# High Quality Random Number Generator using Programmable Cellular Automata


By

Azfar Shakeel Khan
2007-NUST-MS-PhD IT-23

## Supervisor

**Dr. Nazar Abbas Saqib**
Associate Professor
NUST- SEECS

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Information Technology (MS IT)

in

School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

**(April 2011)**

# APPROVAL

It is certified that the contents and form of thesis entitled "**High Quality Random Number Generator using Programmable Cellular Automata**" submitted by **Mr. Azfar Shakeel Khan** have been found satisfactory for the requirement of the degree.

Advisor: **Dr. Nazar Abbas Saqib**

Signature: _____

Date: _____

Committee Member1: **Dr. Fauzan Mirza**

Signature

_____

Date:

_____

Committee Member2: **Dr. Zahid Anwar**

Signature

_____

Date:

_____

Committee Member3: **Mr. Ammar Karim**

Signature

_____

Date:

_____

# Abstract

Random Numbers have applications in a number of computer sciences domains specifically in computer security algorithms they play a vital role. One of the techniques to generate a random number is to follow an algorithm which tries to generate random numbers as close as a true random number generator source, this algorithm based random number generation is known as Pseudo random Number Generation (PRNG). In this research we studied existing techniques of Cellular Automata (CA) based Pseudo Random generator (PRNG) and proposed 5 different variations in the existing algorithm, modified the CA rules as per requirement for the proposed techniques i.e. 3D Neuman and Moore neighborhood, 2D to 3D jumping Neuman and 2D Neuman and Moore jumping neighborhoods. In all the cases Neuman neighborhood means non diagonal adjacent neighbors while Moore neighborhood means diagonally adjacent neighbors. CA rules define which neighbors participate in the current cycle for random number generation.

We implemented the simulation code in C++ and tested the outputs in internationally accepted standard for Random Numbers “DIEHARD”, the analysis of the results are made as per recommendations of ANSI standards and it is discussed which of the above techniques can be declared to be best in terms of quality of randomness in random numbers generation.

# Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person nor material which to a substantial extent has been accepted for the award of any degree or diploma at SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis.

Any contribution made to the research by others, with whom I have worked at SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: Azfar Shakeel Khan

Signature:

# Acknowledgments

First and foremost, I am immensely thankful to Almighty Allah for letting me pursue and fulfill my dreams. Nothing could have been possible without His blessings.

I am deeply indebted to my supervisor, Dr Nazar Abbas Saqib, for providing me the supervision, motivation and encouragement throughout the span of my work. His insight, breadth of knowledge, and enthusiasm has been invaluable. Without his care and able guidance, I would not have been able to complete my thesis.

I am also grateful to my committee members, Dr Fauzan Mirza, Dr Zahid Anwar and Mr. Ammar Karim for their valuable suggestions and guidance in my research. They were abundantly helpful, and I owe a lot to them.

I am thankful to National University of Science and Technology (NUST), Pakistan for providing me scholarship due to which I carried out my research activities with good mind set.

Azfar Shakeel Khan

# Table of Contents

# List of Figures

# Chapter 1 Introduction

## 1.1 Random Number

Random numbers have applications in many areas from simulation, game-playing, cryptography, statistical sampling, and evaluation of multiple integrals, particle transport calculations, and computations in statistical physics, to name a few.

Random numbers are numbers that occur in a sequence such that two conditions are met, first the values are uniformly distributed over a defined interval or set, and secondly it is impossible to predict future values based on past or present ones. Random numbers are important in statistical analysis and probability theory.

Although it may look simple at first sight to give a definition of what a random number is, it proves to be quite difficult in practice.

A random number is a number generated by a process, whose outcome is unpredictable, and which cannot be sub sequentially reliably reproduced. This definition works fine provided that one has some kind of a black box, such a black box is usually called a random number generator that fulfills this task.

However, if one were to be given a number, it is simply impossible to verify whether it was produced by a random number generator or not. In order to study the randomness of the output of such a generator, it is hence absolutely essential to consider sequences of numbers.

It is quite straightforward to define whether a sequence of infinite length is random or not. This sequence is random if the quantity of information it contains in the sense of Shannon's information theory is also infinite.

## 1.2 Motivation

As the discussions made in the above sections, there are lot of applications in the domain of computer sciences, IT and other fields of research where random numbers play a vital role, studying existing solutions to generate a random sequence and what possibilities are there to improve the quality of randomness and efficiency of random

number generators is a vast and interesting topic, its very motivating to find a new or better random number generator or discovering better technique for implementing the existing solutions as this addition in the knowledge domain has practical applications and can be useful for many other applications.

## 1.3 Problem Statement

After doing a comprehensive literature survey, understanding of what random numbers are, their properties, testing quality of randomness we have selected the topic of Cellular Automata to work on, there is vast scope in the filed of cellular automata specifically with respect to the two and three dimensional structures as very less people have worked on this area and comprehensive results specifically with respect to a three dimensional cellular automata are not found in the literature, So we have decided to narrow down the area of interest, following statement can be treated as our problem statement.

“High Quality PRNG exploring 3D PCA Structure and possibilities for 2D to 3D jumping PCA”

## 1.4 Goals and Objectives of the Research

The objectives of the research is to add new ideas with respect to neighbor selection criteria in programmable cellular automata and their impact on the quality of randomness, one of the goals is finding a better random generator than the existing solutions, this opens a new track of research for the future as well as all possible neighborhood combinations can not be covered in a single research.

Our specific goals to be achieved are

- Implementation and Results for 2-D PCA [both for von Neumann and Moore neighborhood].
- 3-D PCA Implementation with All Neighbors.

- 2-D to 3-D jumping PCA where the Z- Neighbors are selected depending on the z-index control register, this area is never explored before.

## 1.5 Scope of the Research

The scope of this research is limited to programmable cellular automata for 3D lattice of cells and its variations for 3D to 2D jumping, both the Neumann and Moore neighbors are considered, the work studies choosing different neighbors and their effect on the quality of random number sequence generated, this work does not include FPGA implementation details and their hardware complexities due to limited amount of time.

## 1.6 Thesis outline

The document has the following chapters.

**Chapter 2:** Background and Experimental Setup

Discuses in general about the types of random numbers, various commonly used generators and about Cellular Automata generators, it also describes about the background needed for testing and simulation environment we used to accomplish the proposed solutions.

**Chapter 3:** Literature Review/ Related work

Discuses the most relevant research papers regarding the programmable cellular automata, the authors approach and the results they found.

**Chapter 4:** Methodology

Discuss the work flow and steps taken to accomplish this research work.

**Chapter 5:** Proposed Solutions

Detail discussion about the solutions proposed, the algorithm, mechanism of working for picking different neighbors and the programmable cellular automat rules applied.

**Chapter 6:** Testing of Proposed Solutions

In this chapter testing strategy is discussed, the tests we applied and the process of testing of proposed random number generators is explained.

**Chapter 7:** Analysis of Results

This chapter describes the analysis of all the results we got as a result of testing; the criteria of analysis, memory and run time cost comparisons are also discussed.

**Chapter 8:** Conclusion and Future Work

This chapter gives the concluding remarks, which random number generator is the best among the proposed methods and also what future work can be done if some one wants to extend this piece of work.

At the end of report the Appendices discuss important code and data structures, DIEHARD dumps and detail results of each implemented solution.

# Chapter 2 Background

In this chapter first we will discuss some background about types of random numbers and commonly used random number generators and then the tools and technologies required to accomplish the implementation of our proposed solutions, for simulation of each proposed generator we used standard C++ complier, for testing the DIEHARD tool is used and for comparison of results, calculating means and standard deviations and other analysis Microsoft Excel is used.

## 2.1 Types of random Numbers

In this section we briefly discuss the major categories of Random Number generators.

### 2.1.1 True Random Numbers or Physical Random-Number Generators

Games of chance are the classic examples of random processes, and the first choice would be to use traditional gambling devices as random-number generators.

Unfortunately, these devices are rather slow, especially since the typical computer application may require millions of numbers per second. Also, the numbers obtained from such devices are not always truly random e.g. cards may be imperfectly shuffled, dice and wheels may not be balanced, the random sequence obtained through the physical sources are thus know as physical random numbers or true random numbers, other physical phenomenon can also be considered in this regard, like the little variations in somebody's mouse movements or in the amount of time between keystrokes, a really good physical phenomenon to use is a radioactive source. The points in time at which a radioactive source decays are completely unpredictable, and they can quite easily be detected and fed into a computer. Another suitable physical phenomenon is atmospheric noise, which is quite easy to pick up with a normal radio.

Similarly background noise from an office or laboratory or even from a car engine can also be used as source.

One important shortcoming of any physical generator is the lack of reproducibility.

### 2.1.2 Pseudo Random Number Generators or Arithmetical Random Generators

The most common method of generating pseudo-random numbers on the computer uses a recursive technique called the linear-congruential, or Lehmer, generator. The sequence is defined on the set of integers by the recursion formula

$$X_{n+1} = AX_n + C \text{ (MOD } m)$$

One method to improve the pseudo-random number generators is to use a combination of two or more unrelated generators.

Some other well known PRNG techniques are discussed below very briefly.

**Linear Feed back Shift Registers (LFSR)**

A feedback shift register is made up of two parts: a shift register and a feedback function. The shift register is a sequence of bits. (The length of a shift register is figured in bits; if it is n bits long, it is called an n-bit shift register.) Each time a bit is needed, all of the bits in the shift register are shifted 1 bit to the right. The new left-most bit is computed as a function of the other bits in the register. The output of the shift register is 1 bit, often the least significant bit. The period of a shift register is the length of the output sequence before it starts repeating.

The simplest kind of feedback shift register is a linear feedback shift register, or LFSR. The feedback function is simply the XOR of certain bits in the register; the list of these bits is called a tap sequence. Sometimes this is called a Fibonacci configuration.

**Geffe Generator**

This key stream generator uses three LFSRs, combined in a nonlinear manner
Two of the LFSRs are inputs into a multiplexer, and the third LFSR controls the output of the multiplexer. If a1, a2, and a3 are the outputs of the three LFSRs, the output of the Geffe generator can be described by

$$b = (a1 \wedge a2) \bullet ((\neg a1) \wedge a3)$$

a variation is known as Generalized Gaffe Generator, Instead of choosing between two LFSRs, this scheme chooses between k LFSRs, as long as k is a power of 2. There are k + 1 LFSRs total. LFSR-1 must be clocked log2k times faster than the other k LFSRs.

Similar other variations of using LFSR in different combinations are, Jennings Generator, Beth-Piper Stop-and-Go Generator, Alternating Stop-and-Go Generator and Bilateral Stop-and-Go Generator.

**Self-Decimated Generators**

Self-decimated generators are generators that control their own clock. When the output of the LFSR is 0, the LFSR is clothe LFSR is clocked d times. When the output of the LFSR is 1, the LFSR is clocked k times.

**Shrinking Generator**

The shrinking generator uses a different form of clock control than the previous generators. Take two LFSRs: LFSR-1 and LFSR-2. Clock both of them. If the output of LFSR-1 is 1, then the output of the generator is LFSR-2. If the output of LFSR-1 is 0, discard the two bits, clock both LFSRs, and try again.

**A5**

A5 is the stream cipher used to encrypt GSM (Group Special Mobile). A5 consists of three LFSRs; the register lengths are 19, 22, and 23; all the feedback polynomials are sparse. The output is the XOR of the three LFSRs. A5 uses variable clock control. Each register is clocked based on its own middle bit, XORed with the inverse threshold function of the middle bits of all three registers. Usually, two of the LFSRs clock in each round.

**Additive Generators**

Additive generators (sometimes called lagged Fibonacci generators) are extremely efficient because they produce random words instead of random bits, basically these are the combinations of techniques discussed above, Fish, Pike and Mush are some of the examples of additive generators.

### 2.1.3 Properties of a Good Random Number Generator

If we go through the literature about random number generators and their qualities we get the following points for a good random generator to have:

- Random pattern : Passes statistical tests of randomness
- Long period : Goes as long as possible before repeating
- Efficiency : Executes rapidly and requires little storage
- Repeatability : Produces same sequence if started with same initializations
- Portability : Capable of producing same results on different computers

### 2.1.4 The Cellular Automata and its types

The concept of Cellular Automata (CA) was initiated in the early 1950's by J. Von Neumann and Stan Ulam, Cellular automata can be viewed as a simple model of a

spatially extended decentralized system made up of a number of individual components (cells). The communication between constituent cells is limited to local interaction.

Each individual cell is in a specific state which changes over time depending on the states of its local neighbors.

The reason behind the popularity of cellular automata is their simplicity in understanding and implementation, different structural variations of CA have been proposed in their designs and analysis of the CA to make it versatile for modeling purposes.

The two variations of neighborhood are termed as Von Neumann and Moore neighborhood, Neumann means adjacent Non Diagonal and Moore means adjacent Diagonal neighbors.

Cellular automata on multi-dimensional grids have also been proposed. The grids have either null or periodic boundary. In null boundary configurations the boundary cells are assumed to have `null' dependency. A periodic boundary is one in which the grid is considered to be folded.

The nature of next state functions also varies significantly; researchers have defined the rule set according to the design requirements of the applications. Also there are some standard rule sets, in all cases, the output of next state depends upon the output of the previous state.

In general a random number generator can be treated as a block box to which we give initial seed and it is assumed to give random number sequence as output for a desired length.

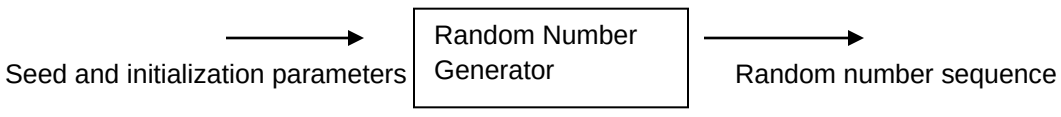



**Figure 1: Random Number Generator**

The multidimensional grid of a cellular automata can be found in various forms like 1D cellular automata, where the cells are placed adjacent to each other in linear fashion, 2D cellular automata where the lattice of cells are placed in a matrix structure that can be viewed in rows and columns, similarly in 3D cellular automata the structure can be viewed in the form of planes, rows and columns.

In each configuration each cell has certain neighbors, the number of neighbors considered is known as the radius, thus a radius of 1 means adjacent left and right neighbors in case of 1D cellular automata and four neighbors i.e. left, top, right, bottom in case of 2D cellular structure.

When we talk about a cellular automaton greater than one dimension there is also concept of Neumann and Moore neighbors in cellular automata, the Non Diagonally adjacent neighbors are known as Neumann while the diagonally adjacent neighbors are known as Moore neighbors.

## 2.2 DIEHARD Test and its Details

DIEHARD is Internationally Accepted for Randomness Testing and treated as de facto standard for testing RNGs.

DIEHARD is suit of tests, it basically comprises of 18 rigorous statistical tests, each test checks the randomness of a series of numbers on a specific parameter defined.

Most of the tests in DIEHARD return a p-value, which should be uniform on [0,1) if the input file contains truly independent random bits. Those p-values are obtained by p=F(X), where F is the assumed distribution of the sample random variable X, which is often normal.

### 2.2.1 DIEHARD Tests Applied

We applied the following tests of DIEHARD on all the proposed solutions.

- BIRTHDAY SPACINGS TEST
- THE OVERLAPPING 5-PERMUTATION TEST
- BINARY RANK TEST
- THE BITSTREAM TEST
- OPSO(Overlapping-Pairs-Sparse-Occupancy) TEST
- OQSO(Overlapping-Quadruples-Sparse-Occupancy) TEST
- DNA TEST
- COUNT-THE-1's TEST
- PARKING LOT TEST
- THE MINIMUM DISTANCE TEST
- THE 3DSPHERES TEST
- SQEEZE TEST
- OVERLAPPING SUMS TEST
- RUNS TEST
- CRAPS TEST

Here we briefly discuss the internal workings of each test applied.

- BIRTHDAY SPACINGS TEST

  Choose m birthdays in a year of n days. List the spacing's between the birthdays. If j is the number of values that occur more than once in that list, then j is asymptotically Poisson distributed with mean m^3/(4n).

- THE OVERLAPPING 5-PERMUTATION TEST

  This tests looks at a sequence of one million 32-bit random integers. Each set of five consecutive integers can be in one of 120 states, for the 5! possible orderings of five numbers. Thus the 5th, 6th, 7th, numbers each provide a state. As many thousands of state transitions are observed, cumulative counts are made of the number of occurrences of each state.

- BINARY RANK TEST

  This is the BINARY RANK TEST for 31x31 matrices. The leftmost 31 bits of 31 random integers from the test sequence are used to form a 31x31 binary matrix over the field {0,1}. The rank is determined. Ranks are found for 40,000 such random matrices and a chi-square test is performed on counts for ranks 31,30,29 and <=28.

- THE BITSTREAM TEST

  The file under test is viewed as a stream of bits. Call them b1,b2,... . Consider an alphabet with two "letters", 0 and 1 and think of the stream of bits as a succession of 20-letter words, overlapping. Thus the first word is b1b2...b20, the second is b2b3...b21, and so on. The bit stream test counts the number of missing 20-letter (20-bit) words in a string of 2^21 overlapping 20-letter words.

- OPSO(Overlapping-Pairs-Sparse-Occupancy) TEST

  OPSO means Overlapping-Pairs-Sparse-Occupancy, The OPSO test considers 2-letter words from an alphabet of 1024 letters. Each letter is determined by a specified ten bits from a 32-bit integer in the sequence to be tested. OPSO generates 2^21 (overlapping) 2-letter words, The OPSO test takes 32 bits at a time from the test file and uses a designated set of ten consecutive bits. It then restarts the file for the next designated 10 bits, and so on.

- OQSO(Overlapping-Quadruples-Sparse-Occupancy) TEST

  The test OQSO is similar, except that it considers 4-letter words from an alphabet of 32 letters, each letter determined by a designated string of 5 consecutive bits from the test file, elements of which are assumed 32-bit random integers.

- COUNT-THE-1's TEST

  It considers the file under test as a stream of bytes (four per 32 bit integer). Each byte can contain from 0 to 8 1's, with probabilities 1, 8,28,56,70,56,28,8,1 over 256. Now let the stream of bytes provide a string of overlapping 5-letter words, each "letter" taking values A,B,C,D,E.

- PARKING LOT TEST

  In a square of side 100, randomly "park" a car---a circle of radius 1. Then try to park a 2nd, a 3rd, and so on, each time parking. That is, if an attempt to park a car causes a crash with one already parked, try again at a new random location.

- THE MINIMUM DISTANCE TEST

  It does this 100 times, choose n=8000 random points in a square of side 10000. Find d, the minimum distance between the (n^2-n)/2 pairs of points. If the points are truly independent uniform, then d^2, the square of the minimum distance should be (very close to) exponentially distributed with mean .995.

- THE 3DSPHERES TEST

  Chooses 4000 random points in a cube of edge 1000. At each point, center a sphere large enough to reach the next closest point. Then the volume of the smallest such sphere is (very close to) exponentially distributed with mean 120pi/3.

- SQEEZE TEST

  Random integers are floated to get uniforms on [0,1). Starting with k=2^31=2147483647, the test finds j, the number of iterations necessary to

reduce k to 1, using the reduction k=ceiling(k*U), with U provided by floating integers from the file being tested.

- OVERLAPPING SUMS TEST

  Integers are floated to get a sequence U(1),U(2),... of uniform [0,1) variables. Then overlapping sums, S(1)=U(1)+...+U(100), S2=U(2)+...+U(101),... are formed. The S's are virtually normal with a certain covariance matrix. A linear transformation of the S's converts them to a sequence of independent standard normals.

- RUNS TEST

  This is the RUNS test. It counts runs up, and runs down, in a sequence of uniform [0,1) variables, obtained by floating the 32-bit integers in the specified file. Runs are counted for sequences of length 10,000. This is done ten times.

- CRAPS TEST

  It plays 200,000 games of craps, finds the number of wins and the number of throws necessary to end each game. The number of wins should be (very close to) a normal with mean 200000p and variance 200000p(1-p), with p=244/495. Throws necessary to complete the game can vary from 1 to infinity.

DIEHARD requires a binary file of 32 bit integers to be generated; following is the code we used to generate the required binary file.

First the file is opened in binary append mode using the following statement

```
if ( (outfile = fopen("Rnd2DM.rng", "ab+" )) == NULL )
```

to write each character as 8 bits, where each bit in our case represents either 1 or 0 of a random number sequence we used the technique of character wise bit shifting operator, we declared 8 unsigned characters to hold each individual bit.

unsigned char mask8, mask7, mask6, mask5, mask4, mask3, mask2, mask1;

Each mask corresponds to bit 1 at a specific position,

e.g. mask8 = mask8<<7; // 1 0 0 0 0 0 0 0

Similarly we adjust each mask at the time of writing, then copy these values to a temp character variable by using bitwise OR operator using temp = temp | mask8; at the end this temp variable is written to the binary file using the following statement.

fwrite(&temp, sizeof(temp), 1, outfile);

the whole code is written in a iterative loop, so at the end we have generated a binary file in the form as required by DIEHARD.

## 2.3 Simulation Environment

The whole code for the proposed solutions is written in C++ and compiled and tested on both Microsoft Visual C++ and Borland Turbo C++ compilers, no compiler specific routine or function is called to maintain cross compilers compatibility, the written code can be easily ported to any OOP like C# or Java if any one wish to use those programming environments in future.

# Chapter 3 Literature Review/ Related work

## 3.1 Literature Review

Random number generators are used in number of applications like computer simulations, built-in self test, and cryptography to name a few [1]. Most random number generators are deterministic in nature as the numbers are generated using some mathematical formulas or deterministic algorithms, known as Pseudo Random Number Generators or PRNG as opposed to a True random Number Generator where the source of a random number might be some natural source.

Standard Statistical tests are conducted to ensure a PRNG produces numbers that are uniformly distributed, uncorrelated with extreme long period [2]. There are quite a number of tests that can be applied to judge the quality of randomness, failing a single test does not mean that the sequence is not random so normally a set of tests is applied to check the quality of randomness, some times the tests to be applied might be application specific as well.

