Study on Non-Uniform Number Conserving Cellular Automata Rules Over a Space of Higher Cardinality

[1]Sudhakar Sahoo, [2]Suryakanta Pal, [3]Padmalochan Panda and [4]Somanath Dutta

[1]Institute of Mathematics and Applications, Andharua, Bhubaneswar -751029

Email: sudhakar.sahoo@gmail.com

[2]Model Degree College, Malkangiri-764048

Email: surya_tuna@yahoo.co.in

[3, 4] Silicon Institute of Technology, Silicon Hills, Patia, Bhubaneswar-751024

Email: pl5141panda@gmail.com and sdutta5264@gmail.com

Abstract: The purpose of the present study is to implement different one dimensional elementary Cellular Automata (CA) problems defined over a higher space of cardinality using python programming. The exact size of the rule space is $(256^{l})^{n}$ where l is the length of the CA configuration and n is the number of iterations in the space-time diagram. The experiments are enumerated and the outcomes obtained after simulation are reported. Important observations are highlighted for the purpose of characterizing the CA rules and the mathematical justification has been given for every experimental results.

Keyword: Cellular automata rules, Monotonic increasing and decreasing cellular automata rules, Number conserving cellular automata (NCCA) rules, State transition diagrams, Python Programming.

1. Introduction

The concept of cellular automata (CA), proposed by John von Neumann [18] has attracted many researchers due to its simple structure and several applications in various field of science like parallel computing, physics, image processing, biology, pattern recognition, VLSI design, etc. Cellular automaton is a discrete dynamical system as it operates in a discrete time steps on a discrete space (the lattice) with a discrete set of states [30, 31].

The CA is described completely with the help of following 4 parameters [7, 20, 32].

1. Dimension
2. Number of States
3. Size of Neighborhood
4. Local transition rule

Therefore, in most of the previously dealt article, the concept of CA is defined as a quadruple (d, Q, N, f) where $d \in N$ is the dimension of cellular space. Q={0,1,2,………..,q-1} is a finite set of all possible q number of states.

N= $\{(\overrightarrow{v_1}, \overrightarrow{v_2}, ..., \overrightarrow{v_n}) \mid \overrightarrow{v_i} \in \mathbb{Z}^d\}$ is the set of all d-dimensional neighborhood vector of size n.

$f: Q^{r_l + r_r + 1} \rightarrow Q$ is a local update rule which is a mapping from set of all possible neighborhoods to set of states where the positive integers r_l and r_r are called the left and right radii of the rule.

For any $\vec{v} \in \mathbb{Z}^d$, the neighborhood consists of $\vec{v} + \overrightarrow{v_1}, \vec{v} + \overrightarrow{v_2}, ..., \vec{v} + \overrightarrow{v_n}$ where n= $r_r + r_l + 1$. CA rule is displayed as a lookup table (table-1) listing each possible neighborhood with a state. According to Wolfram's naming convention, we can assign a rule number N(f) to each function f where $N(f) = \sum_{(v_1, v_2, ..., v_n)} f(v_1, v_2, ..., v_n) q^{q^{n-1} v_1 + q^{n-2} v_2 + ... + q^0 v_n}$. By changing the conditions on d, Q, N and f, we can produce different types of CA. There is a large variety of CAs such as uniform, non-uniform (hybrid), null boundary, periodic boundary, linear, non-linear, reversible, irreversible, programmable CA etc. which are used as mathematical models to design different complex systems. In literature [8, 10, 23, 24], CA have been characterized using reachability tree, matrix algebra, de-Brujin graph, space-time diagram, state transition diagram (STD) etc. One of the important characterizations is classification of CA rules based on certain properties [7]. Wolfram classified the dynamic behavior of two states one dimensional CA into four classes, namely homogeneous, periodic, chaotic and complex localized structure [30]. Another fundamental characterization of CA is to search the appropriate CA rule for the solution of Density Classification Task (DCT), where number conserving property of CA rules (NCCA) is an important criteria [16, 19, 21, 26, 27, 28].

