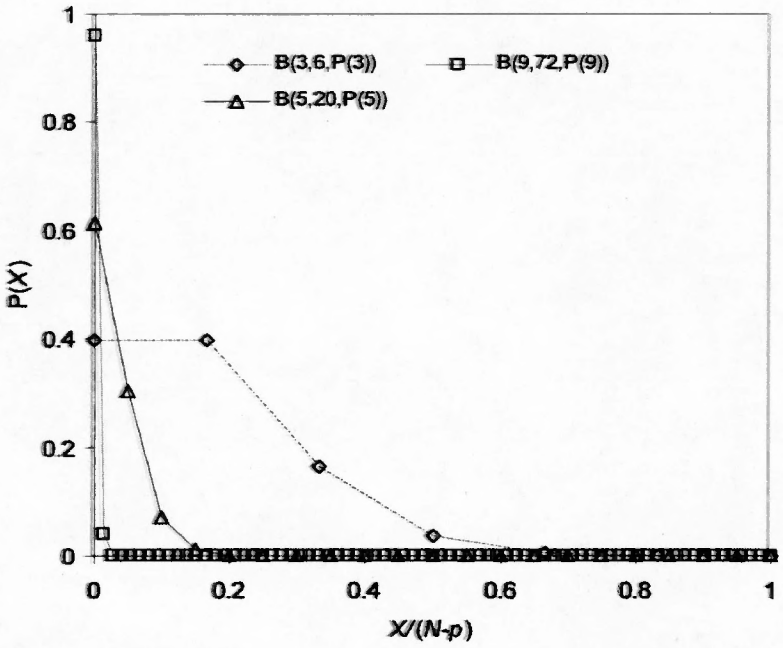


Fig 2.3.5. Binomial distribution of $X/(N-p)$.

p	binomial	Equation (1)
3	0.3965694566	0.3779520516
4	0.3965694566	0.4784714721
5	0.6140148805	0.6139471048
7	0.8521545778	0.8521545397
8	0.9199664662	0.9199664656
9	0.9591730710	0.9591730710
11	0.9905671742	0.9905671742
13	0.9980314454	0.9980314454
16	0.9998331765	0.9998331765
17	0.9999283683	0.9999283683
19	0.9999871334	0.9999871334
23	0.9999996173	0.9999996173
25	0.9999999360	0.9999999360

Table 3.2.4. Probability of no-extra ones in an output.

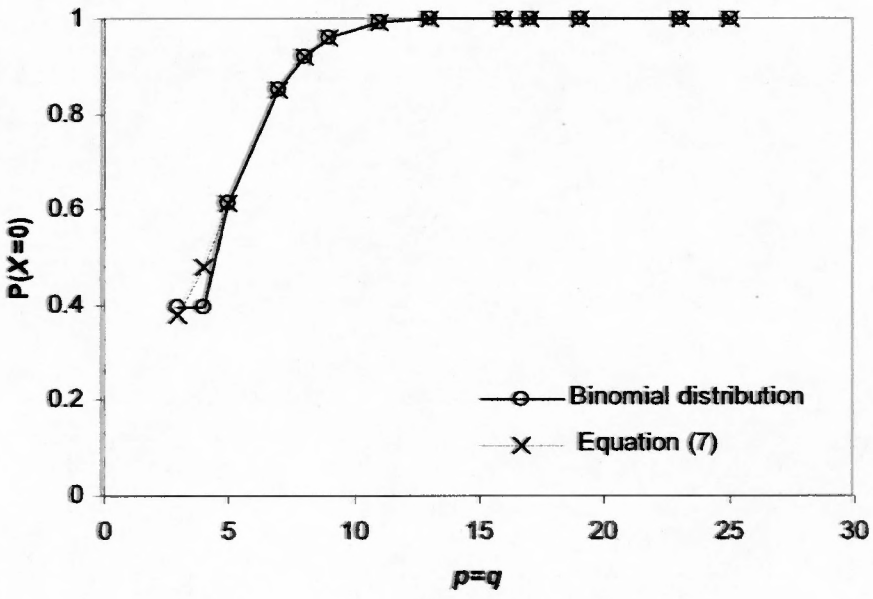


Fig 2.3.6. Probability of no-extra ones in an output.

On Fig.2.3.6 we plot two calculation results, binomial and equation (7), to compare. Both calculation results are going to be closer when q increases.

4. SUMMARY

In this thesis we tried to study the behavior of Boolean Networks, Cellular Automata (CA) and Associate memory model with coding theory.

In the first part of the thesis we studied the computability of 1-D CA by inventing a simple CA model, 3 colored balls model (3CBM). With the model we implemented "NOT", "OR" and "AND" collision based gates to show that my simple model can be computational universal. Since the model can be computational universal and all gates in this model have a common property that is "annihilation ", we tried to use this knowledge to show that other 1-D CA , R110 and R184, could be computational universal too.

In the other part of my thesis, we studied Associate Memory models with coding theory. To study dynamics of the recurrent model, we used same MDS code as an input and an output and found the probability distribution of the number of extra ones that may appear through associate map. By the probability distribution we found, it was seen that the larger MDS code is used, the number of error is going to decrease and also by comparing to binomial distribution, both results are close and when we choose the larger MDS code the difference of both is getting closer.

References

[1] S. Wolfram, A New Kind of Science.

[2] A.Ilachinski, Cellular Automata. World Scientific, 2001.

[3] A. Adamatzky, Computing in nonlinear Media and Automata Collectives. IOP, 2001.

[4] R. Hill, A First Course in Coding Theory. Oxford University Press, 1986.

[5] G. Palm, On Associative Memory. *Biological Cybernetics* 36,19 31 1980.

[6] V.V. Rykov and A.G.D'yachkov. On A Model of Associative Memory. *Problemy Predachi Informatsii*, vol. 24, no.3, pp.107-110, 1988.

[7] A.G.. D'yachkov, A.J.Macula and V.V. Rykov. New Applications and Results of Superimposed Code Theory Arising from the Potentialities of Molecular Biology. In the book "Numbers, Information and Complexity", pp.265-282, Kluwer Academic Publishers, 2000.

[8] A.G. .D'yachkov and V.V. Rykov. Optimal Superimposed Codes and Designs for Renyi's Search Model. *Journal of Statistical Planning and Inference,* vol. 100, (2002) pp. 281-302.

[9] F.J. Macwilliams, N.J.A.Sloane. The Theory of Error-Correcting Codes. North-Holland publishing company, 1978.

[10] A.G. .D'yachkov and V.V. Rykov. A Survey of Superimposed Code Theory. *Problem of Control and Information Theory*, Vol. 12 No. 4, pp. 229-242, 1983.

[11] W.H. Kautz, R.C. Singleton, Nonrandom Binary Superimposed Codes, *IEEE Trans. Inform. Theory*, vol. 10, no. 4, pp. 363-377, 1964.

[12] V.V.Rykov and S.M.Yekhanin, Average number of vectors covered by an arbitrary p-system of codewords of superimposed code based on MDS code. *Private correspond.*