There are quite a number of PRNGs. Some examples are linear feedback shift register (LFSR), linear congruential generator (LCG), and cellular automata (CA) based PRNGs.

CA-based PRNGs were focus of researchers because its easy to implement in hardware, typically FPGA based Implementations, also computer based simulations in which implementation of cell structures and the corresponding CA rules in a corresponding data structure with each iteration generating the next state can be done easily.

The majority of research on CA-based PRNG has been focused on one-dimensional (1-D) CA with nearest three-cell neighborhood known as elementary CA, different types of cellular Automata and their applications are thoroughly discussed in a survey done on cellular automata [3].

The latest research focuses on increasing complexity of CA cell configurations and increasing CA dimensionality that can lead to better performance [4]. By increasing complexity we mean what logical operations are performed on a single cell per iteration and which neighbors are participating in finding the next sate value, to make the

situation further implausible the dimensions of the lattice structure is increased thus making more options for the participating neighbors.

Certain modifications in basic CA are also suggested using extra control registers which transforms the basic CA to Programmable CA (PCA) [5]. These control registers works as decision makers in each iteration for number of participating neighbors, whether the current value of cell itself participates or not and similarly any kind of transformation in value post CA-logical operation is required or not. The control register values are themselves updated after each iteration.

One of the first 2D based CA was proposed by [6] based on 8 x 8 lattice of cells, while [7] worked on Asymmetric neighborhood and lattice structure PRNG for 2D CA.

For testing of Random Numbers the DIEHARD test is used by a number of researchers as it is treated as industry standard for checking the randomness quality of random numbers [8]. DIEHARD comprises of some 18 different statistical tests which are supposed to be passed if the resultant values are in a specific range, DIEAHRD requires a binary file of at least 10MB as an input.

The analysis of results produced by the DIEHARD is done by comparing p-values and finding the safe ranges when we compare them among several algorithms as done by [9]. Similarly others values can be calculated e.g. incomplete gamma count to check the uniformity of probability distribution among the generated p-values [2].

Sheng-Uei Guan etal. [10], describes the concept of self programmable automata (SPACA)and how to generate PRNG using this technique, authors used the concept on linear form of cells where the rule selection values are updated internally along with two external control registers.

The basic concept is to add rule selection neighborhood to the state transition neighborhood of PCA, thus making it SPCA. Authors used rules 90/150/165/105 for their rule selection neighborhood.

The paper discusses the concept for 1D CA structure and more emphasis is given on making FPGA implementation more efficient and simpler, moreover no results

comparison is given in results section using same approach; however authors considered different algorithms with different length blocks for analysis, also suggested out put sampling to increase efficiency of algorithm.

## 3.2 Related Work

A Two Dimensional Cellular Automata with programmable scheme is used by Byung-Heon Kang etal [11]. The authors applied rules 15, 31, 47 and 63 with two external control registers and compared the results with Guan, authors concluded that their scheme passes all DIEHARD tests with better quality as generated by Guan.

| Created Rule Number | Complemented Control Signal | Self Control Signal |
|---|---|---|
| 15 (001111) | 0 | 0 |
| 31 (011111) | 0 | 1 |
| 47 (101111) | 1 | 0 |
| 63 (111111) | 1 | 1 |

**Figure 2: CA Rules Implemented for 2D-PCA [11]**

The two control registers are self control register and sign control register, self control register's value decide whether the current value of cell participates in finding next state and sign control's value is used to logically invert the calculated results for next state's value.

The proposed algorithm is run 100 times and a pass is considered if the p-values from DIEHARD fall in the range $0.025 < p < 0.975$. Authors also used a novel time spacing technique for minimization of auto correlations among block of random numbers.

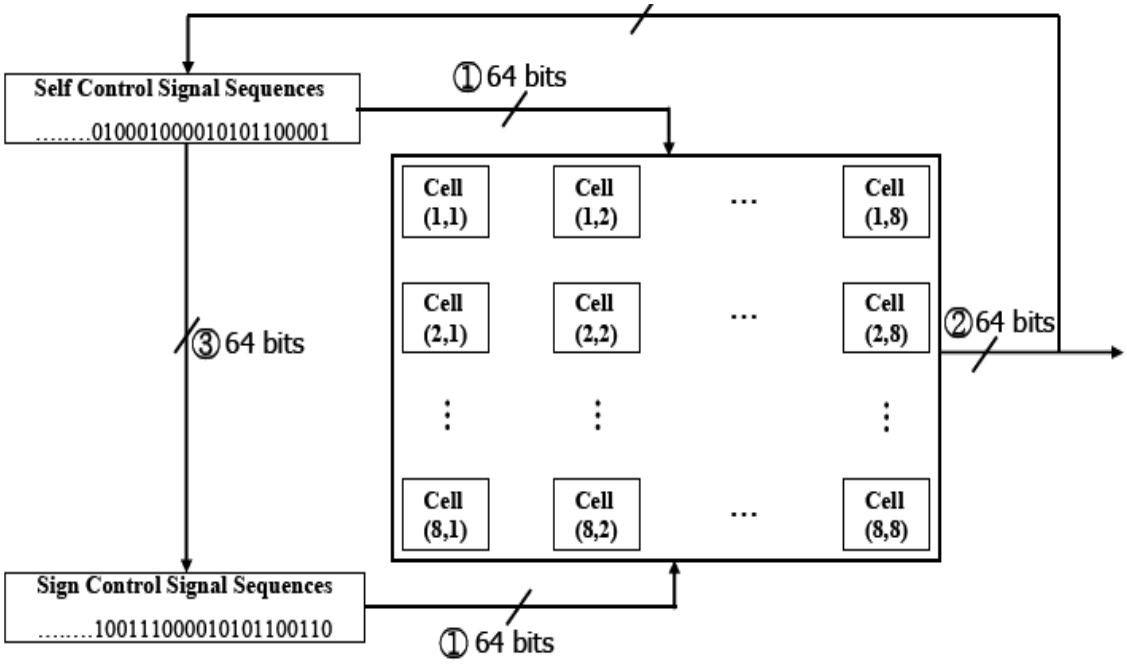



**Figure 3: 2D-PCA Structure [11]**

The research paper discusses only Von-Neumann neighbors for 2D CA implementation; strategy of how initial values are given to control registers is not discussed.

Sang-Ho Shin etal [12] worked on 3Dimmensional CA but with Moore Neighborhood only i.e. considering the diagonal neighbors, the focus of paper was to reduce the Hardware Implementation cost, the results were given in the form of Passing or Failing a DIEHARD test, what actual p-values are produced are not discussed so the quality of Randomness can not be discussed. No experimental setup or implementation methodology details are discussed, however authors claim that they have executed their scheme 100 times to generate the results.

Sang-Ho Shin, Kee-Young Yoo [13] worked on finding out on the optimal combination of Logical operations and CA rules for 3-Neighbors i.e. r+1 Von Neumann, the authors suggest a number of CA rules LCA, NCAAX, and NCAOX (as per their naming convention) for better results and also suggested that periodic boundary condition is better than the NULL boundary conditions i.e. for extreme cells the structure wraps around. The results discussed are just in the form of pass/ fail rate but the comparison is given between Null boundary and periodic boundary conditions.

| Test Name | The results of tests (Pass or Fail) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Null boundary | | | | | **Periodic boundary** | | | | |
| | $L^*$ | $N^*_{AX}$ | $N^*_{OX}$ | $N^*_{AO}$ | $N^*_{AOX}$ | $L^*$ | $N^*_{AX}$ | $N^*_{OX}$ | $N^*_{AO}$ | $N^*_{AOX}$ |
| Birthday Spacing | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** |
| Overlapping 5-Permutation | **Pass** | **Pass** | **Pass** | Fail | **Pass** | **Pass** | **Pass** | **Pass** | Fail | **Pass** |
| Binary Rank 31 × 31 | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** |
| Binary Rank 32 × 32 | **Pass** | **Pass** | **Pass** | **Pass** | Fail | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** |
| Binary Rank 6 × 8 | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** |
| Bitstream | Fail | **Pass** | **Pass** | Fail | Fail | Fail | **Pass** | **Pass** | Fail | Fail |
| OPSO | Fail | Fail | **Pass** | **Pass** | **Pass** | Fail | Fail | **Pass** | **Pass** | **Pass** |
| OQSO | **Pass** | Fail | Fail | Fail | Fail | **Pass** | Fail | Fail | Fail | Fail |
| DNA | **Pass** | Fail | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** |
| Count-The-1's 01 | **Pass** | **Pass** | **Pass** | **Pass** | Fail | **Pass** | **Pass** | **Pass** | **Pass** | Fail |
| Count-The-1's 02 | **Pass** | Fail | **Pass** | Fail | **Pass** | **Pass** | **Pass** | **Pass** | Fail | **Pass** |
| Parking Lot | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** |
| Minimum Distance | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** |
| 3DS Spheres | Fail | **Pass** | Fail | Fail | **Pass** | **Pass** | **Pass** | **Pass** | Fail | Fail |
| Squeeze | **Pass** | **Pass** | **Pass** | **Pass** | Fail | **Pass** | **Pass** | **Pass** | **Pass** | Fail |
| Overlapping Sums | **Pass** | **Pass** | **Pass** | **Pass** | Fail | **Pass** | **Pass** | **Pass** | **Pass** | **Pass** |
| Runs | **Pass** | **Pass** | **Pass** | Fail | **Pass** | **Pass** | **Pass** | **Pass** | Fail | **Pass** |
| Craps | **Pass** | **Pass** | **Pass** | Fail | Fail | **Pass** | **Pass** | **Pass** | **Pass** | Fail |
| The number of pass | 15 | 14 | 16 | 11 | 11 | 16 | 16 | **17** | 12 | 13 |

**Figure 4: Results for Different Logical Operations with CA Rules [12]**

The research paper is very good if some one wants to explore the results of different combinations of logical gates with different CA rules.

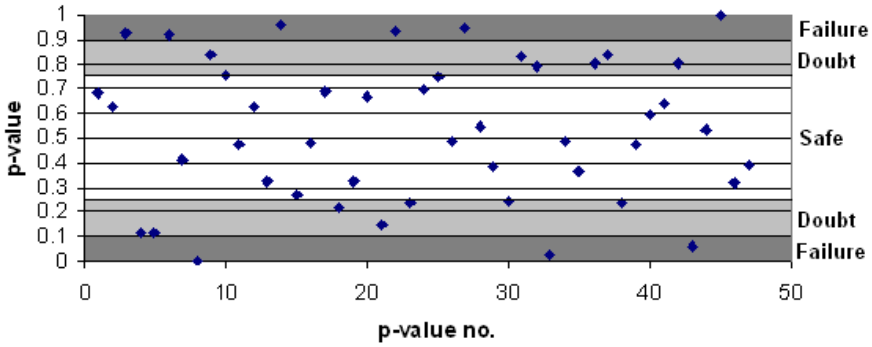



**Figure 5:DIEHARD Results Compared using Safe Window Range [9]**

Instead of comparing just Pass and Fail results as done by [12] and [13] a

better approach is adopted by [9].

Here the author has defined three regions, Failure, Doubtful and Safe values providing a more better and quantitative analysis among the results being compared.

# Chapter 4 Methodology

To complete this research work we followed the following sequence of events.

## 4.1 Literature Survey Phase

First of all a through Literature Survey is done to study current techniques applied in generation of Random Numbers specifically with respect to Cellular Automata, types of Random Numbers and their properties etc. this phase of work is already discussed in previous chapters.

## 4.2 Idea Generation Phase

Then we explored the areas where we can work and finalized the methods for Random Number Generation to be implemented. This idea generation phase lead to the proposed solutions, the methods to be implemented are given below; the details about each are discussed in chapter 5.

1. 2D Neumann Neighbors PCA
2. 2D Moore Neighbors PCA
3. 2D Neumann Moore jumping Neighbors PCA
4. 3D Neumann Neighbors PCA
5. 3D to 2D Jumping Neumann Neighbors PCA
6. 3D Moore + Neumann Neighbors PCA

## 4.3 Proposed Solution Implementation Phase

We used Computer Simulation to implement the required group of PCA for 3D CA and its variations and generated binary files of Random Numbers; the important piece of code is discussed in Appendix A.

The complete implementation is done and no complier specific function is used nor any complier settings has been made for the code to work, this makes the code portable with very little changes to any other object oriented programming language.

## 4.4 Testing Phase

In the next phase Testing is done on DIEHARD suit of Tests, DIEHARD is Internationally Accepted suit of tests for Randomness Testing and treated as de facto standard for testing RNGs, details of this phase are discussed in chapter 4 and detailed results are shown in Appendix B. 100 Binary files in the specific format as required by the DIEHARD are generated for each proposed solution and the DIEHARD dumps are gathered for the analysis phase.

## 4.5 Analysis of Results Phase

The Analysis and Comparison of Results comes in the last phase and it concludes which approach is better for Random number generation, this is discussed in chapter 6, analysis phase also describes the criteria we have selected for checking the quality of randomness of each proposed method.

## 4.6 Assumptions

We have made the following assumptions at the start of this research; these assumptions are further verified by the results produced.

- In general if you increase the complexity of neighbors the quality of Randomness for cellular automata will increase.
- By increasing dimensions the run time efficiency will not be increasing in exponential order.

- Although various other tests are present for randomness testing but DIEHARD is selected as it is considered as toughest to pass, plus DIEHARD suit contains a set of tests.

- The random numbers generated follows the normal distribution for a large data set; the large data set i.e. at least a 10MB file is also requirement of DIEHARD suit.

- The safe window size i.e. values that pass in certain range is taken from a journal paper published [9], the conclusive remarks might change if different safe window size is taken e.g. a broader window might pass those values that were considered to be failed and vice versa.

# Chapter 5 Proposed Solutions

The analysis of literature survey gives us the following points:

1. Most of the work is done on 2D CA based structures, researchers tried different length/ sized structures and their symmetric and asymmetric variations.
2. Finding out which rules or logical gates work best, e.g. instead of using XOR, if we use AND or a combination of AND, OR gates etc, which rules of CA gives better results with which combination of gates.
3. Changing the rule selection mechanism, i.e. after each iteration a specific rule has to be selected from a given set of rules, this kind of PCA is also known as Self Programmable PCA or SPCA.
4. Different control registers added for different purposes e.g. introducing a Sign Control register that decides whether to take logical inverse of the result or the value should be simply saved, this type of operations increases the level of unpredictability and hence increases the level of randomness.
5. Hardware implementations of different CA proposed by the researchers, especially FPGA based implementations and their run time efficiencies.
6. We get only one paper on 3D PCA (only for Moore neighbors) and that also does not discuss results in detail nor are any implementation details given.

So we decided to explore 2D PCA and its variations, 3D PCA and its different variations like jumping between 3D PCA to 2D PCA with greater in-depth analysis of results.

## 5.1 Proposed Methods

We have proposed the following 3D PCA and variations of 2D PCA i.e. programmable cellular automata.

1. 2D Neumann Neighbors PCA
2. 2D Moore + Neumann Neighbors PCA
3. 2D Neumann Moore jumping Neighbors PCA

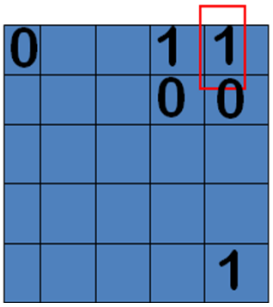
4. 3D Neumann Neighbors PCA
5. 3D to 2D Jumping Neumann Neighbors PCA
6. 3D Moore + Neumann Neighbors PCA

All the above ideas are never tried before by any of the researcher except the first one i.e. 2D Neumann Neighbors PCA, we re-implemented the 2D Neumann Neighbors PCA case to use it as a base case for comparison of results. It is to be noted that this study compares the effect of positional changes with respect to neighbors therefore we used the same algorithm for all the methods.

## 5.2 Algorithm used

We applied the following steps for each and every method.

1. Initialize All the Cell Values
2. Initialize Self Control register
3. Initialize Sign Control register
4. Pick Each Cell, select the Rule to be applied, Apply the Rule and Update Cell value
5. After Complete iteration Update the values of Registers
6. Extract 64bits Random Number (8 x 8 in case of 2D) and Go for Next iteration i.e. Step 4 for 64 * N bits of Random where N is the Number of iterations. For 3D it will be giving 8 x 64 i.e. 8 sets of 64 bit random numbers




For all the methods proposed we will apply the Non Null Boundary conditions i.e. the structure wraps around for extreme cells to pick the neighbors.

e.g. if the red boxed cell is the current cell and we want to apply rule 15 on it i.e. current cell will not participate and the neighbors to pick are left, top, right, bottom then the result will be, xor(1,1,0,0) = 1.

Now we will discuss each method one by one.

### 5.2.1 2D Neumann Neighbors

It is a two dimensional structure of 8 x 8 matrix, where we pick only Neumann neighbors i.e. immediate Non diagonal Neighbors of each cell.

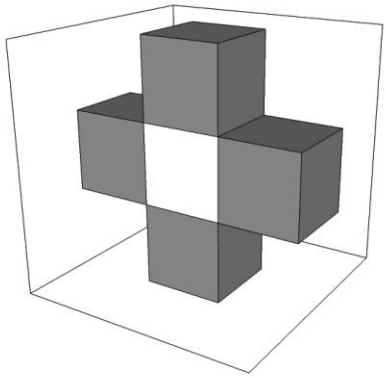


**Figure 6: 2D Neumann Neighbors Illustration**

In Fig. 6, we have the following participating neighbors,

White Cell: Current cell whose value is to be updated.

Gray Neighbors: Participating Neighbors (Non Diagonal Adjacent) Left, Top, Right and Bottom.

The cell structure is shown in Fig. 7, there are two control registers

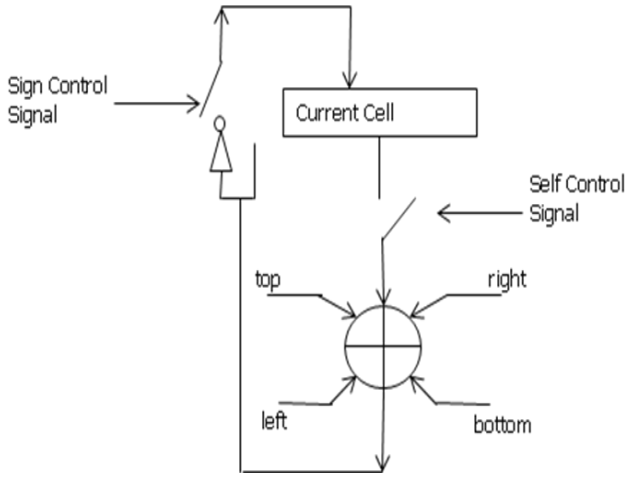



**Figure 7: 2D Neumann Cell Structure**

Sign Control: Sign control value decides whether we have to do a logical negation of result produced post XOR operation or should we store the value directly to the current cell.

Self Control: Self control value decides whether the current cells value participates in the XOR operation or only adjacent non diagonal neighbors have to participate.

Table 1. Shows the rules used during each iteration for calculating the cells next value.

| Rule | CL | SI | Left | Top | Right | Bottom |
|---|---|---|---|---|---|---|
| 15 | 0 | 0 | 1 | 1 | 1 | 1 |
| 31 | 0 | 1 | 1 | 1 | 1 | 1 |
| 47 | 1 | 0 | 1 | 1 | 1 | 1 |
| 63 | 1 | 1 | 1 | 1 | 1 | 1 |

Table1: 2D Neumann CA Rules

e.g. Applying Rule 15 means

(001111) → (15)10 == (001111)2

The first two zeros correspond to values of CL and SI register while the four 1's shows which neighbors are participating.

### 5.2.2 2D Moore + Neumann Neighbors

It is a two dimensional structure of 8 x 8 matrix, where we pick both the Neumann neighbors and Moore neighbors i.e. immediate Non diagonal Neighbors of each cell and diagonal neighbors as well, this increases the participating neighbors count and thus expected to give a better result in terms of randomness of generated number.

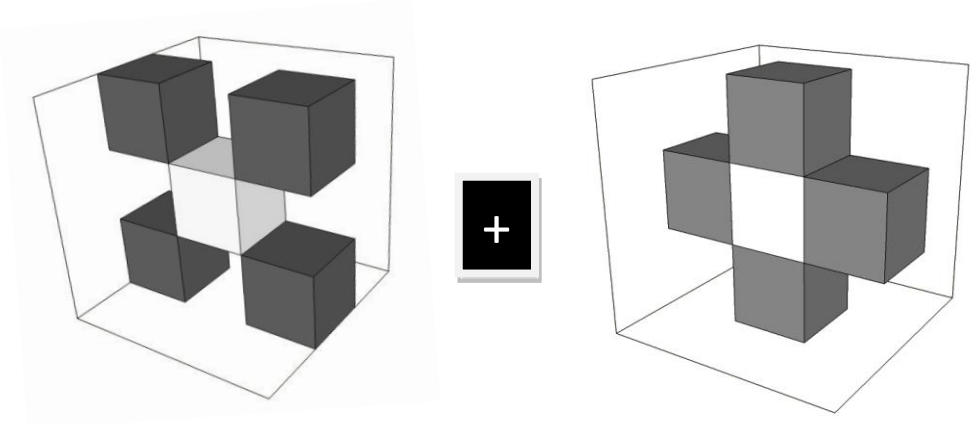


**Figure 8: 2DMoore + Neumann Neighbors Illustration**

In Fig. 8, We have the following participating neighbors,

White Cell: Current cell whose value is to be up dated.

Gray Neighbors (left): Participating Moore Neighbors (Diagonally Adjacent); Left-Top, Right-Bottom, Right-Top and Left-Bottom.