In this article, we have used only one dimensional two states non-uniform CA i.e. d=1 and Q={0,1}. In a standard CA, size of the neighborhood is taken as 2r+1 where r is called radius of the CA. For r=1, there are 2^8=256 CA rules which are called elementary cellular automata in which neighbourhood of a cell consists of the cell itself and some neighbours on the either side of the cell. The next state of a cell is determined by the formula $c_i(t+1) = \mathrm{f}(c_{i-1}(t), c_i(t), c_{i+1}(t))$ where $c_i(t)$ denotes the state of the i-th cell at time t.

Table-1: Shows CA rule 184

Block	111	110	101	100	011	010	001	000
Rule 184	1	0	1	1	1	0	0	0

It shifts a "1" to its right if its right cell is in state 0 and hence is called as right shift rule i.e. it converts all blocks "10" to "01". CA rule 184 can be represented as $c_i(t+1) = \begin{cases} c_{i-1}(t) & \text{if } \mathrm{c_i}(t) = 0 \\ c_{i+1}(t) & \text{if } \mathrm{c_i}(t) = 1 \end{cases}$

In this article an attempt has been made to study the CA rules where in the space-time diagram, at any position any elementary CA rule out of 256 rules can be applied. Thus, the rule space cardinality becomes $(256^l)^n$ where l is the length of the CA configuration and n is the number of iterations in the space-time diagram.

In section 2, definitions regarding the monotonic properties of CA rules have been given. Section 3 deals with the CA experiments carried out using python programming language. In this particular section the CA problems are chosen with a specific objective to search the CA rules which are number conserving in nature. Experimental outcomes obtained in section 3 are analyzed and the results are given in section 4. Section 5 concludes the article specifying little future research scopes.

2. Definitions

Monotonically increasing rules:

A one-dimensional, m-neighborhood CA with local update rule $f: \mathrm{Q}^m \to \mathrm{Q}$ is monotonically increasing under periodic boundary condition if for all binary strings $x_1x_2....x_n$ of length $n \geq m$, $f(x_1, x_2, ..., x_{m-1}, x_m) + f(x_2, x_3, ..., x_m, x_{m+1}) + ... + f(x_n, x_1, ..., x_{m-2}, x_{m-1}) \geq x_1 + x_2 + ... + x_n$.

For elementary CA, the above condition reduces to

$f(x_1, x_2, x_3) + f(x_2, x_3, x_1) + ... + f(x_n, x_1, x_2) \geq x_1 + x_2 + ... + x_n$.........................(1)

Monotonically decreasing rules:

A one-dimensional, m-neighborhood CA with local update rule $f: \mathrm{Q}^m \to \mathrm{Q}$ is monotonically increasing under periodic boundary condition if for all binary strings $x_1x_2....x_n$ of length $n \geq m$, $f(x_1, x_2, ..., x_{m-1}, x_m) + f(x_2, x_3, ..., x_m, x_{m+1}) + ... + f(x_n, x_1, ..., x_{m-2}, x_{m-1}) \leq x_1 + x_2 + ... + x_n$.

For elementary CA, the above condition reduces to

$f(x_1, x_2, x_3) + f(x_2, x_3, x_1) + ... + f(x_n, x_1, x_2) \leq x_1 + x_2 + ... + x_n$.........................(2)

If a CA rule is both monotonically number increasing and number decreasing, then it is called as number conserving CA [1, 2, 3]. i.e.

$f(x_1, x_2, ..., x_{m-1}, x_m) + f(x_2, x_3, ..., x_m, x_{m+1}) + ... + f(x_n, x_1, ..., x_{m-2}, x_{m-1}) = x_1 + x_2 + ... + x_n$.

For elementary CA, the above condition reduces to

$f(x_1, x_2, x_3) + f(x_2, x_3, x_1) + ... + f(x_n, x_1, x_2) = x_1 + x_2 + ... + x_n$.........................(3)

Strictly increasing rules:
The definition follows when we replace inequality $\geq$ in (1) by $>$.

Strictly decreasing rules:
The definition follows when we replace inequality $\leq$ in (2) by $<$.

Number Conserving Cellular Automata:
Cellular Automata Rules which are both monotonically increasing and decreasing are called number conserving CA rules.

3. Simulation of Cellular Automata Problems using Python

Our programming efforts on Cellular Automata study are enumerated as follows.