Gray Neighbors (right): Participating Neumann Neighbors, Left, Top, Right, Bottom

The Cell structure is shown in Fig. 9

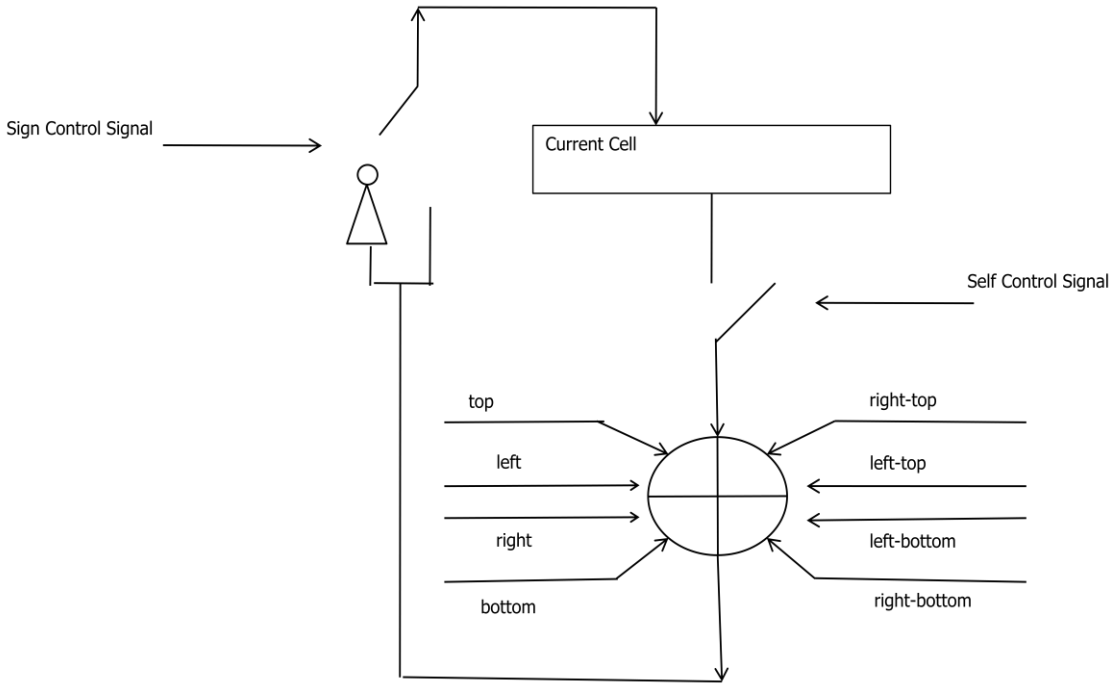



**Figure 9: 2DMoore + Neumann Neighbors Cell Structure**

There are again two control registers which play the same role as in 2D Neumann case

| Rule | CL | SI | Left | Top | Right | Bottom | Left | Right | Left | Right |
|---|---|---|---|---|---|---|---|---|---|---|

| | | | | | | | Top | Top | Bottom | Bottom |
|---|---|---|---|---|---|---|---|---|---|---|
| 255 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 511 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 767 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1023 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Table2: 2D Moore+ Neumann CA Rules

Table 2. Shows the rules used during each iteration for calculating the cells next value.

### 5.2.3 2D Neumann Moore jumping Neighbors

Its again a two dimensional structure of 8 x 8 matrix, this time we pick between Neumann neighbors and Moore neighbors depending upon the newly introduced control register i.e. some times we pick the Moore neighbors and some times we pick the Neumann neighbors.

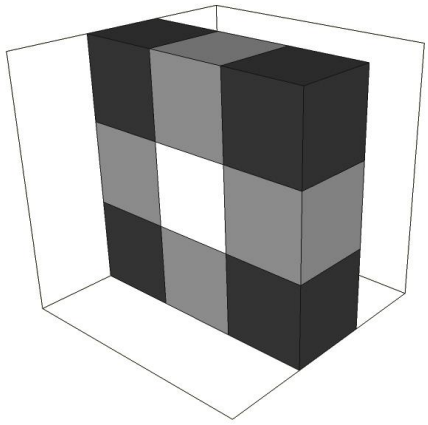


**Figure 10: 2D Neumann Moore jumping Neighbors Illustration**

In Fig. 10, we have the following participating neighbors,

White Cell: Current cell whose value is to be up dated.

Light Gray Neighbors: Participating Neighbors (Non Diagonal Adjacent) Left, Top, Right and Bottom.

Black Neighbors: Participating Neighbors (Diagonally Adjacent)

Left-Top, Right-Bottom, Right-Top and Left-Bottom.

The Cell structure is shown in Fig. 11

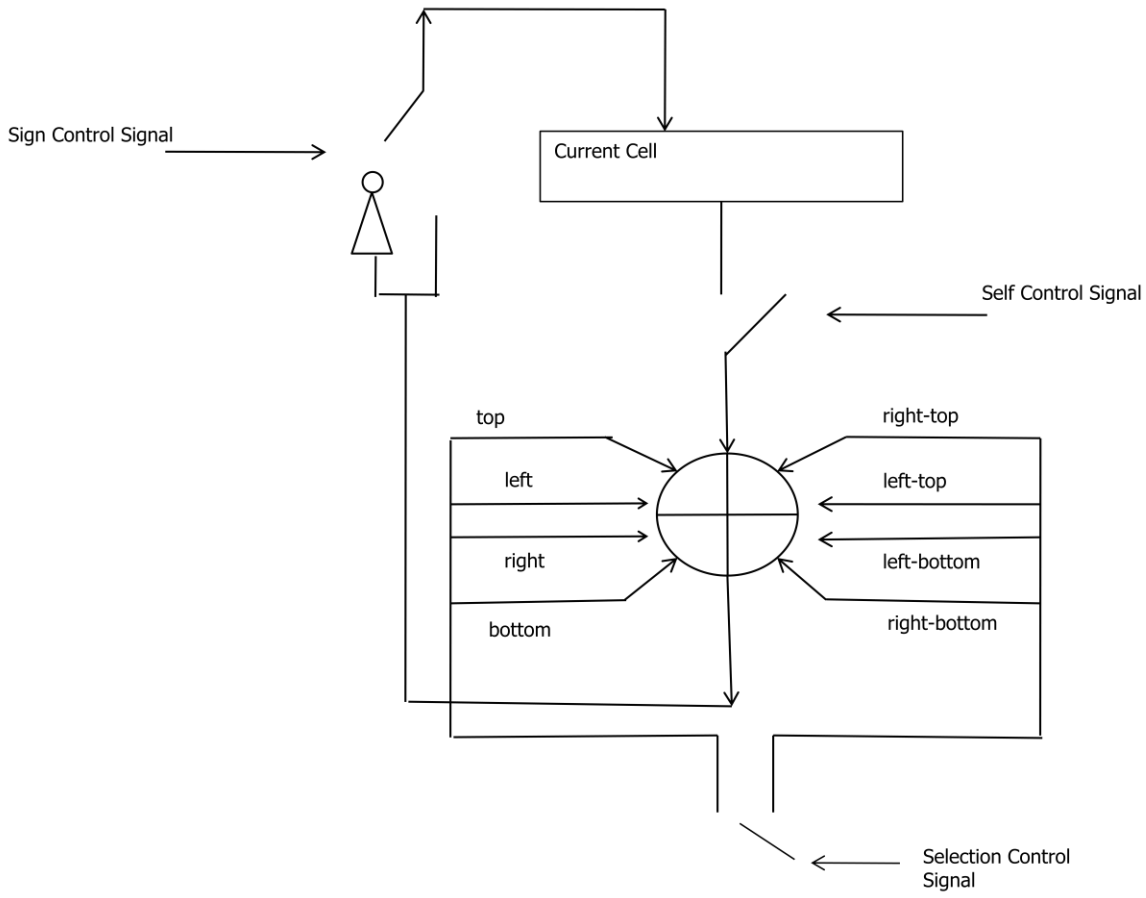



**Figure 11: 2D Neumann Moore jumping Neighbors Cell Structure**

This time we have three two control registers, the Sign and Self control registers play the same role as in 2D Neumann case while the third control register is used for deciding whether we opt for Diagonally Adjacent or Non Diagonally Adjacent neighbors.

Table 3. Shows the rules used during each iteration for calculating the cells next value.

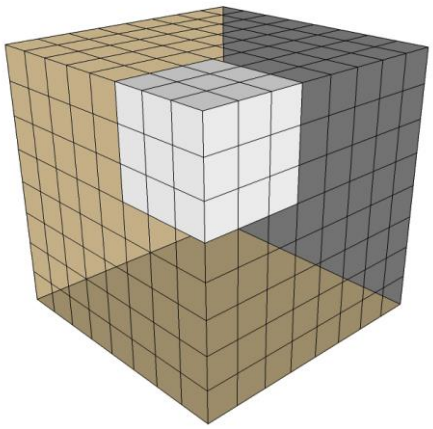
| Rule | CL | SI | DI | Left Top | Right Top | Left Bottom | Right Bottom | Left | Top | Right | Bottom |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 496 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 527 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1008 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1039 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1520 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1551 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 2032 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

In Table3. The DI is the newly added control register

Table3: 2D Neumann Moore jumping CA Rules

### 5.2.4 3D Neumann Neighbors

This time we are going to use a Three Dimensional Structure for generation of Random numbers, it will be an array



of 8 x 8 x 8 as shown in Fig. 12 consisting of 8 planes in which each plane has got 8 rows and 8 columns.

**Figure 12: 3D Lattice of cells used for all the 3D PCA Algorithms**

We will pick all the Neumann neighbors i.e. the six non-diagonally adjacent neighbors; four on the XY plane while two of them on Z axis. We will be using two control registers, one for Self Control and the other for Sign Control. Fig 13. Shows the diagram of participating neighbors while Fig. 14 shows the cell structure for the 3D Neumann case.

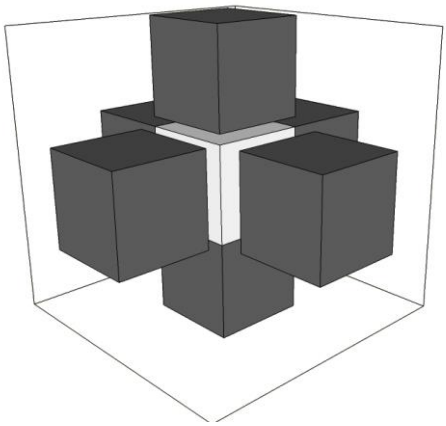


**Figure 13: 3D Neumann Neighbors Illustration**

In Fig. 13, we have the following participating neighbors,

White Cell: Current cell whose value is to be updated.

Gray Neighbors: Participating Neighbors (Non Diagonal Adjacent)

Left, Top, Right, Bottom, Facing Front on Z-axis, Facing Back on Z-axis.

The Cell structure is shown in Fig. 14

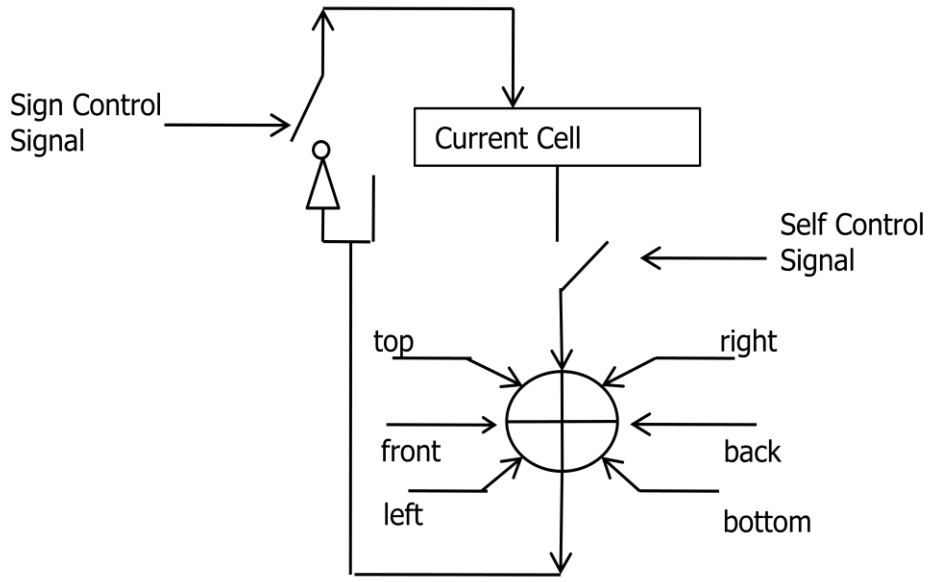



**Figure 14: 3D Neumann Neighbors Cell Structure**

Table 4. Shows the rules used during each iteration for calculating the cells next value.

| Rule | CL | SI | Back | Front | Left | Top | Right | Bottom |
|---|---|---|---|---|---|---|---|---|
| 63 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 127 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 191 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 255 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Table4: 3D Neumann CA Rules

### 5.2.5 3D to 2D Jumping Neumann Neighbors

3D to 2D Jumping Neumann Neighbors uses a 8 x 8 x 8 lattice of cells for Random numbers generation, this 3D cube structure picks all the 6 adjacent Neumann Neighbors just like the proposed solution number 4 does but with a subtle variation in which it jumps to the 2D Neumann structure i.e. picking only the adjacent non diagonal neighbors from XY plane only, this jumping is dependent upon the value a third control register.

Thus it could be considered as a mixture of solution two already proposed solutions, some times it behave like 2D Neumann CA and some times behave like a 3D Neumann CA, this kind of experimental jumping is done to check the quality of random numbers generated versus the efficiency of algorithm as approximately half of the times it is expected to be picking 2D Neumann neighbors (more efficient) and half of the time generating a random number while considering the z-plane neighbors (more randomness).

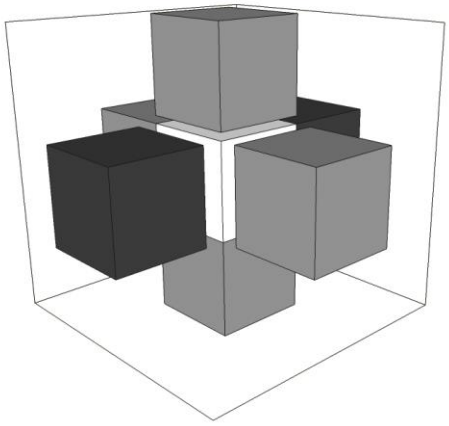


**Figure 15: 3D-2D jumping Neumann Neighbors Illustration**

Participating Neighbors are as follows,

| | |
|---|---|
| White Cell: | Current cell whose value is to be updated. |
| Light Gray Neighbors: | Participating Neighbors (Non Diagonal Adjacent), Left, Top, Right and Bottom. |
| Black Neighbors: | Participating Neighbors (Non Diagonal Adjacent), Facing Front on Z-axis, Facing Back on Z-axis + Light Gray Neighbors. |

Light Gray Neighbors are picked in case the current iteration is calculating for 2D Neumann while Black Neighbors along with Light Gray Neighbors are used in case 3D Neumann is selected for the current iteration; the cell structure is shown in Fig. 16.

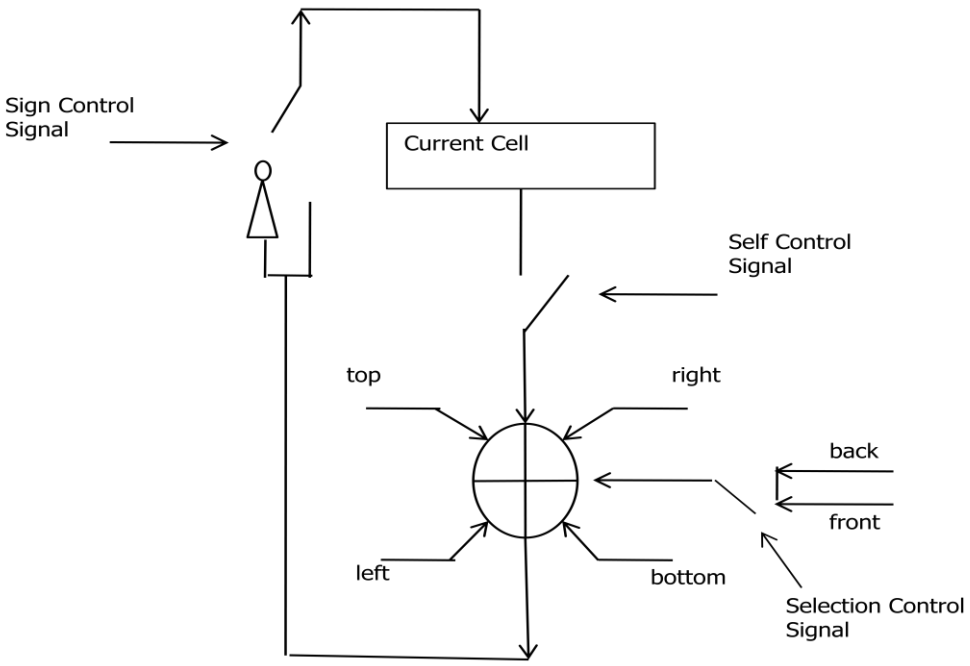



**Figure 16: 3D to 2D Jumping Neumann Neighbors Cell Structure**

Table 5. Shows the rules used in each iteration.

| Rule | CL | SI | DI | Back | Front | Left | Top | Right | Bottom |
|---|---|---|---|---|---|---|---|---|---|
| 15 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 127 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 143 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 255 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 271 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 383 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 399 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 511 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Table5: 3D to 2D jumping Neumann CA Rules

The third control register DI is used for jumping between 3D to 2D Neumann, the SI and DI plays the same role as in simple 3D Neumann solution.

### 5.2.6 3D Moore + Neumann Neighbors

In this last proposed scheme we opted to select the all possible neighbors of a 3D cell, i.e. all the Diagonally Adjacent Moore neighbors along with all the Non Diagonally Adjacent Neumann neighbors, In Fig 17. We show the graphical illustration for such a configuration.

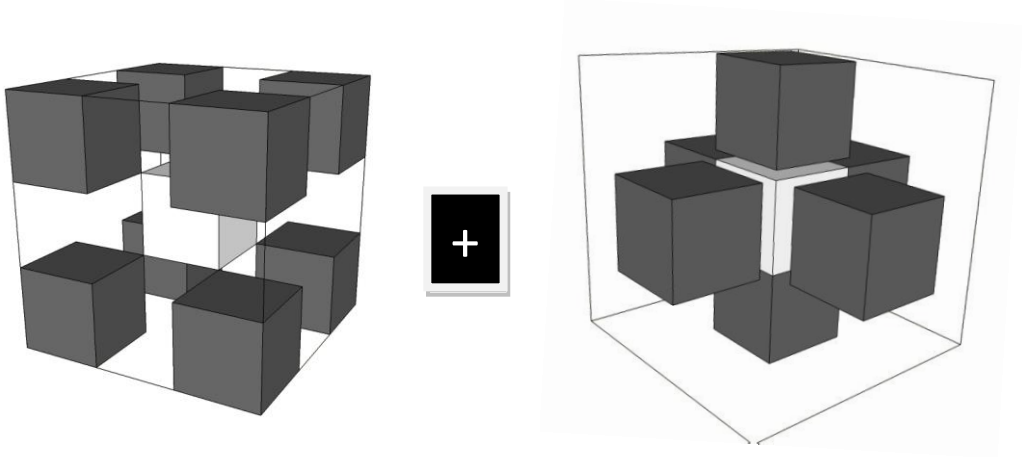


**Figure 17: 3D Moore + Neumann Neighbors Illustration**

Participating Neighbors are

White Cell: Current cell whose value is to be updated.

Gray Neighbors (left): Participating Neighbors (Diagonally Adjacent)

Front Left-Top and Back Right-Bottom, Front Right-Top and Back Left-Bottom, Back Left-Top and Front Right-Bottom, Back Right-Top and Front Left-Bottom.

Gray Neighbors (right): Non Diagonally adjacent Neumann

There are just two control registers used in this scheme the SI and CI i.e. self control and complement control registers with the same roles as they have played in the schemes discussed before.

The Fig. 18 shows the cell structure for 3D Moore + Neumann.

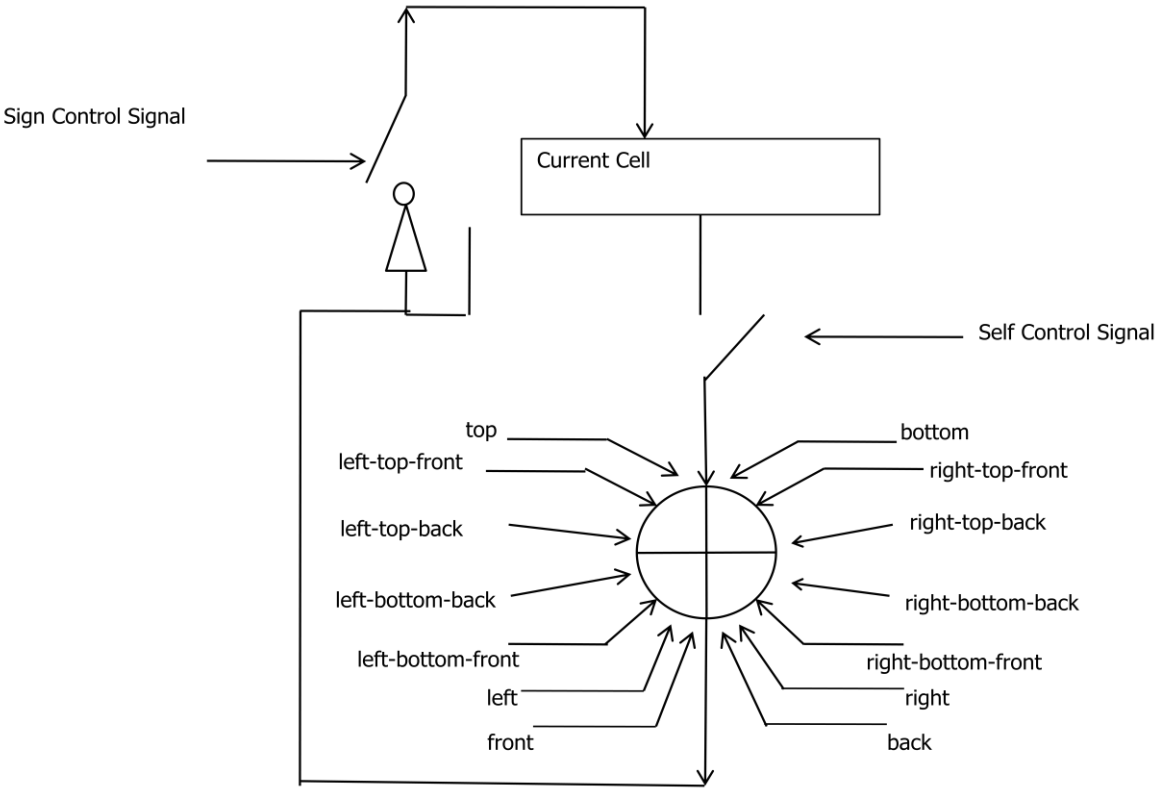



**Figure 18: 3D Moore + Neumann Neighbor Cell Structure**

Table 6. shows the rules used in each iteration

| Rule | CL | SI | Left-Top Front | Left-Bottom Front | Right-Top Front | Right-Bottom Front | Left-Top Back | Left-Bottom Back | Right-Top Back | Right-Bottom Back |
|---|---|---|---|---|---|---|---|---|---|---|
| 16383 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 32767 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 49151 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 65535 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **Rule** | **CL** | **SI** | **Back** | **Front** | **Left** | **Top** | **Right** | **Bottom** | | |
| 16383 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | | |
| 32767 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | |
| 49151 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | | |
| 65535 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | |

.

Table6: 3D Moore + Neumann CA Rules

# Chapter 6 Testing of Proposed Solutions

The Testing is done on DIEHARD suit of Tests; DIEHARD is Internationally Accepted for Randomness Testing and treated as de facto standard for testing RNGs.

As said DIEHARD is suit of tests, it basically comprises of 18 rigorous statistical tests, each test checks the randomness of a series of numbers on a specific parameter defined.

## 6.1 DIEHARD Tests Applied

We applied the following tests of DIEHARD on all the proposed solutions.

- BIRTHDAY SPACINGS TEST
- THE OVERLAPPING 5-PERMUTATION TEST
- BINARY RANK TEST
- THE BITSTREAM TEST
- OPSO(Overlapping-Pairs-Sparse-Occupancy)
- OQSO(Overlapping-Quadruples-Sparse-Occupancy)
- DNA
- COUNT-THE-1's TEST
- PARKING LOT TEST
- THE MINIMUM DISTANCE TEST
- THE 3DSPHERES TEST
- SQEEZE test
- OVERLAPPING SUMS test
- RUNS test
- CRAPS TEST

The internal workings of these tests are available with the DIEHARD documentation and we have briefly discussed them in chapter 2 as well, here we use all the tests as Black-Box testing approach, we will be feeding the random numbers to these tests and then analyze the results obtained.

DIEHARD requires a binary file of 32 bit integers of at least 10MB size i.e. 80 million bits and gives you results in a text based output file.

Most of the tests in DIEHARD return a p-value, which is uniform on [0,1). This means that to qualify for attest the p-value should fall between 0 and 1 otherwise the test is supposed to be failed.

These p-values are obtained by p=F(X), where F is the assumed distribution of the sample random variable X---often normal. But that assumed F is just an asymptotic approximation, for which the fit will be worst in the tails [8].

## 6.2 Testing Process

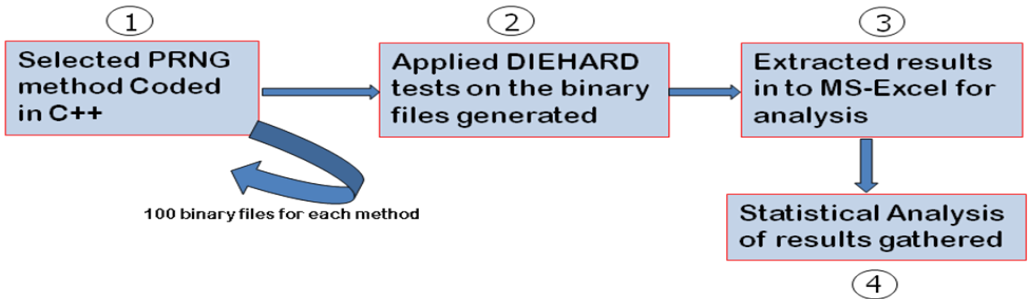
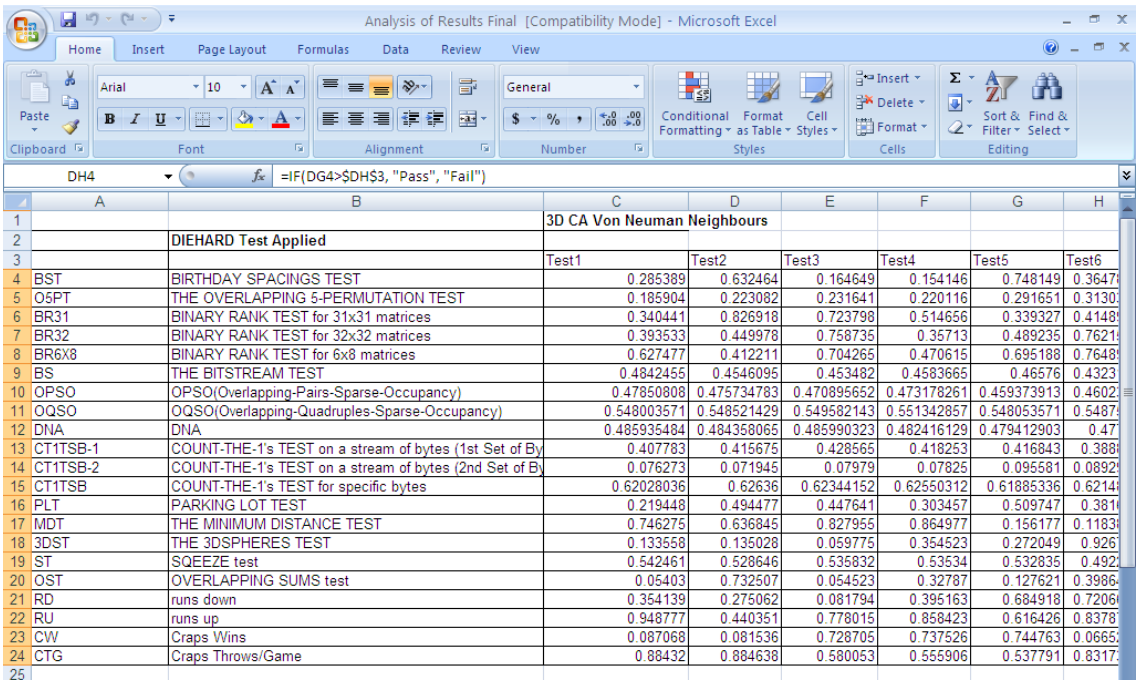



**Figure 19: Testing Phases block diagram**

As Fig. 19 shows we generated 100 binary files for each proposed method we discussed and then applied all the DIEHARD tests on these binary files, in the next phase we extracted results from the DIEHARD produced text files and gathered all the values for analysis.

# Chapter 7 Analysis of Results

First of all we gathered 100 set of values for each method in Excel Sheet,

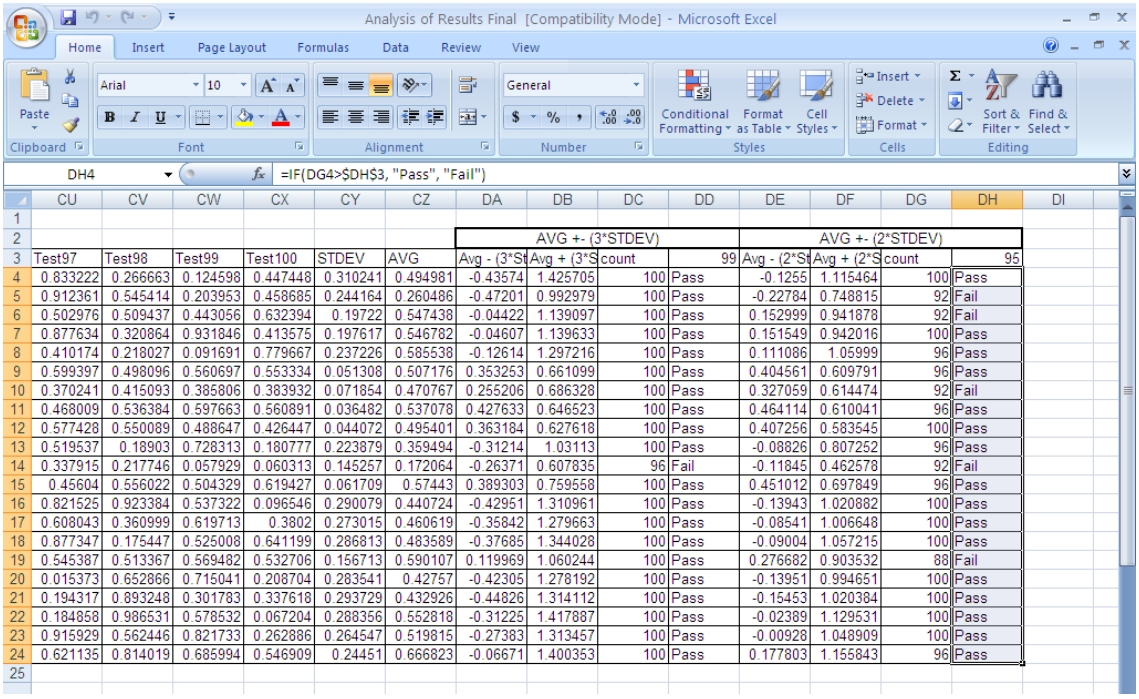


Analysis of Results Final [Compatibility Mode] - Microsoft Excel

=IF(DG4>$DH$3, "Pass", "Fail")

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | | | 3D CA Von Neuman Neighbours | | | | | |
| 2 | | DIEHARD Test Applied | | | | | | |
| 3 | | | Test1 | Test2 | Test3 | Test4 | Test5 | Test6 |
| 4 | BST | BIRTHDAY SPACINGS TEST | 0.285389 | 0.632464 | 0.164649 | 0.154146 | 0.748149 | 0.3647 |
| 5 | O5PT | THE OVERLAPPING 5-PERMUTATION TEST | 0.185904 | 0.223082 | 0.231641 | 0.220116 | 0.291651 | 0.3130 |
| 6 | BR31 | BINARY RANK TEST for 31x31 matrices | 0.340441 | 0.826918 | 0.723798 | 0.514656 | 0.339327 | 0.4148 |
| 7 | BR32 | BINARY RANK TEST for 32x32 matrices | 0.393533 | 0.449978 | 0.758735 | 0.35713 | 0.489235 | 0.7621 |
| 8 | BR6X8 | BINARY RANK TEST for 6x8 matrices | 0.627477 | 0.412211 | 0.704265 | 0.470615 | 0.695188 | 0.7648 |
| 9 | BS | THE BITSTREAM TEST | 0.4842455 | 0.4546095 | 0.453482 | 0.4583665 | 0.46576 | 0.4323 |
| 10 | OPSO | OPSO(Overlapping-Pairs-Sparse-Occupancy) | 0.47850808 | 0.475734783 | 0.470895652 | 0.473178261 | 0.459373913 | 0.4602 |
| 11 | OQSO | OQSO(Overlapping-Quadruples-Sparse-Occupancy) | 0.548003571 | 0.548521429 | 0.549582143 | 0.551342857 | 0.548053571 | 0.5487 |
| 12 | DNA | DNA | 0.485935484 | 0.484358065 | 0.485990323 | 0.482416129 | 0.479412903 | 0.47 |
| 13 | CT1TSB-1 | COUNT-THE-1's TEST on a stream of bytes (1st Set of By | 0.407783 | 0.415675 | 0.428565 | 0.418253 | 0.416843 | 0.388 |
| 14 | CT1TSB-2 | COUNT-THE-1's TEST on a stream of bytes (2nd Set of By | 0.076273 | 0.071945 | 0.07979 | 0.07825 | 0.095581 | 0.0892 |
| 15 | CT1TSB | COUNT-THE-1's TEST for specific bytes | 0.62028036 | 0.62636 | 0.62344152 | 0.62550312 | 0.61885336 | 0.6214 |
| 16 | PLT | PARKING LOT TEST | 0.219448 | 0.494477 | 0.447641 | 0.303457 | 0.509747 | 0.381 |
| 17 | MDT | THE MINIMUM DISTANCE TEST | 0.746275 | 0.636845 | 0.827955 | 0.864977 | 0.156177 | 0.1183 |
| 18 | 3DST | THE 3DSPHERES TEST | 0.133558 | 0.135028 | 0.059775 | 0.354523 | 0.272049 | 0.926 |
| 19 | ST | SQEEZE test | 0.542461 | 0.528646 | 0.535832 | 0.53534 | 0.532835 | 0.492 |
| 20 | OST | OVERLAPPING SUMS test | 0.05403 | 0.732507 | 0.054523 | 0.32787 | 0.127621 | 0.3986 |
| 21 | RD | runs down | 0.354139 | 0.275062 | 0.081794 | 0.395163 | 0.684918 | 0.7206 |
| 22 | RU | runs up | 0.948777 | 0.440351 | 0.778015 | 0.858423 | 0.616426 | 0.8378 |
| 23 | CW | Craps Wins | 0.087068 | 0.081536 | 0.728705 | 0.737526 | 0.744763 | 0.0665 |
| 24 | CTG | Craps Throws/Game | 0.88432 | 0.884638 | 0.580053 | 0.555906 | 0.537791 | 0.8317 |
| 25 | | | | | | | | |

Then we



Analysis of Results Final [Compatibility Mode] - Microsoft Excel

=IF(DG4>$DH$3, "Pass", "Fail")

| | CU | CV | CW | CX | CY | CZ | DA | DB | DC | DD | DE | DF | DG | DH | DI |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | | | | | |
| 2 | | | | | | | AVG +- (3*STDEV) | | | | AVG +- (2*STDEV) | | | | |
| 3 | Test97 | Test98 | Test99 | Test100 | STDEV | AVG | Avg - (3*St | Avg + (3*S | count | 99 | Avg - (2*St | Avg + (2*S | count | 95 | |
| 4 | 0.833222 | 0.266663 | 0.124598 | 0.447448 | 0.310241 | 0.494981 | -0.43574 | 1.425705 | 100 | Pass | -0.1255 | 1.115464 | 100 | Pass | |
| 5 | 0.912361 | 0.545414 | 0.203953 | 0.458685 | 0.244164 | 0.260486 | -0.47201 | 0.992979 | 100 | Pass | -0.22784 | 0.748815 | 92 | Fail | |
| 6 | 0.502976 | 0.509437 | 0.443056 | 0.632394 | 0.19722 | 0.547438 | -0.04422 | 1.139097 | 100 | Pass | 0.152999 | 0.941878 | 92 | Fail | |
| 7 | 0.877634 | 0.320864 | 0.931846 | 0.413575 | 0.197617 | 0.546782 | -0.04607 | 1.139633 | 100 | Pass | 0.151549 | 0.942016 | 100 | Pass | |
| 8 | 0.410174 | 0.218027 | 0.091691 | 0.779667 | 0.237226 | 0.585538 | -0.12614 | 1.297216 | 100 | Pass | 0.111086 | 1.05999 | 96 | Pass | |
| 9 | 0.599397 | 0.498096 | 0.560697 | 0.553334 | 0.051308 | 0.507176 | 0.353253 | 0.661099 | 100 | Pass | 0.404561 | 0.609791 | 96 | Pass | |
| 10 | 0.370241 | 0.415093 | 0.385806 | 0.383932 | 0.071854 | 0.470767 | 0.255206 | 0.686328 | 100 | Pass | 0.327059 | 0.614474 | 92 | Fail | |
| 11 | 0.468009 | 0.536384 | 0.597663 | 0.560891 | 0.036482 | 0.537078 | 0.427633 | 0.646523 | 100 | Pass | 0.464114 | 0.610041 | 96 | Pass | |
| 12 | 0.577428 | 0.550089 | 0.488647 | 0.426447 | 0.044072 | 0.495401 | 0.363184 | 0.627618 | 100 | Pass | 0.407256 | 0.583545 | 100 | Pass | |
| 13 | 0.519537 | 0.18903 | 0.728313 | 0.180777 | 0.223879 | 0.359494 | -0.31214 | 1.03113 | 100 | Pass | -0.08826 | 0.807252 | 96 | Pass | |
| 14 | 0.337915 | 0.217746 | 0.057929 | 0.060313 | 0.145257 | 0.172064 | -0.26371 | 0.607835 | 96 | Fail | -0.11845 | 0.462578 | 92 | Fail | |
| 15 | 0.45604 | 0.556022 | 0.504329 | 0.619427 | 0.061709 | 0.57443 | 0.389303 | 0.759558 | 100 | Pass | 0.451012 | 0.697849 | 96 | Pass | |
| 16 | 0.821525 | 0.923384 | 0.537322 | 0.096546 | 0.290079 | 0.440724 | -0.42951 | 1.310961 | 100 | Pass | -0.13943 | 1.020882 | 100 | Pass | |
| 17 | 0.608043 | 0.360999 | 0.619713 | 0.3802 | 0.273015 | 0.460619 | -0.35842 | 1.279663 | 100 | Pass | -0.08541 | 1.006648 | 100 | Pass | |
| 18 | 0.877347 | 0.175447 | 0.525008 | 0.641199 | 0.286813 | 0.483589 | -0.37685 | 1.344028 | 100 | Pass | -0.09004 | 1.057215 | 100 | Pass | |
| 19 | 0.545387 | 0.513367 | 0.569482 | 0.532706 | 0.156713 | 0.590107 | 0.119969 | 1.060244 | 100 | Pass | 0.276682 | 0.903532 | 88 | Fail | |
| 20 | 0.015373 | 0.652866 | 0.715041 | 0.208704 | 0.283541 | 0.42757 | -0.42305 | 1.278192 | 100 | Pass | -0.13951 | 0.994651 | 100 | Pass | |
| 21 | 0.194317 | 0.893248 | 0.301783 | 0.337618 | 0.293729 | 0.432926 | -0.44826 | 1.314112 | 100 | Pass | -0.15453 | 1.020384 | 100 | Pass | |
| 22 | 0.184858 | 0.986531 | 0.578532 | 0.067204 | 0.288356 | 0.552818 | -0.31225 | 1.417887 | 100 | Pass | -0.02389 | 1.129531 | 100 | Pass | |
| 23 | 0.915929 | 0.562446 | 0.821733 | 0.262886 | 0.264547 | 0.519815 | -0.27383 | 1.313457 | 100 | Pass | -0.00928 | 1.048909 | 100 | Pass | |
| 24 | 0.621135 | 0.814019 | 0.685994 | 0.546909 | 0.24451 | 0.666823 | -0.06671 | 1.400353 | 100 | Pass | 0.177803 | 1.155843 | 96 | Pass | |
| 25 | | | | | | | | | | | | | | | |

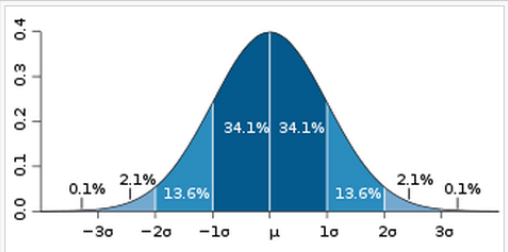
**Figure 20: Snap shot of results**

calculated μ±2σ and μ±3σ for each method, and also calculated number of values falling in Safe Range for each Test (values >=0.25 AND values <=0.75).

## 7.1 Criteria 1

99.6% values should fall in μ±3σ.

95.4% values should fall in μ±2σ.



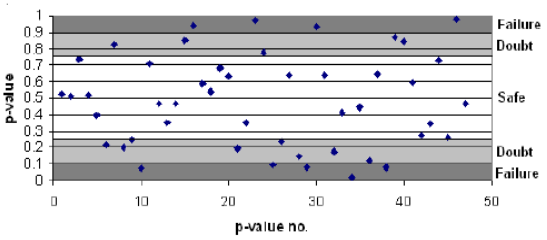
A plot of a normal distribution (or bell curve). Each colored band has a width of one standard deviation.

**Figure 21: Courtesy: en.wikipedia.org/wiki/Standard_deviation**

This criteria is applied as the p-values generated by the DIEHARD approximates to the Normal distribution[2], and it is well known fact that for a normal distribution 99.6% values should fall in μ±3σ and 95.4% values fall in μ±2σ.

## 7.2 Criteria 2

Check number of values falling in Safe Range for each Test (values >=0.25 AND values <=0.75), i.e. out of 100 values per DIEHARD test it shows the count of how many values fall in this safe range.




Applying **Computer Science and** this criterion takes the safest values among the p-values produced by DIEHARD test.

The following figure discusses the analysis when we applied Criteria 1 on the results gathered.

| | No of Passes Avg + - 3 Std. Dev 99% Values Fall | | | | | |
|---|---|---|---|---|---|---|
| | Jumping 2D-3D | Rand3DVonNeuman | 3D Moore + Neuman | 2D Moore | 2D Von Neuman Moore Jumping | 2D Von Neuman |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Fail | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| Failures | 0 | 1 | 0 | 0 | 0 | 0 |

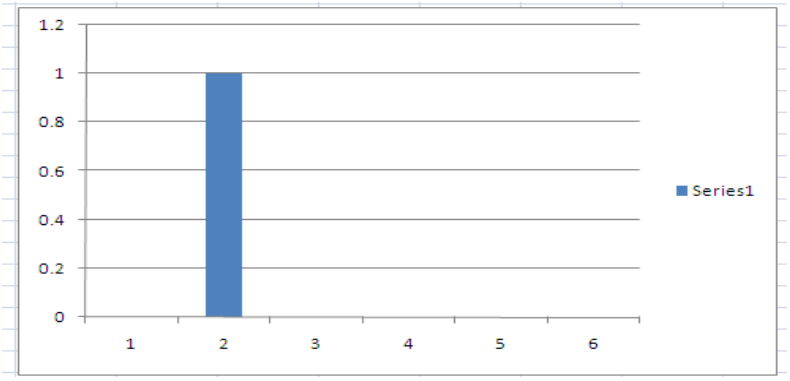



**Figure 23: Results and graph showing values in the range μ±3σ**

An entry of “PASS” in Fig. 23 show that the values for each test falls in the range of μ±3σ, it is clear from the figure that 3D-Neuman case fails in just one case, this analysis does not give us much information as all the methods perform equivalently so we made the selection window further tight by checking it for μ±2σ.