Experiment 1: Simple 1D Cellular automaton was developed for application of a particular non-uniform elementary rule on a string.

```
#Defines a single transition and returns the changed state value
print(transition('001',191))

def transition(st,r):
    r=bin(r)[2:]
    r='00000000'[:-len(r)]+r
    r=r[::-1]
    st=int('0b'+st,2)
    return r[st]

#Returns a binary string by applying <rule> to string <bi>
print(apply('111',[192,23,44]))
def apply(bi,rule):
    bi=bi[-1]+bi+bi[0]
    temp=[]
    for i in range(1,len(bi)-1):
        temp.append(transition(bi[i-1:i+2],rule[i-1]))
    return(''.join(temp))
```

Sample Output:

```
>>> print(apply('11011',[21,55,162,45,33]))
   00110
```

Experiment 2: (**To generate State Transition Diagram (STD) of an arbitrary non-uniform CA rule**). Given a rule vector and a size n (4<= n <=8), cellular automaton was designed to apply the non-uniform rule vector to all the 2^n possible binary strings of length n.

```
def apply_all(n,rule):
     print('For Rule : ',rule)
     for i in range(0,2**n):
          s=bin(i)[2:]
          string='0'*n
          string=string[:-len(s)]+s
          print('String :',string,'\t->\t',apply(string,rule))
```

Sample Output:

```
>>> apply_all(4,[96,45,78,156])

     For Rule :  [96, 45, 78, 156]
     String : 0000   ->       0100
     String : 0001   ->       0111
     String : 0010   ->       0011
     String : 0011   ->       0010
     String : 0100   ->       0100
     String : 0101   ->       1101
     String : 0110   ->       0111
     String : 0111   ->       1100
     String : 1000   ->       0000
     String : 1001   ->       1011
     String : 1010   ->       0110
     String : 1011   ->       1111
     String : 1100   ->       0000
     String : 1101   ->       0001
     String : 1110   ->       0010
     String : 1111   ->       0001
```

Experiment 3: (**STD for elementary non-uniform CA rules varying over time**). Given a size n (4 <= n <= 8) and an integer k, considering all the possible binary strings of size n, random non-uniform rule vectors were applied k times to generate a sequence of evolution from all possible binary strings of length n where different non-uniform CA acts at different time steps. This can be a generic model for those applications where a sequence of non-uniform rule vectors are acting differently in different time steps. Study of these STDs is clearly a complicated extension in the field of standard CA as the maximum size of these rule space is $(256^l)^n$ where l is the length of the CA configuration and n is the number of iterations in the space-time diagram.

```
def print_pattern(n,k):
    print('\t\t\t\tSequence of Rule vectors')
    for i in range(0,2**n):
        s=bin(i)[2:]
        string='0'*n
        string=string[:-len(s)]+s
        for i in range(k):
            rule=np.array(create_rule(len(string)))
```

```
                print('k=',i,'\t',string,"\t",rule)
                string=apply(string,rule)
            print('k=',k,'\t',string,'\n')
```

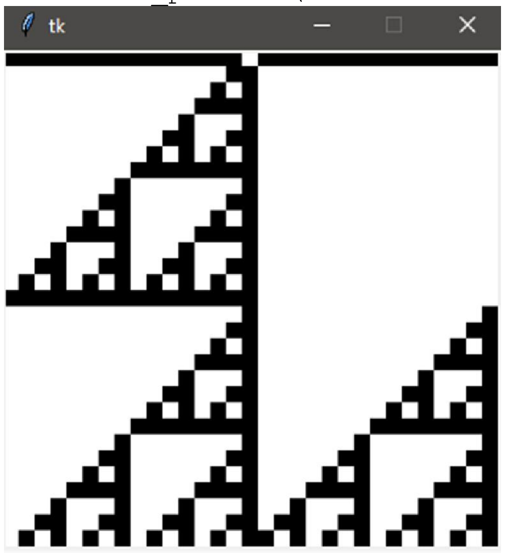
```
Sample Output:
>>>print_pattern(4,2)
                      Sequence of Rule vectors
     k= 0     0000    [ 33  59 153 252]
     k= 1     1110    [ 16 228 219  15]
     k= 2     0110

     k= 0     0001    [ 49 231   5 221]
     k= 1     1101    [139  19 237 248]
     k= 2     1011
     .
     .
     .
     k= 0     1001    [119 163  37 219]
     k= 1     1001    [ 36 236  43 115]
     k= 2     0010

     k= 0     1010    [106  38 250  89]
     k= 1     0100    [210 119   6 250]
     k= 2     1100
     .
     .
     .
     k= 0     1110    [ 18 127  40  97]
     k= 1     0001    [232 171  32 106]
     k= 2     0100

     k= 0     1111    [125 191 125 178]
     k= 1     0101    [176 129 200 146]
     k= 2     1000
```