We obtained the following out come when we applied μ±2σ on all the tests for each method.

| | No of Passes Avg + - 2 Std. Dev 95% Values Fall | | | | | |
|---|---|---|---|---|---|---|
| | Jumping 2D-3D | Rand3DVonNeuman | 3D Moore + Neuman | 2D Moore | 2D Von Neuman Moore Jumping | 2D Von Neuman |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Fail | Fail | Pass | Pass | Pass | Pass |
| | Pass | Fail | Pass | Pass | Pass | Pass |
| | Fail | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Fail | Pass | Pass | Pass | Pass | Pass |
| | Fail | Fail | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Fail | Pass | Pass | Pass | Fail |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Fail | Pass | Pass | Pass |
| | Pass | Fail | Pass | Pass | Pass | Fail |
| | Pass | Pass | Pass | Fail | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Fail | Pass | Pass |
| Failures | 4 | 5 | 1 | 2 | 0 | 2 |

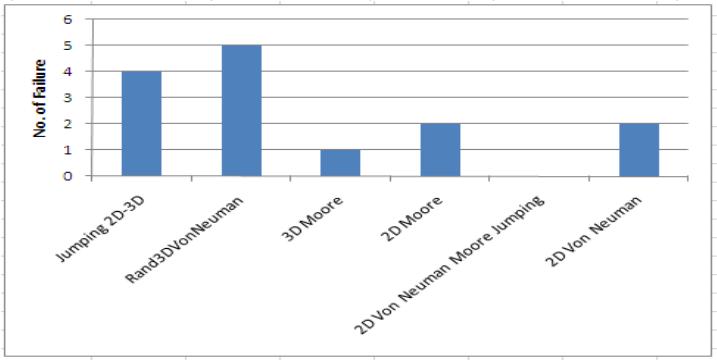



**Figure 24: Results and graph showing values in the range μ±2σ**

This time we get two favorable methods, 2D Neumann – Moore jumping has got zero failures while 3D Moore + Neumann has just one failure, all the other methods are more failures with 3D Neumann has the maximum failures.

Then we applied criteria 2 to check among all the 100 values per method for each test falling in the safe range i.e. (values >=0.25 AND values <=0.75), this type of analysis considers all the p-values instead of using average and standard deviation of all the values.

Fig 25. Discusses the results obtained using this approach.

All 100 values Comparison (values >=0.25 AND values <=0.75) Safe Range

| | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|
| | Jumping 2D-3D | Rand 3D Von Neuman | 3D Moore + Neuman | 2D Moore Neighbours | 2D Von Neuman Moore Jump N | 2D Von Neuman Neighbors | Max Value Series |
| BST | 68 | 40 | 56 | 52 | 44 | 56 | 1 |
| O5PT | 28 | 24 | 36 | 36 | 36 | 28 | 3, 4, 5 |
| BR31 | 80 | 84 | 84 | 72 | 68 | 0 | 2 , 3 |
| BR32 | 84 | 76 | 72 | 76 | 80 | 0 | 1 |
| BR6X8 | 60 | 60 | 44 | 56 | 52 | 36 | 1 , 2 |
| BS | 100 | 100 | 100 | 100 | 100 | 8 | 1, 2, 3, 4, 5 |
| OPSO | 100 | 100 | 100 | 76 | 80 | 100 | 1, 2, 3, 6 |
| OQSO | 100 | 100 | 100 | 76 | 80 | 100 | 1, 2, 3, 6 |
| DNA | 100 | 100 | 100 | 76 | 80 | 100 | 1, 2, 3, 6 |
| CT1TSB-1 | 40 | 52 | 48 | 40 | 80 | 28 | 5 |
| CT1TSB-2 | 48 | 20 | 72 | 60 | 68 | 40 | 3 |
| CT1TSB | 100 | 100 | 100 | 100 | 100 | 100 | 1, 2, 3, 4, 5, 6 |
| PLT | 48 | 40 | 40 | 52 | 60 | 64 | 6 |
| MDT | 48 | 64 | 52 | 44 | 28 | 32 | 2 |
| 3DST | 36 | 48 | 48 | 72 | 52 | 48 | 4 |
| ST | 44 | 84 | 48 | 40 | 44 | 0 | 2 |
| OST | 56 | 56 | 52 | 48 | 52 | 60 | 6 |
| RD | 64 | 48 | 56 | 56 | 56 | 60 | 1 |
| RU | 40 | 48 | 52 | 44 | 40 | 52 | 3, 6 |
| CW | 36 | 64 | 56 | 24 | 52 | 64 | 2, 6 |
| CTG | 60 | 48 | 44 | 64 | 52 | 56 | 4 |

| | Jumping 2D-3D | Rand 3D Von Neuman | 3D Moore + Neuman | 2D Moore Neighbours | 2D Von Neuman Moore Jump N | 2D Von Neuman Neighbors |
|---|---|---|---|---|---|---|
| Max Occurance | 9 | 10 | 9 | 5 | 4 | 8 |

**Figure 25: Values falling in the Safe Range (values >=0.25 AND values <=0.75)**

The right most columns shows which method qualifies maximum number of times in each test e.g. 2D-3D jumping Neumann qualifies with the 68 tests, so the right most column represents this information, similarly we calculated for all the methods and all the tests. At the bottom the summary is shown, 3D Von-Neumann appears 10 times in the rightmost column while 3D Moore + Neumann appears 9 times, so these two methods are the candidates from the above figure.

## 7.3 Comparative Analysis of Results

So we have the following results as best performing candidates.

Criteria 1: (Tests passing in $\mu \pm 2\sigma$ range)

- 2D Neumann – Moore jumping
- 3D Moore + Neumann

Criteria 2: (Values falling in the Safe Range, values >=0.25 AND values <=0.75)

- 3D Neumann
- 3D Moore + Neumann

2D Moore-Neumann jumping does not perform well in Criteria 2 as it occurs only at 4 places, similarly 3D Neumann which is a candidate in Criteria 2 does not perform well in Criteria 1 by failing at 5 places, so we can safely conclude that 3D Moore + Neumann is the best performing method among all the methods we discussed.

## 7.4 Run Time Analysis of the Proposed Methods

The criteria 1 and criteria 2 discussed in sections 5.1 and 5.2 solely discuss the issue with respect to the quality of random numbers generated as we compared the DIEHARD produced values and applied certain statistical operations on the results, the discussion remains incomplete if we do not compare the run time efficiency of methods among each other.

The basic point is to measure the cost we have to pay in increasing the quality of randomness of a sequence i.e. how much we have to pay in terms of number of operations to get a better random numbers sequence.

To make the things simpler we can divide the methods in to three categories,

1. Working on pure 2D structures
    - 2D Neumann Neighbors
    - 2D Moore + Neumann Neighbors
    - 2D Neumann Moore jumping Neighbors

2. Working on pure 3D structures
    - 3D Neumann Neighbors
    - 3D Moore + Neumann Neighbors

3. Working on 3D to 2D transition structures
    - 3D to 2D Jumping Neumann Neighbors

### 7.4.1 Main Memory Requirements

Memory Requirement for pure 2D structures is common for all as we use the same data structure for all 2D cases; the difference is just in picking neighbors.

One integer takes 4 bytes, so a matrix of 8 x 8 i.e. 64 integers, would take 64 x 4 = 256 bytes of main memory.

For pure 3D cases and transition cases we used the grid of 8 x 8 x 8 i.e. 512 integers, would take 2048 bytes of main memory.

The memory requirements will remain constant during complete execution as after each iteration the generated numbers are flushed to the binary file.

### 7.4.2 Number of operations per iteration

For 2D cases we have to go through each cell thus requiring a two level nested loops, outer for rows and inner for columns, plus we need to pick 4 neighbors in case of Neumann, and 8 neighbors in case of combination of Neumann and Moore, similarly Neumann Moore jumping also requires just 4 neighbors to be picked (either Neumann's or Moore's), the cost of XOR operation and writing to binary file remains same for all the cases, thus for a 2D Neumann PCA with Self Cell participating we can calculate the running time as follows,

| **Statement** | **Cost** | **Time** |
|---|---|---|
| for (int rows = 1; rows <=8; rows++) | C1 | m |
| for (int cols = 1; cols<=8; cols++) | C2 | n |
| Pick all the five values, the white cell + all Gray neighbors | C3 | 5mn |
| Find XOR (five values) | C4 | 5mn |
| Update the White Cell | C5 | mn |

The temporary variables used to store intermediate result represent the intermediate cost and that will be constant for all the cases with no significant increase in the run time of algorithm.

Exact Running Time/ Time Complexity = c1m + c2n + 5c3mn + 5c4mn + c5mn

Order of Growth= O(mn)

As we have worked on 8 x 8 grid we have, m = n therefore Order of Growth is O(n2), this will be the case for all the 2D cases as the maximum order of growth will not go beyond O(n2).

For 3D structures we have to apply 3 nested loops, first for the plane, second for the rows and third for the columns to iterate through each cell, the Neumann neighbors of each 3D cell be 6 while for Neumann + Moore the neighbors are 10 in number.

Thus for 3D Neumann we can calculate the cost as follows

| Statement | Cost | Time |
|---|---|---|
| for (int planes = 1; planes <=8; planes++) | C0 | p |
| for (int rows = 1; rows <=8; rows++) | C1 | m |
| for (int cols = 1; cols<=8; cols++) | C2 | n |
| Pick all the seven values, the white cell + all Gray neighbors | C3 | 7pmn |
| Find XOR (seven values) | C4 | 7pmn |
| Update the White Cell | C5 | pmn |

Exact Running Time = c0p + c1m + c2n + 7c3pmn + 7c4pmn + c5pmn

Order of Growth= O(pmn)

As we have worked on 8 x 8 x 8 grid we have, p = m = n therefore Order of Growth is O(n3), similarly all the 3D cases will have the same order of growth as maximum order of growth will not jump beyond O(n3).

In case of 3D to 2D Jumping Neumann Neighbors, it is expected that 50% of the time it behaves like 3D and 50% of time behaves like 2D, thus the running cost will be calculated by 0.5 x Cost 2D Neumann + 0.5 x Cost 3D Neumann.

Table 7 summarizes the run time calculations and order of growth for all the proposed methods.

| Method | Run time = No. of Operations per iteration x Cost | Order of Growth |
|---|---|---|
| 2D Neumann | c1m + c2n + 5c3mn + 5c4mn + c5mn | $O(N^2)$ |
| 2D Neumann Moore jumping | c1m + c2n + 5c3mn + 5c4mn + c5mn | $O(N^2)$ |
| 2D Moore + Neumann | c1m + c2n + 9c3mn + 9c4mn + c5mn | $O(N^2)$ |
| 3D Neumann | c0p + c1m + c2n + 7c3pmn + 7c4pmn + c5pmn | $O(N^3)$ |
| 3D Moore+ Neuman | c0p + c1m + c2n + 15c3pmn + 15c4pmn + c5pmn | $O(N^3)$ |
| 3D to 2D jumping Neumann | 0.5*( c0p + c1m + c2n + 7c3pmn + 7c4pmn + c5pmn) + 0.5*( c1m + c2n + 5c3mn + 5c4mn + c5mn) | $O(N^3)$ half of iterations<br>$O(N^2)$ half of iterations |

Table7: Run time cost and order of growth

# Chapter 8 Conclusion and Future Work

## 8.1 Conclusion

After implementing all the six methods and analysis of DIEHARD results we applied the two criteria discussed in the previous section, the first criteria is about checking number of values falling in the range of standard deviation from twice of mean, while the second criteria is to find all the values passing in the safe window i.e. in between the range of 0.25 and 0.75.

Applying the discussed criteria, the most successful candidate with maximum number of values falling in the safe range and just one less than the maximum in the $\mu \pm 2\sigma$ range is 3D Moore + Neumann, the computational cost of this method is just $O(N^3)$ with 2048MB memory requirements.

As concluding remarks we suggest 3D Moore + Neumann method for better quality random numbers generation using programmable cellular automata approach, it combines the properties of both Neumann and Moore neighbors at the same time in a three dimensional space.

This also suggests that in general the more number of non-correlated neighbors you pick the more randomness you get but at the cost of increased processing time.

Although we have chosen the best technique in terms of quality of randomness but the requirements of application at hand could be a deciding factor as some applications might require lesser randomness quality but should be efficient, if this is the case then the detail results provided in the appendix and analysis chapter can be consulted to pick the better suited PRNG.

## 8.2 Future Work

The work done in this research study can be further extended using the following potential research paths.

1. Implementation of other structural organizations that are not explored, various other combinations of Moore and Neumann and their joint schemes can be explored for better results.
2. Strategies proposed in this work can be tested for a better designed algorithm working with different logical operations other than XOR.
3. Implementing the same concepts for r + 2 adjacent neighbors, the whole literature survey suggests that all of the researchers went for r + 1 neighbors i.e. immediately adjacent neighbors of the current cell, no work is found on r + 2 or r + 3 neighbors, what effect would be on the results if we consider a larger set of neighbors.
4. Testing the performance of presented solutions on Hardware implementations, specifically the run time efficiencies and complexities involved if the solution is implemented in FPGA.

The points discussed above shows that there is a lot of potential in the field of Random Numbers Generation using Cellular Automata, one can easily opt any path and may find better results in terms of quality of random numbers and in terms of efficiency of a generator.

## References


*1. P. Hellekalek*, “Good Random Number Generators are (not so) Easy to Find”, *Mathematics and Computing Simulation, 46(5-6), pp 485-505 (1998).*

*2.* A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, NIST (national Institute of Standards and Technology), *Special Publication 800-22 Revision 1a.*

*3. Niloy Ganguly et al.*A Survey on Cellular Automata*, Center for High Performance Computing, Dresden University of Technology, Dresden, Germany.*

*4. S. Nandi and P. P. Chaudhuri*, “Analysis of periodic and intermediate boundary 90/150 cellular automata,” *IEEE Trans. Comput., vol. 45, Jan. 1999.*

*5. Su Guan* “An evolutionary approach to the design of controllable cellular automata structure for random number generation”. *IEEE Trans. Evolutionary Computation, vol. 7, no. 1, pp. 23–36, Feb. 2003*

*6. Tomassini, M., Sipper, M., Perrenoud, M., “* On the generation of high quality random numbers by two-dimensional cellular automata*.” IEEE Transactions on Computers 49, 1146–1151 (2000)*

*7. Guan, S.-U., Zhang, S., Quieta, M.T*, “2-D Variation With Asymmetric Neighborship for Pseudorandom Number Generation*.” IEEE Transaction on Computers 23, 378–388 (2004).*

*8.* The Marsaglia Random Number CDROM, with The Diehard Battery of Tests of Randomness, produced at Florida State University under a grant from The National Science Foundation (1985).

*9. Dr.Mohammed M. Alani,* “Testing Randomness in Ciphertext of Block-Ciphers Using DieHard Tests *“, IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.4, April 2010.*

*10. Sheng-Uei Guan and Syn Kiat Tan,* “Pseudorandom Number Generation With Self-Programmable Cellular Automata”, *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 23, NO. 7, JULY 2004*

*11. Byung-Heon Kang, Dong-Ho Lee, and Chun-Pyo Hon,* “High-Performance Pseudorandom Number Generator Using Two-Dimensional Cellular Automata”*, 4th IEEE International Symposium on Electronic Design, Test & Applications*

*12. Sang-Ho Shin, Geum-Dal Park2, and Kee-Young Yoo*, “A Virtual Three-Dimension Cellular Automata Pseudorandom Number Generator Based on the Moore Neighborhood Method*”, D.-S. Huang et al. (Eds.): ICIC 2008, LNAI 5227, pp. 174–181, 2008. Springer-Verlag Berlin Heidelberg 2008*

*13. Sang-Ho Shin, Kee-Young Yoo*, “Analysis of 2-State, 3-Neighborhood Cellular Automata Rules for Cryptographic Pseudorandom Number Generation”, *2009 International Conference on Computational Science and Engineering, 978-0-7695-3823-5/09, 2009 IEEE DOI 0.1109/CSE.2009.299*

# Appendix A

Important piece of code explained here (for 3D Moore + Neumann Method)

Data structure to hold the random numbers and register values for the SI and DI control registers

```
int random_array [8][8][8];

int final_randNo [8][8][8];

int SI[64];

int CI[64];
```

*The master loop calling doing the initialization phase and calling function which generates the random number sequences.*

```
for (iteration =1; iteration<=cycles; iteration++)
        {
                initialize_array();
                initialize_SI();
                initialize_CI();
                generate_random();          //This call gives 64bits of Random
        }
```

***The generate_random() function,***

```
void generate_random(void)
{
int SI_val, CI_val;
   int control_index=0;

        int plane, row, col;
  for (plane=0; plane<=7; plane++)
  {
                for (row=0; row<=7; row++)
                {
```

```
                    for (col=0; col<=7; col++)
                    {
                SI_val = SI[control_index];
                CI_val = CI[control_index];
                control_index++;
                if (control_index == 64) control_index =0;
            if (SI_val == 0 && CI_val == 0)
                final_randNo[plane][row][col]= applyRule1023(plane,row, col);
            if (SI_val == 1 && CI_val == 0)
                final_randNo[plane][row][col]=  applyRule2047(plane,row, col);
            if (SI_val == 0 && CI_val == 1)
                final_randNo[plane][row][col]=  applyRule3071(plane,row, col);
            if (SI_val == 1 && CI_val == 1)
                final_randNo[plane][row][col]=  applyRule4095(plane,row, col);

            }       }       }
```

*The Cellular Automata Rules Applied (as an example coding for rule 1023 is shown)*

```
int applyRule1023(int plane, int row, int col)
{
        //Pick 10NB XOR them and update cell val, 6 Normal + 4 Diagonal
        int n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, val, val1, val2, val3, val4, val5;
        int temprow, tempcol, tempplane;
        int tempval1, tempval2, tempval3, tempval4, tempval5;
        int val6, val7, val8, val9, val10;
          if (col==0)
                n1 = final_randNo[plane][row][7];
          else
```

```
                n1 = final_randNo[plane][row][col-1];
            if (col==7)
                n2 = final_randNo[plane][row][0];
            else
                n2 = final_randNo[plane][row][col+1];
            if (row==0)
                n3 = final_randNo[plane][7][col];
            else
                n3 = final_randNo[plane][row-1][col];
            if (row==7)
                n4 = final_randNo[plane][0][col];
            else
                n4 = final_randNo[plane][row+1][col];
           if (plane==0)
                n5 = final_randNo[7][row][col];
            else
                n5 = final_randNo[plane-1][row][col];
            if (plane==7)
                n6 = final_randNo[0][row][col];
            else
                n6 = final_randNo[plane+1][row][col];
// Calculating Indexes for Diagonal Neighbors
            //Left Top Diagonal
            temprow = row-1;
            tempcol = col-1;
            tempplane = plane-1;
            if (temprow<0) temprow=7;
            if (tempcol<0) tempcol=7;
```

```
if (tempplane<0) tempplane = 7;
n7 = final_randNo[tempplane][temprow][tempcol];
//Left Bottom Diagonal
temprow = row+1;
tempcol = col-1;
tempplane = plane+1;
if (temprow>7) temprow=0;
if (tempcol<0) tempcol=7;
if (tempplane>7) tempplane = 0;
n8 = final_randNo[tempplane][temprow][tempcol];
//Right Top Diagonal
temprow = row-1;
tempcol = col+1;
tempplane= plane-1;
if (temprow<0) temprow=7;
if (tempcol>7) tempcol=0;
if (tempplane<0) tempplane = 7;
n9 = final_randNo[tempplane][temprow][tempcol];
//Right Bottom Diagonal
temprow = row+1;
tempcol = col+1;
tempplane = plane+1;
if (temprow>7) temprow=0;
if (tempcol>7) tempcol=0;
if (tempplane>7) tempplane =0;
n10 = final_randNo[tempplane][temprow][tempcol];
val1 = calc_xor(n1, n2);
val2 = calc_xor(n3, n4);
```

```
        val3 = calc_xor(n5, n6);
        val4 = calc_xor(n7, n8);
        val5 = calc_xor(n9, n10);
        tempval1 =  calc_xor(val1, val2);
        tempval2 = calc_xor(val3, val4);
        tempval3 = calc_xor(val5, val6);
        tempval4 = calc_xor(val7, val8);
        tempval5 = calc_xor(val9, val10);
        val =  calc_xor(tempval1, tempval2);
        val =  calc_xor(val, tempval3);
        val =  calc_xor(val, tempval4);
        val =  calc_xor(val, tempval5);
        return val;

}
```

# Appendix B

**2D Neumann Neighbors Results for $\mu \pm 3\sigma$**

| **DIEHARD Test Applied** | Avg - (3*Stdev) | Avg + (3*Stdev) | Test falling in the Range |
|---|---|---|---|
| BIRTHDAY SPACINGS TEST | -0.37901336 | 1.4573814 | Pass |
| THE OVERLAPPING 5-PERMUTATION TEST | -0.468926744 | 1.577111184 | Pass |
| BINARY RANK TEST for 31x31 matrices | 1 | 1 | Pass |
| BINARY RANK TEST for 32x32 matrices | 1 | 1 | Pass |
| BINARY RANK TEST for 6x8 matrices | -0.468123509 | 1.387819949 | Pass |
| THE BITSTREAM TEST | 0.64739095 | 0.96889153 | Pass |
| OPSO(Overlapping-Pairs-Sparse-Occupancy) | 0.314224618 | 0.747193078 | Pass |
| OQSO(Overlapping-Quadruples-Sparse-Occupancy) | 0.356984112 | 0.709225745 | Pass |
| DNA | 0.357390793 | 0.603823885 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (1st Set of Bytes) | -0.277511751 | 1.565505631 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (2nd Set of Bytes) | -0.019845513 | 1.435136513 | Pass |
| COUNT-THE-1's TEST for specific bytes | 0.386797919 | 0.716937619 | Pass |
| PARKING LOT TEST | -0.229618712 | 1.165533792 | Pass |
| THE MINIMUM DISTANCE TEST | -0.228734204 | 1.576438724 | Pass |
| THE 3DSPHERES TEST | -0.495386736 | 1.319387096 | Pass |
| SQEEZE test | 0.735636176 | 1.159628904 | Pass |
| OVERLAPPING SUMS test | -0.361506123 | 1.304533283 | Pass |
| runs down | -0.165312854 | 1.232009374 | Pass |
| runs up | -0.392507335 | 1.248203295 | Pass |
| Craps Wins | 0.167254054 | 1.207466546 | Pass |
| Craps Throws/Game | -0.280681021 | 1.254041701 | Pass |

**2D Neumann Neighbors Results for $\mu \pm 2\sigma$**

| **DIEHARD Test Applied** | Avg - (2*Stdev) | Avg + (2*Stdev) | Test falling in the Range |
|---|---|---|---|
| BIRTHDAY SPACINGS TEST | -0.072947567 | 1.151315607 | Pass |
| THE OVERLAPPING 5-PERMUTATION TEST | -0.127920423 | 1.236104863 | Pass |
| BINARY RANK TEST for 31x31 matrices | 1 | 1 | Pass |
| BINARY RANK TEST for 32x32 matrices | 1 | 1 | Pass |
| BINARY RANK TEST for 6x8 matrices | -0.158799599 | 1.078496039 | Pass |
| THE BITSTREAM TEST | 0.70097438 | 0.9153081 | Pass |
| OPSO(Overlapping-Pairs-Sparse-Occupancy) | 0.386386028 | 0.675031668 | Pass |
| OQSO(Overlapping-Quadruples-Sparse-Occupancy) | 0.415691051 | 0.650518806 | Pass |
| DNA | 0.398462975 | 0.562751703 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (1st Set of Bytes) | 0.029657813 | 1.258336067 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (2nd Set of Bytes) | 0.222651491 | 1.192639509 | Fail |
| COUNT-THE-1's TEST for specific bytes | 0.441821202 | 0.661914335 | Pass |
| PARKING LOT TEST | 0.002906705 | 0.933008375 | Pass |
| THE MINIMUM DISTANCE TEST | 0.07212795 | 1.27557657 | Pass |
| THE 3DSPHERES TEST | -0.192924431 | 1.016924791 | Pass |
| SQEEZE test | 0.806301631 | 1.088963449 | Fail |
| OVERLAPPING SUMS test | -0.083832889 | 1.026860049 | Pass |
| runs down | 0.067574184 | 0.999122336 | Pass |
| runs up | -0.119055563 | 0.974751523 | Pass |
| Craps Wins | 0.340622803 | 1.034097797 | Pass |
| Craps Throws/Game | -0.024893901 | 0.998254581 | Pass |

**2D Moore Neighbors Results for $\mu \pm 3\sigma$**

| **DIEHARD Test Applied** | Avg - (3*Stdev) | Avg + (3*Stdev) | Test falling in the Range |
|---|---|---|---|
| BIRTHDAY SPACINGS TEST | -0.405930619 | 1.281688619 | Pass |
| THE OVERLAPPING 5-PERMUTATION TEST | -0.450601672 | 1.496250472 | Pass |
| BINARY RANK TEST for 31x31 matrices | -0.03108691 | 1.26347067 | Pass |
| BINARY RANK TEST for 32x32 matrices | -0.04078692 | 1.24184764 | Pass |
| BINARY RANK TEST for 6x8 matrices | -0.314643493 | 1.242396373 | Pass |
| THE BITSTREAM TEST | 0.308041156 | 0.707091764 | Pass |
| OPSO(Overlapping-Pairs-Sparse-Occupancy) | -0.064900167 | 1.284560629 | Pass |
| OQSO(Overlapping-Quadruples-Sparse-Occupancy) | -0.132204355 | 1.298127153 | Pass |
| DNA | -0.104396892 | 1.290750123 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (1st Set of Bytes) | -0.442939472 | 1.516708272 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (2nd Set of Bytes) | 0.135512226 | 1.278446094 | Pass |
| COUNT-THE-1's TEST for specific bytes | 0.372058966 | 0.662507546 | Pass |
| PARKING LOT TEST | -0.347835562 | 1.279498442 | Pass |
| THE MINIMUM DISTANCE TEST | -0.41950569 | 1.35722649 | Pass |
| THE 3DSPHERES TEST | -0.240050889 | 1.099710969 | Pass |
| SQEEZE test | -0.369980803 | 0.944059443 | Pass |
| OVERLAPPING SUMS test | -0.173670267 | 1.381949307 | Pass |
| runs down | -0.321742978 | 1.240430418 | Pass |
| runs up | -0.341292964 | 1.450865524 | Pass |
| Craps Wins | -0.408912737 | 1.698362897 | Pass |
| Craps Throws/Game | -0.231027673 | 1.137971113 | Pass |

**2D Moore Neighbors Results for $\mu \pm 2\sigma$**

| **DIEHARD Test Applied** | Avg - (2*Stdev) | Avg + (2*Stdev) | Test falling in the Range |
|---|---|---|---|
| BIRTHDAY SPACINGS TEST | -0.124660746 | 1.000418746 | Pass |
| THE OVERLAPPING 5-PERMUTATION TEST | -0.126126315 | 1.171775115 | Pass |
| BINARY RANK TEST for 31x31 matrices | 0.184672687 | 1.047711073 | Pass |
| BINARY RANK TEST for 32x32 matrices | 0.172985507 | 1.028075213 | Pass |
| BINARY RANK TEST for 6x8 matrices | -0.055136849 | 0.982889729 | Pass |
| THE BITSTREAM TEST | 0.374549591 | 0.640583329 | Pass |
| OPSO(Overlapping-Pairs-Sparse-Occupancy) | 0.160009965 | 1.059650496 | Pass |
| OQSO(Overlapping-Quadruples-Sparse-Occupancy) | 0.106184229 | 1.059738568 | Pass |
| DNA | 0.12812761 | 1.05822562 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (1st Set of Bytes) | -0.116331515 | 1.190100315 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (2nd Set of Bytes) | 0.326001204 | 1.087957116 | Pass |
| COUNT-THE-1's TEST for specific bytes | 0.420467063 | 0.614099449 | Pass |
| PARKING LOT TEST | -0.076613228 | 1.008276108 | Pass |
| THE MINIMUM DISTANCE TEST | -0.12338366 | 1.06110446 | Pass |
| THE 3DSPHERES TEST | -0.016757246 | 0.876417326 | Pass |
| SQEEZE test | -0.150974095 | 0.725052735 | Pass |
| OVERLAPPING SUMS test | 0.085599662 | 1.122679378 | Fail |
| runs down | -0.061380745 | 0.980068185 | Pass |
| runs up | -0.042599883 | 1.152172443 | Pass |
| Craps Wins | -0.057700131 | 1.347150291 | Pass |
| Craps Throws/Game | -0.002861209 | 0.909804649 | Fail |

**2D Neumann + Moore jumping Neighbors Results for $\mu \pm 3\sigma$**

| **DIEHARD Test Applied** | Avg - (3*Stdev) | Avg + (3*Stdev) | Test falling in the Range |
|---|---|---|---|
| BIRTHDAY SPACINGS TEST | -0.492415371 | 1.458226971 | Pass |
| THE OVERLAPPING 5-PERMUTATION TEST | -0.477770625 | 1.471587345 | Pass |
| BINARY RANK TEST for 31x31 matrices | 0.002145266 | 1.208466334 | Pass |
| BINARY RANK TEST for 32x32 matrices | -0.092559753 | 1.117303353 | Pass |
| BINARY RANK TEST for 6x8 matrices | -0.439251463 | 1.294689783 | Pass |
| THE BITSTREAM TEST | 0.385449873 | 0.681850207 | Pass |
| OPSO(Overlapping-Pairs-Sparse-Occupancy) | 0.283655443 | 1.150127121 | Pass |
| OQSO(Overlapping-Quadruples-Sparse-Occupancy) | -0.073994631 | 1.221349162 | Pass |
| DNA | -0.056520442 | 1.22619266 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (1st Set of Bytes) | 0.096587341 | 1.129198019 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (2nd Set of Bytes) | -0.158886658 | 1.336438418 | Pass |
| COUNT-THE-1's TEST for specific bytes | 0.344986305 | 0.687405765 | Pass |
| PARKING LOT TEST | -0.292434096 | 1.363166336 | Pass |
| THE MINIMUM DISTANCE TEST | -0.253056069 | 1.553231349 | Pass |
| THE 3DSPHERES TEST | -0.312896071 | 1.367575591 | Pass |
| SQEEZE test | -0.007536914 | 1.381652994 | Pass |
| OVERLAPPING SUMS test | -0.205477795 | 1.374255955 | Pass |
| runs down | -0.332370843 | 1.201641243 | Pass |
| runs up | -0.381712114 | 1.309248594 | Pass |
| Craps Wins | -0.356770545 | 1.352534625 | Pass |
| Craps Throws/Game | -0.125391412 | 1.445069812 | Pass |

**2D Neumann + Moore jumping Neighbors Results for $\mu \pm 2\sigma$**

| **DIEHARD Test Applied** | Avg - (2*Stdev) | Avg + (2*Stdev) | Test falling in the Range |
|---|---|---|---|
| BIRTHDAY SPACINGS TEST | -0.167308314 | 1.133119914 | Pass |
| THE OVERLAPPING 5-PERMUTATION TEST | -0.15287763 | 1.14669435 | Pass |
| BINARY RANK TEST for 31x31 matrices | 0.203198777 | 1.007412823 | Pass |
| BINARY RANK TEST for 32x32 matrices | 0.109084098 | 0.915659502 | Pass |
| BINARY RANK TEST for 6x8 matrices | -0.150261255 | 1.005699575 | Pass |
| THE BITSTREAM TEST | 0.434849929 | 0.632450151 | Pass |
| OPSO(Overlapping-Pairs-Sparse-Occupancy) | 0.428067389 | 1.005715175 | Pass |
| OQSO(Overlapping-Quadruples-Sparse-Occupancy) | 0.141896001 | 1.00545853 | Pass |
| DNA | 0.157265075 | 1.012407143 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (1st Set of Bytes) | 0.26868912 | 0.95709624 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (2nd Set of Bytes) | 0.090334188 | 1.087217572 | Pass |
| COUNT-THE-1's TEST for specific bytes | 0.402056215 | 0.630335855 | Pass |
| PARKING LOT TEST | -0.016500691 | 1.087232931 | Pass |
| THE MINIMUM DISTANCE TEST | 0.047991834 | 1.252183446 | Pass |
| THE 3DSPHERES TEST | -0.032817461 | 1.087496981 | Pass |
| SQEEZE test | 0.223994737 | 1.150121343 | Pass |
| OVERLAPPING SUMS test | 0.057811163 | 1.110966997 | Pass |
| runs down | -0.076702162 | 0.945972562 | Pass |
| runs up | -0.099885329 | 1.027421809 | Pass |
| Craps Wins | -0.07188635 | 1.06765043 | Pass |
| Craps Throws/Game | 0.136352125 | 1.183326275 | Pass |

**Jumping 2D-3D Neumann Neighbors Results for $\mu \pm 3\sigma$**

| **DIEHARD Test Applied** | Avg - (3*Stdev) | Avg + (3*Stdev) | Test falling in the Range |
|---|---|---|---|
| BIRTHDAY SPACINGS TEST | -0.284166685 | 1.269533085 | Pass |
| THE OVERLAPPING 5-PERMUTATION TEST | -0.236278358 | 1.635656958 | Pass |
| BINARY RANK TEST for 31x31 matrices | -0.13134028 | 1.1971842 | Pass |
| BINARY RANK TEST for 32x32 matrices | -0.028839558 | 1.124761558 | Pass |
| BINARY RANK TEST for 6x8 matrices | -0.185201521 | 1.279951361 | Pass |
| THE BITSTREAM TEST | 0.310501995 | 0.612340325 | Pass |
| OPSO(Overlapping-Pairs-Sparse-Occupancy) | 0.353045146 | 0.746977115 | Pass |
| OQSO(Overlapping-Quadruples-Sparse-Occupancy) | 0.398038539 | 0.645760318 | Pass |
| DNA | 0.372102329 | 0.68452451 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (1st Set of Bytes) | -0.552849191 | 1.344949431 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (2nd Set of Bytes) | -0.445457402 | 1.250162282 | Pass |
| COUNT-THE-1's TEST for specific bytes | 0.365247026 | 0.726112433 | Pass |
| PARKING LOT TEST | -0.327767566 | 1.405126046 | Pass |
| THE MINIMUM DISTANCE TEST | -0.376181538 | 1.404031858 | Pass |
| THE 3DSPHERES TEST | -0.452852651 | 1.460075291 | Pass |
| SQEEZE test | -0.162542352 | 1.340220752 | Pass |
| OVERLAPPING SUMS test | -0.300312996 | 1.192523236 | Pass |
| runs down | -0.335455436 | 1.388049276 | Pass |
| runs up | -0.437017668 | 1.414065588 | Pass |
| Craps Wins | -0.105606359 | 1.498981639 | Pass |
| Craps Throws/Game | 0.020667064 | 1.304900456 | Pass |

**Jumping 2D-3D Neumann Neighbors Results for $\mu \pm 2\sigma$**

| **DIEHARD Test Applied** | Avg - (2*Stdev) | Avg + (2*Stdev) | Test falling in the Range |
|---|---|---|---|
| BIRTHDAY SPACINGS TEST | -0.025216724 | 1.010583124 | Pass |
| THE OVERLAPPING 5-PERMUTATION TEST | 0.075710861 | 1.323667739 | Fail |
| BINARY RANK TEST for 31x31 matrices | 0.090080467 | 0.975763453 | Pass |
| BINARY RANK TEST for 32x32 matrices | 0.163427295 | 0.932494705 | Fail |
| BINARY RANK TEST for 6x8 matrices | 0.058990626 | 1.035759214 | Pass |
| THE BITSTREAM TEST | 0.360808383 | 0.562033937 | Fail |
| OPSO(Overlapping-Pairs-Sparse-Occupancy) | 0.418700474 | 0.681321787 | Fail |
| OQSO(Overlapping-Quadruples-Sparse-Occupancy) | 0.439325502 | 0.604473355 | Pass |
| DNA | 0.424172692 | 0.632454146 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (1st Set of Bytes) | -0.236549421 | 1.028649661 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (2nd Set of Bytes) | -0.162854121 | 0.967559001 | Pass |
| COUNT-THE-1's TEST for specific bytes | 0.425391261 | 0.665968199 | Pass |
| PARKING LOT TEST | -0.038951964 | 1.116310444 | Pass |
| THE MINIMUM DISTANCE TEST | -0.079479305 | 1.107329625 | Pass |
| THE 3DSPHERES TEST | -0.134031327 | 1.141253967 | Pass |
| SQEEZE test | 0.087918165 | 1.089760235 | Pass |
| OVERLAPPING SUMS test | -0.051506958 | 0.943717198 | Pass |
| runs down | -0.04820465 | 1.10079849 | Pass |
| runs up | -0.128503792 | 1.105551712 | Pass |
| Craps Wins | 0.161824974 | 1.231550306 | Pass |
| Craps Throws/Game | 0.234705963 | 1.090861557 | Pass |

3**D Neumann Neighbors Results for** $\mu \pm 3\sigma$

| **DIEHARD Test Applied** | Avg - (3*Stdev) | Avg + (3*Stdev) | Test falling in the Range |
|---|---|---|---|
| BIRTHDAY SPACINGS TEST | -0.435741614 | 1.425704574 | Pass |
| THE OVERLAPPING 5-PERMUTATION TEST | -0.472007462 | 0.992979062 | Pass |
| BINARY RANK TEST for 31x31 matrices | -0.044220687 | 1.139097487 | Pass |
| BINARY RANK TEST for 32x32 matrices | -0.046068254 | 1.139632974 | Pass |
| BINARY RANK TEST for 6x8 matrices | -0.126140145 | 1.297216305 | Pass |
| THE BITSTREAM TEST | 0.353253297 | 0.661098503 | Pass |
| OPSO(Overlapping-Pairs-Sparse-Occupancy) | 0.255205702 | 0.686328167 | Pass |
| OQSO(Overlapping-Quadruples-Sparse-Occupancy) | 0.427632583 | 0.64652256 | Pass |
| DNA | 0.363183714 | 0.627617834 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (1st Set of Bytes) | -0.312142744 | 1.031130424 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (2nd Set of Bytes) | -0.26370657 | 0.60783529 | Fail |
| COUNT-THE-1's TEST for specific bytes | 0.389302573 | 0.759557811 | Pass |
| PARKING LOT TEST | -0.429513575 | 1.310961175 | Pass |
| THE MINIMUM DISTANCE TEST | -0.35842469 | 1.27966301 | Pass |
| THE 3DSPHERES TEST | -0.376849976 | 1.344027976 | Pass |
| SQEEZE test | 0.119969075 | 1.060244365 | Pass |
| OVERLAPPING SUMS test | -0.42305232 | 1.27819208 | Pass |
| runs down | -0.448259736 | 1.314112296 | Pass |
| runs up | -0.312251235 | 1.417887235 | Pass |
| Craps Wins | -0.273826748 | 1.313456508 | Pass |
| Craps Throws/Game | -0.066707395 | 1.400352675 | Pass |

3**D Neumann Neighbors Results for** $\mu \pm 2\sigma$

| **DIEHARD Test Applied** | Avg - (2*Stdev) | Avg + (2*Stdev) | Test falling in the Range |
|---|---|---|---|
| BIRTHDAY SPACINGS TEST | -0.125500582 | 1.115463542 | Pass |
| THE OVERLAPPING 5-PERMUTATION TEST | -0.227843042 | 0.748814642 | Fail |
| BINARY RANK TEST for 31x31 matrices | 0.152999009 | 0.941877791 | Fail |
| BINARY RANK TEST for 32x32 matrices | 0.151548618 | 0.942016102 | Pass |
| BINARY RANK TEST for 6x8 matrices | 0.11108593 | 1.05999023 | Pass |
| THE BITSTREAM TEST | 0.404560831 | 0.609790969 | Pass |
| OPSO(Overlapping-Pairs-Sparse-Occupancy) | 0.327059447 | 0.614474423 | Fail |
| OQSO(Overlapping-Quadruples-Sparse-Occupancy) | 0.464114246 | 0.610040897 | Pass |
| DNA | 0.407256067 | 0.583545481 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (1st Set of Bytes) | -0.088263882 | 0.807251562 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (2nd Set of Bytes) | -0.118449593 | 0.462578313 | Fail |
| COUNT-THE-1's TEST for specific bytes | 0.45101178 | 0.697848604 | Pass |
| PARKING LOT TEST | -0.13943445 | 1.02088205 | Pass |
| THE MINIMUM DISTANCE TEST | -0.085410074 | 1.006648394 | Pass |
| THE 3DSPHERES TEST | -0.090036984 | 1.057214984 | Pass |
| SQEEZE test | 0.276681623 | 0.903531817 | Fail |
| OVERLAPPING SUMS test | -0.139511587 | 0.994651347 | Pass |
| runs down | -0.154531064 | 1.020383624 | Pass |
| runs up | -0.023894824 | 1.129530824 | Pass |
| Craps Wins | -0.009279539 | 1.048909299 | Pass |
| Craps Throws/Game | 0.177802617 | 1.155842663 | Pass |

3**D Neumann + Moore Neighbors Results for** $\mu \pm 3\sigma$

| **DIEHARD Test Applied** | Avg - (3*Stdev) | Avg + (3*Stdev) | Test falling in the Range |
|---|---|---|---|
| BIRTHDAY SPACINGS TEST | -0.362109399 | 1.266893879 | Pass |
| THE OVERLAPPING 5-PERMUTATION TEST | -0.56245372 | 1.41907132 | Pass |
| BINARY RANK TEST for 31x31 matrices | 0.134424213 | 1.109243787 | Pass |
| BINARY RANK TEST for 32x32 matrices | -0.080318823 | 1.221617143 | Pass |
| BINARY RANK TEST for 6x8 matrices | -0.429557295 | 1.502997695 | Pass |
| THE BITSTREAM TEST | 0.255794304 | 0.625934056 | Pass |
| OPSO(Overlapping-Pairs-Sparse-Occupancy) | 0.266061272 | 0.694059771 | Pass |
| OQSO(Overlapping-Quadruples-Sparse-Occupancy) | 0.373966009 | 0.58667942 | Pass |
| DNA | 0.377457885 | 0.614176179 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (1st Set of Bytes) | -0.239838667 | 1.356826667 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (2nd Set of Bytes) | -0.207017927 | 1.156700007 | Pass |
| COUNT-THE-1's TEST for specific bytes | 0.3960481 | 0.69391325 | Pass |
| PARKING LOT TEST | -0.403298639 | 1.430999359 | Pass |
| THE MINIMUM DISTANCE TEST | -0.275662406 | 1.421238886 | Pass |
| THE 3DSPHERES TEST | -0.399816278 | 1.040410598 | Pass |
| SQEEZE test | -0.007541349 | 1.324701509 | Pass |
| OVERLAPPING SUMS test | -0.45990008 | 1.39001168 | Pass |
| runs down | -0.291639118 | 1.367565118 | Pass |
| runs up | -0.423079908 | 1.312661828 | Pass |
| Craps Wins | -0.342775048 | 0.946304648 | Pass |
| Craps Throws/Game | -0.310330192 | 1.212486672 | Pass |