Experiment 4: To generate space time diagram starting from a standard initial configuration <0…0 1 0…0> that contains a single 1 in the binary string.

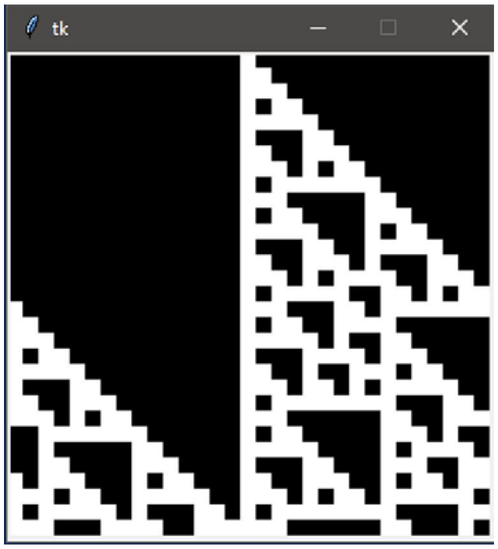
```
    #function to create binary image from the matrix of rules
generated from string

    def create(arr):
        arr=x=[list(map(int,i)) for i in arr]
        s=len(arr[0])
        n=len(arr)
        master = Tk()
        master.resizable(0,0)
        w1 = Canvas(master, width=10*s, height=10*n)
        w1.pack()
        w=h=10
        x,y=0,0
        for i in (arr):
            for j in (i):
                if j==0:
                    w1.create_rectangle(x, y, x+w, y+h,
          fill="black",outline="")
                else:
                    w1.create_rectangle(x, y, x+w, y+h,
          fill="white",outline="")
                x=x+w
            y=y+h
            x=0
        mainloop()

    #function to trigger the create function
    def particular_pattern(string,rule):
        st=[]
        for i in range(len(string)):
            st.append(string)
            string=apply(string,rule)
        create(st)
```

Sample Output:

```
>>> particular_pattern('0'*15+'1'+'0'*15,[153]*31)
```



```
>>> particular_pattern('0'*15+'1'+'0'*15,[124]*31)
```



Experiment 5 (Drawing STD of a NCCA rule vector): To solve the problem of density classification task (DCT), one of the CA rule vector will be < 170, 240, 238, 192, 204 > which is a number conserving CA and has been used to solve DCT [4- 6, 11-14, 25, 29]. To draw the STD of CA rule vector < 170, 240, 238, 192, 204 > where weight of input string=weight of output string, following algorithm has been used.
Algorithm 1:

Step 1. Input rule vector: < 170, 240, 238, 192, 204 >

Step 2. Apply this particular rule vector for all possible 5-bit strings and print the output

0=00000 goes to………X_1
1=00001 goes to….. …X_2
2=00010 goes to………X_3
.
.
.
31=11111 goes to………X_{31}
where each X_i is the decimal equivalent of the 5-bit binary string, after the rule vector is applied on *i*.