3**D Neumann + Moore Neighbors Results for** $\mu \pm 2\sigma$

| **DIEHARD Test Applied** | Avg - (2*Stdev) | Avg + (2*Stdev) | Test falling in the Range |
|---|---|---|---|
| BIRTHDAY SPACINGS TEST | -0.090608852 | 0.995393332 | Pass |
| THE OVERLAPPING 5-PERMUTATION TEST | -0.232199547 | 1.088817147 | Pass |
| BINARY RANK TEST for 31x31 matrices | 0.296894142 | 0.946773858 | Pass |
| BINARY RANK TEST for 32x32 matrices | 0.136670504 | 1.004627816 | Pass |
| BINARY RANK TEST for 6x8 matrices | -0.107464797 | 1.180905197 | Pass |
| THE BITSTREAM TEST | 0.317484263 | 0.564244097 | Pass |
| OPSO(Overlapping-Pairs-Sparse-Occupancy) | 0.337394356 | 0.622726688 | Pass |
| OQSO(Overlapping-Quadruples-Sparse-Occupancy) | 0.409418244 | 0.551227185 | Pass |
| DNA | 0.416910934 | 0.57472313 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (1st Set of Bytes) | 0.026272222 | 1.090715778 | Pass |
| COUNT-THE-1's TEST on a stream of bytes (2nd Set of Bytes) | 0.020268395 | 0.929413685 | Pass |
| COUNT-THE-1's TEST for specific bytes | 0.445692292 | 0.644269058 | Pass |
| PARKING LOT TEST | -0.097582306 | 1.125283026 | Pass |
| THE MINIMUM DISTANCE TEST | 0.007154476 | 1.138422004 | Pass |
| THE 3DSPHERES TEST | -0.159778465 | 0.800372785 | Fail |
| SQEEZE test | 0.214499128 | 1.102661032 | Pass |
| OVERLAPPING SUMS test | -0.151581453 | 1.081693053 | Pass |
| runs down | -0.015105078 | 1.091031078 | Pass |
| runs up | -0.133789618 | 1.023371538 | Pass |
| Craps Wins | -0.127928432 | 0.731458032 | Pass |
| Craps Throws/Game | -0.056527381 | 0.958683861 | Pass |

# Summary of results for μ±3σ

| | No of Passes Avg + - 3 Std. Dev 99% Values Fall | | | | | |
|---|---|---|---|---|---|---|
| | **Jumping 2D-3D** | **Rand3DVonNeuman** | **3D Moore + Neumann** | **2D Moore** | **2D Von Neumann Moore Jumping** | **2D Von Neumann** |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Fail | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| **Failures** | 0 | 1 | 0 | 0 | 0 | 0 |

# Summary of results for μ±2σ

| | No of Passes Avg + - 2 Std. Dev 95% Values Fall | | | | | |
|---|---|---|---|---|---|---|
| | **Jumping 2D-3D** | **Rand3DVonNeuman** | **3D Moore + Neumann** | **2D Moore** | **2D Von Neumann Moore Jumping** | **2D Von Neumann** |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Fail | Fail | Pass | Pass | Pass | Pass |
| | Pass | Fail | Pass | Pass | Pass | Pass |
| | Fail | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Fail | Pass | Pass | Pass | Pass | Pass |
| | Fail | Fail | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Fail | Pass | Pass | Pass | Fail |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Fail | Pass | Pass | Pass |
| | Pass | Fail | Pass | Pass | Pass | Fail |
| | Pass | Pass | Pass | Fail | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Pass | Pass | Pass |
| | Pass | Pass | Pass | Fail | Pass | Pass |
| **Failures** | 4 | 5 | 1 | 2 | 0 | 2 |

# Appendix C

Results for Safe range value (values >=0.25 AND values <=0.75)
DIEHARD Tests Name used in abbreviated form.

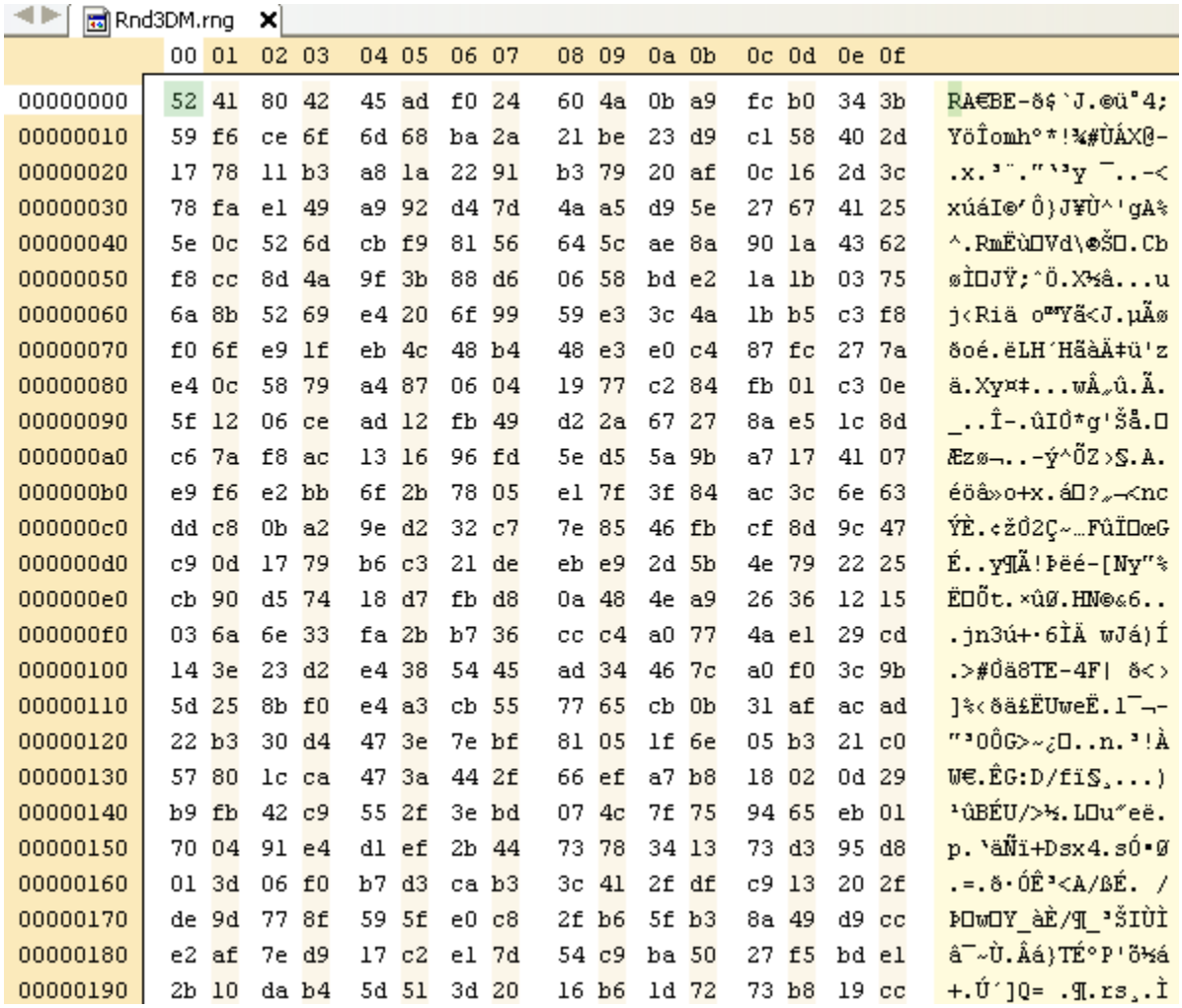
**All 100 values Comparison (values >=0.25 AND values <=0.75) Safe Range**

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| | **Jumping 2D-3D** | **Rand 3D Von Neumann** | **3D Moore + Neumann** | **2D Moore Neighbours** | **2D Von Neumann Moore Jump N** | **2D Von Neumann Neighbors** |
| BST | 68 | 40 | 56 | 52 | 44 | 56 |
| O5PT | 28 | 24 | 36 | 36 | 36 | 28 |
| BR31 | 80 | 84 | 84 | 72 | 68 | 0 |
| BR32 | 84 | 76 | 72 | 76 | 80 | 0 |
| BR6X8 | 60 | 60 | 44 | 56 | 52 | 36 |
| BS | 100 | 100 | 100 | 100 | 100 | 8 |
| OPSO | 100 | 100 | 100 | 76 | 80 | 100 |
| OQSO | 100 | 100 | 100 | 76 | 80 | 100 |
| DNA | 100 | 100 | 100 | 76 | 80 | 100 |
| CT1TSB-1 | 40 | 52 | 48 | 40 | 80 | 28 |
| CT1TSB-2 | 48 | 20 | 72 | 60 | 68 | 40 |
| CT1TSB | 100 | 100 | 100 | 100 | 100 | 100 |
| PLT | 48 | 40 | 40 | 52 | 60 | 64 |
| MDT | 48 | 64 | 52 | 44 | 28 | 32 |
| 3DST | 36 | 48 | 48 | 72 | 52 | 48 |
| ST | 44 | 84 | 48 | 40 | 44 | 0 |
| OST | 56 | 56 | 52 | 48 | 52 | 60 |
| RD | 64 | 48 | 56 | 56 | 56 | 60 |
| RU | 40 | 48 | 52 | 44 | 40 | 52 |
| CW | 36 | 64 | 56 | 24 | 52 | 64 |
| CTG | 60 | 48 | 44 | 64 | 52 | 56 |

| | **Jumping 2D-3D** | **Rand 3D Von Neumann** | **3D Moore + Neumann** | **2D Moore Neighbours** | **2D Von Neumann Moore Jump N** | **2D Von Neumann Neighbors** |
|---|---|---|---|---|---|---|
| **Max Occurance** | 9 | 10 | 9 | 5 | 4 | 8 |

# Appendix D

Sample of Binary file (Snap shot taken for 3D Moore + Neumann Method)



Rnd3DM.rng

| | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0a | 0b | 0c | 0d | 0e | 0f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00000000 | 52 | 41 | 80 | 42 | 45 | ad | f0 | 24 | 60 | 4a | 0b | a9 | fc | b0 | 34 | 3b |
| 00000010 | 59 | f6 | ce | 6f | 6d | 68 | ba | 2a | 21 | be | 23 | d9 | c1 | 58 | 40 | 2d |
| 00000020 | 17 | 78 | 11 | b3 | a8 | 1a | 22 | 91 | b3 | 79 | 20 | af | 0c | 16 | 2d | 3c |
| 00000030 | 78 | fa | e1 | 49 | a9 | 92 | d4 | 7d | 4a | a5 | d9 | 5e | 27 | 67 | 41 | 25 |
| 00000040 | 5e | 0c | 52 | 6d | cb | f9 | 81 | 56 | 64 | 5c | ae | 8a | 90 | 1a | 43 | 62 |
| 00000050 | f8 | cc | 8d | 4a | 9f | 3b | 88 | d6 | 06 | 58 | bd | e2 | 1a | 1b | 03 | 75 |
| 00000060 | 6a | 8b | 52 | 69 | e4 | 20 | 6f | 99 | 59 | e3 | 3c | 4a | 1b | b5 | c3 | f8 |
| 00000070 | f0 | 6f | e9 | 1f | eb | 4c | 48 | b4 | 48 | e3 | e0 | c4 | 87 | fc | 27 | 7a |
| 00000080 | e4 | 0c | 58 | 79 | a4 | 87 | 06 | 04 | 19 | 77 | c2 | 84 | fb | 01 | c3 | 0e |
| 00000090 | 5f | 12 | 06 | ce | ad | 12 | fb | 49 | d2 | 2a | 67 | 27 | 8a | e5 | 1c | 8d |
| 000000a0 | c6 | 7a | f8 | ac | 13 | 16 | 96 | fd | 5e | d5 | 5a | 9b | a7 | 17 | 41 | 07 |
| 000000b0 | e9 | f6 | e2 | bb | 6f | 2b | 78 | 05 | e1 | 7f | 3f | 84 | ac | 3c | 6e | 63 |
| 000000c0 | dd | c8 | 0b | a2 | 9e | d2 | 32 | c7 | 7e | 85 | 46 | fb | cf | 8d | 9c | 47 |
| 000000d0 | c9 | 0d | 17 | 79 | b6 | c3 | 21 | de | eb | e9 | 2d | 5b | 4e | 79 | 22 | 25 |
| 000000e0 | cb | 90 | d5 | 74 | 18 | d7 | fb | d8 | 0a | 48 | 4e | a9 | 26 | 36 | 12 | 15 |
| 000000f0 | 03 | 6a | 6e | 33 | fa | 2b | b7 | 36 | cc | c4 | a0 | 77 | 4a | e1 | 29 | cd |
| 00000100 | 14 | 3e | 23 | d2 | e4 | 38 | 54 | 45 | ad | 34 | 46 | 7c | a0 | f0 | 3c | 9b |
| 00000110 | 5d | 25 | 8b | f0 | e4 | a3 | cb | 55 | 77 | 65 | cb | 0b | 31 | af | ac | ad |
| 00000120 | 22 | b3 | 30 | d4 | 47 | 3e | 7e | bf | 81 | 05 | 1f | 6e | 05 | b3 | 21 | c0 |
| 00000130 | 57 | 80 | 1c | ca | 47 | 3a | 44 | 2f | 66 | ef | a7 | b8 | 18 | 02 | 0d | 29 |
| 00000140 | b9 | fb | 42 | c9 | 55 | 2f | 3e | bd | 07 | 4c | 7f | 75 | 94 | 65 | eb | 01 |
| 00000150 | 70 | 04 | 91 | e4 | d1 | ef | 2b | 44 | 73 | 78 | 34 | 13 | 73 | d3 | 95 | d8 |
| 00000160 | 01 | 3d | 06 | f0 | b7 | d3 | ca | b3 | 3c | 41 | 2f | df | c9 | 13 | 20 | 2f |
| 00000170 | de | 9d | 77 | 8f | 59 | 5f | e0 | c8 | 2f | b6 | 5f | b3 | 8a | 49 | d9 | cc |
| 00000180 | e2 | af | 7e | d9 | 17 | c2 | e1 | 7d | 54 | c9 | ba | 50 | 27 | f5 | bd | e1 |
| 00000190 | 2b | 10 | da | b4 | 5d | 51 | 3d | 20 | 16 | b6 | 1d | 72 | 73 | b8 | 19 | cc |

# Appendix E

Sample of DIEHARD Output file (Snap shot taken for 3D Moore + Neumann Method)

```
BIRTHDAY SPACINGS TEST, M= 512 N=2**24 LAMBDA=  2.0000
          Results for file1
                For a sample of size 500:     mean
          file1           using bits  1 to 24   2.068
 duplicate       number       number
 spacings        observed     expected
       0          67.        67.668
       1         118.       135.335
       2         143.       135.335
       3         103.        90.224
       4          40.        45.112
       5          17.        18.045
 6 to INF         12.         8.282
 Chisquare with  6 d.o.f. =     6.78 p-value=  .658276
  ::::::::::::::::::::::::::::::::::::::::::
                For a sample of size 500:     mean
          file1           using bits  2 to 25   1.938
 duplicate       number       number
 spacings        observed     expected
       0          60.        67.668
       1         148.       135.335
       2         145.       135.335
       3          92.        90.224
       4          27.        45.112
       5          22.        18.045
 6 to INF          6.         8.282
 Chisquare with  6 d.o.f. =    11.55 p-value=  .927114
  ::::::::::::::::::::::::::::::::::::::::::
                For a sample of size 500:     mean
          file1           using bits  3 to 26   1.956
 duplicate       number       number
 spacings        observed     expected
       0          71.        67.668
       1         142.       135.335
       2         124.       135.335
       3          99.        90.224
       4          38.        45.112
       5          21.        18.045
 6 to INF          5.         8.282
 Chisquare with  6 d.o.f. =     5.20 p-value=  .481707
  ::::::::::::::::::::::::::::::::::::::::::
                For a sample of size 500:     mean
          file1           using bits  4 to 27   1.960
 duplicate       number       number
 spacings        observed     expected
```

```
      0          58.       67.668
      1         137.      135.335
      2         158.      135.335
      3          90.       90.224
      4          41.       45.112
      5          10.       18.045
 6 to INF         6.        8.282
Chisquare with  6 d.o.f. =     9.79 p-value=  .866126
 ::::::::::::::::::::::::::::::::::::::::::
               For a sample of size 500:     mean
        file1          using bits  5 to 28   2.116
 duplicate       number       number
 spacings        observed     expected
      0          68.       67.668
      1         133.      135.335
      2         123.      135.335
      3          83.       90.224
      4          53.       45.112
      5          27.       18.045
 6 to INF         13.        8.282
Chisquare with  6 d.o.f. =    10.26 p-value=  .885733
 ::::::::::::::::::::::::::::::::::::::::::
               For a sample of size 500:     mean
        file1          using bits  6 to 29   1.976
 duplicate       number       number
 spacings        observed     expected
      0          72.       67.668
      1         133.      135.335
      2         130.      135.335
      3          98.       90.224
      4          42.       45.112
      5          19.       18.045
 6 to INF         6.        8.282
Chisquare with  6 d.o.f. =     2.09 p-value=  .088971
 ::::::::::::::::::::::::::::::::::::::::::
               For a sample of size 500:     mean
        file1          using bits  7 to 30   1.990
 duplicate       number       number
 spacings        observed     expected
      0          59.       67.668
      1         148.      135.335
      2         141.      135.335
      3          77.       90.224
      4          51.       45.112
      5          15.       18.045
 6 to INF         9.        8.282
Chisquare with  6 d.o.f. =     5.82 p-value=  .555793
 ::::::::::::::::::::::::::::::::::::::::::
               For a sample of size 500:     mean
        file1          using bits  8 to 31   2.032
 duplicate       number       number
```

```
 spacings       observed     expected
       0          62.       67.668
       1         137.       135.335
       2         140.       135.335
       3          88.       90.224
       4          41.       45.112
       5          22.       18.045
 6 to INF         10.        8.282
Chisquare with  6 d.o.f. =     2.31 p-value=  .110796
 ::::::::::::::::::::::::::::::::::::::::::
 ::::::::::::::::::::::::::::::::::::::::::
                For a sample of size 500:     mean
         file1           using bits  9 to 32   2.090
 duplicate       number       number
 spacings       observed     expected
       0          47.       67.668
       1         148.       135.335
       2         139.       135.335
       3          85.       90.224
       4          52.       45.112
       5          18.       18.045
 6 to INF         11.        8.282
Chisquare with  6 d.o.f. =     9.84 p-value=  .868594
 ::::::::::::::::::::::::::::::::::::::::::
 ::::::::::::::::::::::::::::::::::::::::::
  The 9 p-values were
       .658276   .927114   .481707   .866126   .885733
       .088971   .555793   .110796   .868594
 A KSTEST for the 9 p-values yields  .691006

          OPERM5 test for file file1
    For a sample of 1,000,000 consecutive 5-tuples,
chisquare for 99 degrees of freedom= 81.134; p-value= .095538
          OPERM5 test for file file1
    For a sample of 1,000,000 consecutive 5-tuples,
chisquare for 99 degrees of freedom= 94.586; p-value= .393169

 Binary rank test for file1
        Rank test for 31x31 binary matrices:
       rows from leftmost 31 bits of each 32-bit integer
     rank   observed  expected (o-e)^2/e  sum
       28       195     211.4  1.274968    1.275
       29      5161    5134.0   .141886    1.417
       30     23196   23103.0   .373989    1.791
       31     11448   11551.5   .927783    2.719
 chisquare= 2.719 for 3 d. of f.; p-value= .612521


  Binary rank test for file1

        Rank test for 32x32 binary matrices:

       rows from leftmost 32 bits of each 32-bit integer
```

| rank | observed | expected | (o-e)^2/e | sum |
|---|---|---|---|---|
| 29 | 195 | 211.4 | 1.274968 | 1.275 |
| 30 | 5107 | 5134.0 | .142102 | 1.417 |
| 31 | 23086 | 23103.0 | .012578 | 1.430 |
| 32 | 11612 | 11551.5 | .316607 | 1.746 |

chisquare= 1.746 for 3 d. of f.; p-value= .466475

Binary Rank Test for file1

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 1 to 8

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 932 | 944.3 | .160 | .160 |
| r =5 | 22075 | 21743.9 | 5.042 | 5.202 |
| r =6 | 76993 | 77311.8 | 1.315 | 6.517 |

p=1-exp(-SUM/2)= .96155

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 2 to 9

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 922 | 944.3 | .527 | .527 |
| r =5 | 21848 | 21743.9 | .498 | 1.025 |
| r =6 | 77230 | 77311.8 | .087 | 1.112 |

p=1-exp(-SUM/2)= .42639

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 3 to 10

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 952 | 944.3 | .063 | .063 |
| r =5 | 21952 | 21743.9 | 1.992 | 2.054 |
| r =6 | 77096 | 77311.8 | .602 | 2.657 |

p=1-exp(-SUM/2)= .73509

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 4 to 11

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 965 | 944.3 | .454 | .454 |
| r =5 | 21562 | 21743.9 | 1.522 | 1.975 |
| r =6 | 77473 | 77311.8 | .336 | 2.312 |

p=1-exp(-SUM/2)= .68518

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 5 to 12

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 969 | 944.3 | .646 | .646 |
| r =5 | 21465 | 21743.9 | 3.577 | 4.223 |
| r =6 | 77566 | 77311.8 | .836 | 5.059 |

p=1-exp(-SUM/2)= .92031

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 6 to 13

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| r<=4 | 945 | 944.3 | .001 | .001 |
| r =5 | 21674 | 21743.9 | .225 | .225 |
| r =6 | 77381 | 77311.8 | .062 | .287 |

p=1-exp(-SUM/2)= .13375

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 7 to 14

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 976 | 944.3 | 1.064 | 1.064 |
| r =5 | 21475 | 21743.9 | 3.325 | 4.389 |
| r =6 | 77549 | 77311.8 | .728 | 5.117 |