Step 3. Output array: X_1, X_2, X_3,…,X_{31}

Step 4. From this output array the STD of the rule vector < 170, 240, 238, 192, 204 > can be drawn.

```
def print_pattern(n,rule):
    output=[]
    for i in range(0,2**n):
        s=bin(i)[2:]
        string='0'*n
        string=string[:-len(s)]+s
        print(i,' = ',string,' -> ',apply(string,rule))
```
Sample Output:

```
>>> n.print_pattern(5,[170, 240, 238, 192, 204])
```

```
0  =  00000  ->  00000
1  =  00001  ->  00001
.
.
.
15  =  01111  ->  10111
16  =  10000  ->  01000
.
.
30  =  11110  ->  11110
31  =  11111  ->  11111
```

Experiment 6 (List of NCCA rules and their STD): Consider only nine CA rules 136, 170, 184, 192, 204, 226, 238, 240, 252. So, possible number of non-uniform CA rule vector of length 5 is 9^5. Each rule vector will act on all possible 32 binary strings giving some output. With the help of computer programming we have tested how many rule vectors is number conserving [5, 6] and accordingly we have prepared a list i.e. the CA rules that satisfies the condition "weight of input string=weight of output string" on application of the CA rule vector on the input. Here weight means number of 1's in the binary string (Example: weight of string 000101=2). Only for these CA rule vectors, state transition diagrams like Experiment 5 have been drawn.

```
#returns all output -> apply rule on all 2^n binary strings
def print_pattern_defined(n,rule):
      output=[]
      for i in range(0,2**n):
         s=bin(i)[2:]
         string='0'*n
         string=string[:-len(s)]+s
         output.append(apply(string,rule))
      return output

#checks whether rule is number conserving or not
def num_conserving(string,d):
    flag=0
    for i in range(0,len(string)):
        x=bin(i)[2:]
        p=sum(i=='1' for i in x)
        if p==0:
            continue
        if p == sum(j=='1' for j in string[i]):
            flag=1
        else:
            flag=0
            break
    return flag

#driver code for the output
x=9
st="136 170 184 192 204 226 238 240 252"
rule=list(map(int, st.split(' ')[:x]))
```

```
for n in range(5,8):
    comb=list(itertools.product(rule,repeat=n))
    count=0
    for i in comb:
        x=print_pattern_defined(n,i)
        if num_conserving(x,n):
            print(i)
            count+=1
    print("Total no of ",n," bit NCCA rules ",count,"\n\n")
```

Sample Output:

```
(136, 184, 184, 184, 252)
(136, 184, 184, 252, 204)
.
.
.
(252, 238, 192, 204, 136)
(252, 238, 226, 192, 136)
Total no of  5  bit NCCA rules  125

(136, 184, 184, 184, 184, 252)
(136, 184, 184, 184, 252, 204)
.
.
.
(252, 238, 226, 192, 204, 136)
(252, 238, 226, 226, 192, 136)
Total no of  6  bit NCCA rules  326

(136, 184, 184, 184, 184, 184, 252)
(136, 184, 184, 184, 184, 252, 204)
.
.
.
(252, 238, 226, 226, 192, 204, 136)
(252, 238, 226, 226, 226, 192, 136)
Total no of  7  bit NCCA rules  845
```

Experiment 7 (Weight increasing and weight decreasing CA uniform rules): If for all strings of size n (i.e. 2^n), the application of a rule R (of dimension n) changes the weight (i.e. number of 1's) of the string..

There are no such rules which strictly increase or decrease the weight of strings for all CA rules of size n (4 <= n <=9). However there are uniform CA rules which conserve the weight of all binary strings of size n. The uniform CA rules with any of <170, 184, 204, 226, 240> will conserve the weight for all n.

```
def print_pattern_defined(n,rule):
    output=[]
    for i in range(0,2**n):
        s=bin(i)[2:]
        string='0'*n
        string=string[:-len(s)]+s
        output.append(apply(string,rule))
    return output
def num_conserving(n,string):
    arr1=[]
    arr2=[]
    arr3=[]
    for i in range(0,len(string)):
        x=bin(i)[2:]
        p=sum(i=='1' for i in x)
        q=sum(j=='1' for j in string[i])
        if p==q:
            arr3.append(i)
        if p<=q:
            arr1.append(i)
        if p>=q:
           arr2.append(i)
    return(arr1,arr2,arr3)
q1=[]
q2=[]
q3=[]
n=int(input())
for i in range(256):
    x=[i for j in range(n)]
    arr1,arr2,arr3=num_conserving(n,print_pattern_defined(n,x))
    perc=(len(arr1)/(2**n))*100
    if perc==100.0:
        q1.append(i)
    perc=(len(arr2)/(2**n))*100
    if perc==100.0:
        q2.append(i)
    perc=(len(arr3)/(2**n))*100
    if perc==100.0:
        q3.append(i)
print('Less than equal to : Total :',len(q1),'\n',q1)
print('Greater than equal to : Total :',len(q2),'\n',q2)
print('Equals to : Total :',len(q3),'\n',q3)
```