p=1-exp(-SUM/2)= .92259

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 8 to 15

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 924 | 944.3 | .436 | .436 |
| r =5 | 21936 | 21743.9 | 1.697 | 2.134 |
| r =6 | 77140 | 77311.8 | .382 | 2.515 |

p=1-exp(-SUM/2)= .71569

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 9 to 16

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 961 | 944.3 | .295 | .295 |
| r =5 | 22042 | 21743.9 | 4.087 | 4.382 |

| | | | | |
|---|---|---|---|---|
| r =6 | 76997 | 77311.8 | 1.282 | 5.664 |

p=1-exp(-SUM/2)= .94110

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 10 to 17

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 954 | 944.3 | .100 | .100 |
| r =5 | 21496 | 21743.9 | 2.826 | 2.926 |
| r =6 | 77550 | 77311.8 | .734 | 3.660 |

p=1-exp(-SUM/2)= .83957

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 11 to 18

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 905 | 944.3 | 1.636 | 1.636 |
| r =5 | 21575 | 21743.9 | 1.312 | 2.948 |
| r =6 | 77520 | 77311.8 | .561 | 3.508 |

p=1-exp(-SUM/2)= .82695

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 12 to 19

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 954 | 944.3 | .100 | .100 |
| r =5 | 21935 | 21743.9 | 1.680 | 1.779 |
| r =6 | 77111 | 77311.8 | .522 | 2.301 |

p=1-exp(-SUM/2)= .68347

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 13 to 20

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 934 | 944.3 | .112 | .112 |
| r =5 | 21884 | 21743.9 | .903 | 1.015 |
| r =6 | 77182 | 77311.8 | .218 | 1.233 |

p=1-exp(-SUM/2)= .46017

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 14 to 21

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 962 | 944.3 | .332 | .332 |
| r =5 | 21859 | 21743.9 | .609 | .941 |
| r =6 | 77179 | 77311.8 | .228 | 1.169 |

p=1-exp(-SUM/2)= .44265

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 15 to 22

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 948 | 944.3 | .014 | .014 |
| r =5 | 21705 | 21743.9 | .070 | .084 |
| r =6 | 77347 | 77311.8 | .016 | .100 |

p=1-exp(-SUM/2)= .04882

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 16 to 23

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 901 | 944.3 | 1.986 | 1.986 |
| r =5 | 21797 | 21743.9 | .130 | 2.115 |
| r =6 | 77302 | 77311.8 | .001 | 2.117 |

p=1-exp(-SUM/2)= .65294

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 17 to 24

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 876 | 944.3 | 4.940 | 4.940 |
| r =5 | 21771 | 21743.9 | .034 | 4.974 |
| r =6 | 77353 | 77311.8 | .022 | 4.996 |

p=1-exp(-SUM/2)= .91775

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 18 to 25

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 919 | 944.3 | .678 | .678 |
| r =5 | 21678 | 21743.9 | .200 | .878 |
| r =6 | 77403 | 77311.8 | .108 | .985 |

p=1-exp(-SUM/2)= .38897

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 19 to 26

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| r<=4 | 959 | 944.3 | .229 | .229 |
| r =5 | 21604 | 21743.9 | .900 | 1.129 |
| r =6 | 77437 | 77311.8 | .203 | 1.332 |

p=1-exp(-SUM/2)= .48615

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 20 to 27

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 904 | 944.3 | 1.720 | 1.720 |
| r =5 | 21675 | 21743.9 | .218 | 1.938 |
| r =6 | 77421 | 77311.8 | .154 | 2.093 |

p=1-exp(-SUM/2)= .64876

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 21 to 28

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 895 | 944.3 | 2.574 | 2.574 |
| r =5 | 21801 | 21743.9 | .150 | 2.724 |
| r =6 | 77304 | 77311.8 | .001 | 2.725 |

p=1-exp(-SUM/2)= .74394

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 22 to 29

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 932 | 944.3 | .160 | .160 |
| r =5 | 21729 | 21743.9 | .010 | .170 |

| | | | | |
|---|---|---|---|---|
| r =6 | 77339 | 77311.8 | .010 | .180 |

p=1-exp(-SUM/2)= .08608

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 23 to 30

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 967 | 944.3 | .546 | .546 |
| r =5 | 21710 | 21743.9 | .053 | .598 |
| r =6 | 77323 | 77311.8 | .002 | .600 |

p=1-exp(-SUM/2)= .25922

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 24 to 31

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 920 | 944.3 | .625 | .625 |
| r =5 | 21709 | 21743.9 | .056 | .681 |
| r =6 | 77371 | 77311.8 | .045 | .727 |

p=1-exp(-SUM/2)= .30467

Rank of a 6x8 binary matrix,

rows formed from eight bits of the RNG file1

b-rank test for bits 25 to 32

| | OBSERVED | EXPECTED | (O-E)^2/E | SUM |
|---|---|---|---|---|
| r<=4 | 946 | 944.3 | .003 | .003 |
| r =5 | 21697 | 21743.9 | .101 | .104 |
| r =6 | 77357 | 77311.8 | .026 | .131 |

p=1-exp(-SUM/2)= .06323

TEST SUMMARY, 25 tests on 100,000 random 6x8 matrices

These should be 25 uniform [0,1] random variables:

.961546 .426393 .735095 .685179 .920306

.133747 .922587 .715689 .941104 .839569

.826948 .683471 .460169 .442649 .048820

.652939 .917749 .388968 .486151 .648756

.743944 .086080 .259218 .304666 .063231

brank test summary for file1

The KS test for those 25 supposed UNI's yields

KS p-value= .670466

THE OVERLAPPING 20-tuples BITSTREAM TEST, 20 BITS PER WORD, N words

This test uses N=2^21 and samples the bitstream 20 times.

No. missing words should average 141909. with sigma=428.

---

tst no 1: 141373 missing words, -1.25 sigmas from mean, p-value= .10508

tst no 2: 141963 missing words, .13 sigmas from mean, p-value= .54990

tst no 3: 142320 missing words, .96 sigmas from mean, p-value= .83135

tst no 4: 142154 missing words, .57 sigmas from mean, p-value= .71622

tst no 5: 142078 missing words, .39 sigmas from mean, p-value= .65324

tst no 6: 141919 missing words, .02 sigmas from mean, p-value= .50901

tst no 7: 142436 missing words, 1.23 sigmas from mean, p-value= .89075

tst no 8: 141492 missing words, -.98 sigmas from mean, p-value= .16476

tst no 9: 141610 missing words, -.70 sigmas from mean, p-value= .24216

tst no 10: 142457 missing words, 1.28 sigmas from mean, p-value= .89966

tst no 11: 141985 missing words, .18 sigmas from mean, p-value= .57017

tst no 12: 141711 missing words, -.46 sigmas from mean, p-value= .32154

tst no 13:  141792 missing words,  -.27 sigmas from mean, p-value= .39199

tst no 14:  141981 missing words,  .17 sigmas from mean, p-value= .56650

tst no 15:  141545 missing words,  -.85 sigmas from mean, p-value= .19732

tst no 16:  142313 missing words,  .94 sigmas from mean, p-value= .82720

tst no 17:  142317 missing words,  .95 sigmas from mean, p-value= .82958

tst no 18:  142655 missing words,  1.74 sigmas from mean, p-value= .95927

tst no 19:  141925 missing words,  .04 sigmas from mean, p-value= .51460

tst no 20:  141489 missing words,  -.98 sigmas from mean, p-value= .16303

OPSO test for generator file1

Output: No. missing words (mw), equiv normal variate (z), p-value (p)

| | | | | |
|---|---|---|---|---|
| OPSO for file1 | using bits 23 to 32 | 142263 | 1.220 | .8887 |
| OPSO for file1 | using bits 22 to 31 | 141778 | -.453 | .3253 |
| OPSO for file1 | using bits 21 to 30 | 141893 | -.056 | .4775 |
| OPSO for file1 | using bits 20 to 29 | 141881 | -.098 | .4611 |
| OPSO for file1 | using bits 19 to 28 | 141745 | -.567 | .2855 |
| OPSO for file1 | using bits 18 to 27 | 141796 | -.391 | .3480 |
| OPSO for file1 | using bits 17 to 26 | 142021 | .385 | .6499 |
| OPSO for file1 | using bits 16 to 25 | 141711 | -.684 | .2470 |
| OPSO for file1 | using bits 15 to 24 | 141651 | -.891 | .1865 |
| OPSO for file1 | using bits 14 to 23 | 141793 | -.401 | .3442 |
| OPSO for file1 | using bits 13 to 22 | 141866 | -.149 | .4406 |
| OPSO for file1 | using bits 12 to 21 | 141986 | .264 | .6043 |
| OPSO for file1 | using bits 11 to 20 | 141616 | -1.011 | .1559 |
| OPSO for file1 | using bits 10 to 19 | 141921 | .040 | .5161 |

| | | | | |
|---|---|---|---|---|
| OPSO for file1 | using bits 9 to 18 | 141857 | -.180 | .4284 |
| OPSO for file1 | using bits 8 to 17 | 142073 | .564 | .7138 |
| OPSO for file1 | using bits 7 to 16 | 141483 | -1.470 | .0708 |
| OPSO for file1 | using bits 6 to 15 | 141471 | -1.511 | .0653 |
| OPSO for file1 | using bits 5 to 14 | 141986 | .264 | .6043 |
| OPSO for file1 | using bits 4 to 13 | 141602 | -1.060 | .1446 |
| OPSO for file1 | using bits 3 to 12 | 141685 | -.774 | .2196 |
| OPSO for file1 | using bits 2 to 11 | 141473 | -1.505 | .0662 |
| OPSO for file1 | using bits 1 to 10 | 141567 | -1.180 | .1189 |

OQSO test for generator file1

Output: No. missing words (mw), equiv normal variate (z), p-value (p)

| | | | | |
|---|---|---|---|---|
| OQSO for file1 | using bits 28 to 32 | 141863 | -.157 | .4376 |
| OQSO for file1 | using bits 27 to 31 | 142315 | 1.375 | .9155 |
| OQSO for file1 | using bits 26 to 30 | 141982 | .246 | .5973 |
| OQSO for file1 | using bits 25 to 29 | 142036 | .429 | .6662 |
| OQSO for file1 | using bits 24 to 28 | 142131 | .751 | .7738 |
| OQSO for file1 | using bits 23 to 27 | 142027 | .399 | .6550 |
| OQSO for file1 | using bits 22 to 26 | 142093 | .623 | .7332 |
| OQSO for file1 | using bits 21 to 25 | 142086 | .599 | .7254 |
| OQSO for file1 | using bits 20 to 24 | 141587 | -1.093 | .1373 |
| OQSO for file1 | using bits 19 to 23 | 142192 | .958 | .8310 |
| OQSO for file1 | using bits 18 to 22 | 141632 | -.940 | .1736 |
| OQSO for file1 | using bits 17 to 21 | 141389 | -1.764 | .0389 |
| OQSO for file1 | using bits 16 to 20 | 142029 | .406 | .6575 |
| OQSO for file1 | using bits 15 to 19 | 141930 | .070 | .5279 |
| OQSO for file1 | using bits 14 to 18 | 141778 | -.445 | .3281 |

| | | | | |
|---|---|---|---|---|
| OQSO for file1 | using bits 13 to 17 | 142085 | .595 | .7242 |
| OQSO for file1 | using bits 12 to 16 | 141945 | .121 | .5481 |
| OQSO for file1 | using bits 11 to 15 | 142048 | .470 | .6808 |
| OQSO for file1 | using bits 10 to 14 | 141764 | -.493 | .3111 |
| OQSO for file1 | using bits 9 to 13 | 142145 | .799 | .7878 |
| OQSO for file1 | using bits 8 to 12 | 141706 | -.689 | .2453 |
| OQSO for file1 | using bits 7 to 11 | 142063 | .521 | .6988 |
| OQSO for file1 | using bits 6 to 10 | 141460 | -1.523 | .0639 |
| OQSO for file1 | using bits 5 to 9 | 142149 | .812 | .7917 |
| OQSO for file1 | using bits 4 to 8 | 141911 | .006 | .5023 |
| OQSO for file1 | using bits 3 to 7 | 142017 | .365 | .6424 |
| OQSO for file1 | using bits 2 to 6 | 141732 | -.601 | .2739 |
| OQSO for file1 | using bits 1 to 5 | 141645 | -.896 | .1851 |

DNA test for generator file1

Output: No. missing words (mw), equiv normal variate (z), p-value (p)

| | | | | |
|---|---|---|---|---|
| DNA for file1 | using bits 20 to 21 | 141539 | -1.092 | .1373 |
| DNA for file1 | using bits 19 to 20 | 142153 | .719 | .7639 |
| DNA for file1 | using bits 18 to 19 | 142427 | 1.527 | .9366 |
| DNA for file1 | using bits 17 to 18 | 141747 | -.479 | .3160 |
| DNA for file1 | using bits 16 to 17 | 142454 | 1.607 | .9459 |
| DNA for file1 | using bits 15 to 16 | 142369 | 1.356 | .9124 |
| DNA for file1 | using bits 14 to 15 | 142260 | 1.034 | .8495 |
| DNA for file1 | using bits 13 to 14 | 141607 | -.892 | .1862 |
| DNA for file1 | using bits 12 to 13 | 141859 | -.148 | .4410 |
| DNA for file1 | using bits 11 to 12 | 141649 | -.768 | .2213 |
| DNA for file1 | using bits 10 to 11 | 141656 | -.747 | .2274 |

| | | | | |
|---|---|---|---|---|
| DNA for file1 | using bits 9 to 10 | 141961 | .152 | .5606 |
| DNA for file1 | using bits 8 to 9 | 141525 | -1.134 | .1285 |
| DNA for file1 | using bits 7 to 8 | 142007 | .288 | .6134 |
| DNA for file1 | using bits 6 to 7 | 142050 | .415 | .6609 |
| DNA for file1 | using bits 5 to 6 | 142105 | .577 | .7181 |
| DNA for file1 | using bits 4 to 5 | 141334 | -1.697 | .0448 |
| DNA for file1 | using bits 3 to 4 | 141754 | -.458 | .3234 |
| DNA for file1 | using bits 2 to 3 | 141784 | -.370 | .3558 |
| DNA for file1 | using bits 1 to 2 | 141747 | -.479 | .3160 |

Test results for file1

Chi-square with 5^5-5^4=2500 d.of f. for sample size:2560000

| | chisquare | equiv normal | p-value |
|---|---|---|---|

Results fo COUNT-THE-1's in successive bytes:

| | | | |
|---|---|---|---|
| byte stream for file1 | 2405.69 | -1.334 | .091132 |
| byte stream for file1 | 2502.18 | .031 | .512312 |

Results for COUNT-THE-1's in specified bytes:

| | | | |
|---|---|---|---|
| bits 1 to 8 | 2485.26 | -.208 | .417458 |
| bits 2 to 9 | 2533.00 | .467 | .679619 |
| bits 3 to 10 | 2459.37 | -.575 | .282770 |
| bits 4 to 11 | 2523.00 | .325 | .627492 |
| bits 5 to 12 | 2423.42 | -1.083 | .139397 |
| bits 6 to 13 | 2490.86 | -.129 | .448556 |
| bits 7 to 14 | 2518.10 | .256 | .600999 |
| bits 8 to 15 | 2656.09 | 2.207 | .986360 |

bits  9 to 16  2614.31      1.617      .947014

bits 10 to 17  2429.86      -.992      .160633

CDPARK: result of ten tests on file file1

Of 12,000 tries, the average no. of successes

should be 3523 with sigma=21.9

Successes: 3522    z-score:  -.046 p-value: .481790

Successes: 3537    z-score:   .639 p-value: .738676

Successes: 3526    z-score:   .137 p-value: .554479

Successes: 3534    z-score:   .502 p-value: .692266

Successes: 3505    z-score:  -.822 p-value: .205562

Successes: 3546    z-score:  1.050 p-value: .853193

Successes: 3536    z-score:   .594 p-value: .723613

Successes: 3513    z-score:  -.457 p-value: .323972

Successes: 3562    z-score:  1.781 p-value: .962529

Successes: 3526    z-score:   .137 p-value: .554479

square size   avg. no.  parked   sample sigma

100.           3530.700       15.411

KSTEST for the above 10: p=  .638628

This is the MINIMUM DISTANCE test

for random integers in the file file1

Sample no.    d^2     avg     equiv uni

5    1.3638   1.0519     .746056

10     .6086    .7828     .457574

| | | | |
|---|---|---|---|
| 15 | .3159 | .6406 | .272010 |
| 20 | .1821 | .9221 | .167223 |
| 25 | .0348 | .8639 | .034414 |
| 30 | .0262 | .8330 | .026021 |
| 35 | .8224 | 1.0543 | .562435 |
| 40 | .9775 | 1.1022 | .625586 |
| 45 | .0502 | 1.0318 | .049235 |
| 50 | .9500 | .9946 | .615120 |
| 55 | .2766 | .9520 | .242659 |
| 60 | .0701 | .8943 | .068018 |
| 65 | .2568 | .8797 | .227451 |
| 70 | 3.7634 | .8857 | .977230 |
| 75 | 7.1344 | .9775 | .999231 |
| 80 | 1.3870 | .9895 | .751923 |
| 85 | .5250 | 1.0089 | .409974 |
| 90 | 2.8711 | 1.0000 | .944175 |
| 95 | .0495 | 1.0115 | .048536 |
| 100 | 2.0191 | 1.0343 | .868568 |

MINIMUM DISTANCE TEST for file1

Result of KS test on 20 transformed mindist^2's:

p-value= .533812

The 3DSPHERES test for file file1

sample no:  1  r^3=  38.193  p-value= .72004

sample no:  2  r^3=  65.735  p-value= .88821

sample no:  3  r^3=  24.364  p-value= .55609

sample no:  4  r^3=  12.159  p-value= .33323

sample no:  5  r^3=  10.890  p-value= .30441

sample no:  6  r^3=  23.250  p-value= .53929

sample no:  7  r^3=  77.702  p-value= .92499

sample no:  8  r^3=   2.731  p-value= .08700

sample no:  9  r^3=  25.410  p-value= .57131

sample no: 10  r^3=  20.434  p-value= .49396

sample no: 11  r^3=   9.450  p-value= .27021

sample no: 12  r^3=   6.442  p-value= .19323

sample no: 13  r^3=  11.870  p-value= .32676

sample no: 14  r^3=   1.689  p-value= .05474

sample no: 15  r^3=  92.819  p-value= .95468

sample no: 16  r^3=   6.956  p-value= .20694

sample no: 17  r^3=  11.314  p-value= .31418

sample no: 18  r^3=  31.576  p-value= .65095

sample no: 19  r^3=   2.126  p-value= .06841

sample no: 20  r^3=  30.239  p-value= .63504

A KS test is applied to those 20 p-values.

---

3DSPHERES test for file file1  p-value= .241794

RESULTS OF SQUEEZE TEST FOR file1

Table of standardized frequency counts

( (obs-exp)/sqrt(exp) )^2

for j taking values <=6,7,8,...,47,>=48:

2.7 .5 .8 -2.4 -.5 -.1

.7 -1.1 .1 -.4 -1.9 -1.6

1.2 .6 -.8 -.7 1.0 -.2

.8 -1.6 -.8 .2 1.5 1.5

-.2 .9 .0 -1.7 2.9 1.3

.7 .4 -1.1 .9 -.5 1.3

.3 .5 .1 -.1 .9 .0

2.7

Chi-square with 42 degrees of freedom: 62.622

z-score= 2.250 p-value= .978858

Test no. 1 p-value .430451

Test no. 2 p-value .217270

Test no. 3 p-value .622948

Test no. 4 p-value .057847

Test no. 5 p-value .793093

Test no. 6 p-value .142356

Test no. 7 p-value .004947

Test no. 8 p-value .792843

Test no. 9 p-value .983215

Test no. 10 p-value .974643

Results of the OSUM test for file1

KSTEST on the above 10 p-values: .638171

Run test for file1 :

runs up; ks test for 10 p's: .949285

runs down; ks test for 10 p's: .135459

Run test for file1 :

runs up; ks test for 10 p's: .109780

runs down; ks test for 10 p's: .911528

Results of craps test for file1

No. of wins: Observed Expected

98561 98585.86

98561= No. of wins, z-score= -.111 pvalue= .45573

Analysis of Throws-per-Game:

Chisq= 15.24 for 20 degrees of freedom, p= .23739

| Throws | Observed | Expected | Chisq | Sum |
|---|---|---|---|---|
| 1 | 66795 | 66666.7 | .247 | .247 |
| 2 | 37579 | 37654.3 | .151 | .398 |
| 3 | 27003 | 26954.7 | .086 | .484 |
| 4 | 19205 | 19313.5 | .609 | 1.093 |
| 5 | 13753 | 13851.4 | .699 | 1.793 |
| 6 | 10135 | 9943.5 | 3.686 | 5.479 |
| 7 | 7128 | 7145.0 | .041 | 5.520 |
| 8 | 5035 | 5139.1 | 2.108 | 7.627 |
| 9 | 3695 | 3699.9 | .006 | 7.633 |
| 10 | 2733 | 2666.3 | 1.669 | 9.302 |
| 11 | 1867 | 1923.3 | 1.650 | 10.952 |

| | | | | |
|---|---|---|---|---|
| 12 | 1370 | 1388.7 | .253 | 11.205 |
| 13 | 1025 | 1003.7 | .451 | 11.656 |
| 14 | 726 | 726.1 | .000 | 11.656 |
| 15 | 516 | 525.8 | .184 | 11.840 |
| 16 | 404 | 381.2 | 1.370 | 13.210 |
| 17 | 297 | 276.5 | 1.514 | 14.724 |
| 18 | 206 | 200.8 | .133 | 14.857 |
| 19 | 140 | 146.0 | .245 | 15.102 |
| 20 | 107 | 106.2 | .006 | 15.108 |
| 21 | 281 | 287.1 | .130 | 15.238 |

SUMMARY FOR file1

p-value for no. of wins: .455735

p-value for throws/game: .237386