Sample Output: For size 5 :

```
Less than equal to : Total : 46
 [170, 171, 174, 175, 184, 185, 186, 187, 188, 189, 190, 191,
204, 205, 206, 207, 220, 221, 222, 223, 226, 227, 230, 231, 234,
235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247,
248, 249, 250, 251, 252, 253, 254, 255]

Greater than equal to : Total : 46
 [0, 2, 4, 8, 10, 12, 16, 24, 32, 34, 40, 42, 48, 56, 64, 66, 68,
72, 76, 80, 96, 98, 112, 128, 130, 132, 136, 138, 140, 144, 152,
160, 162, 168, 170, 176, 184, 192, 194, 196, 200, 204, 208, 224,
226, 240]

Equals to : Total : 5
 [170, 184, 204, 226, 240]
```

Experiment 8 (Number Promoting and Number Demoting CA rules): If for all strings of size n (i.e. 2^n), the application of a rule R (of dimension n) changes the number (i.e. decimal value of the binary string).

```
def print_pattern_defined(n,rule):
    output=[]
    for i in range(0,2**n):
        s=bin(i)[2:]
        string='0'*n
        string=string[:-len(s)]+s
        output.append(apply(string,rule))
    return output
def num_conserving(n,string):
     arr1=[]
     arr2=[]
     arr3=[]
     for i in range(0,len(string)):
          p=i
          q=int(string[i],2)
          if p==q:
               arr3.append(i)
          if p<=q:
               arr1.append(i)
          if p>=q:
               arr2.append(i)
     return(arr1,arr2,arr3)
q1=[]
q2=[]
q3=[]
n=int(input())
for i in range(256):
    x=[i for j in range(n)]
    arr1,arr2,arr3=num_conserving(n,print_pattern_defined(n,x))
    perc=(len(arr1)/(2**n))*100
    if perc==100.0:
        q1.append(i)
    perc=(len(arr2)/(2**n))*100
    if perc==100.0:
        q2.append(i)
    perc=(len(arr3)/(2**n))*100
    if perc==100.0:
        q3.append(i)
print('Less than equal to : Total :',len(q1),'\n',q1)
print('Greater than equal to : Total :',len(q2),'\n',q2)
print('Equals to : Total :',len(q3),'\n',q3)
```

Sample Output: For size 5

```
Less than equal to : Total : 16
 [204, 205, 206, 207, 220, 221, 222, 223, 236, 237, 238,
239, 252, 253, 254, 255]
Greater than equal to : Total : 16
```

```
 [0, 4, 8, 12, 64, 68, 72, 76, 128, 132, 136, 140, 192,
196, 200, 204]
Equals to : Total : 1
 [204]
```

4. Theoretical Analysis over Experimental Outcomes

Theorem-1: There are 46 monotonically increasing (or decreasing) elementary two state CA rule for any CA of length n=5.

Proof:- Let us denote $f(x_1x_2x_3)=f_i$, i.e. $f(000)=f_0$, $f(001)=f_1$, $f(010)=f_2$, $f(011)=f_3$, etc.
We will prove it for n=5.
Out of $2^n= 2^5=32$ strings, the distinct strings are 00000, 00001, 00011, 00101, 00111, 01011 and 01111. If we take its all possible cyclic rotation, we get all 32 strings.
Applying equation (1) to the string 00000, $f(000)+ f(000)+ f(000)+ f(000)+ f(000)\geq0$
$\Rightarrow 5f_0 \geq 0 \Rightarrow f_0 \geq 0$……………………………………(1)
For the string 00001, $f(100)+ f(000)+ f(000)+ f(001)+ f(010)\geq1$
$\Rightarrow f_4+2f_0 + f_1+f_2 \geq 1$…………………………………...(2)
For the string 00011, $f(100)+ f(000)+ f(001)+ f(011)+ f(110)\geq2$
$\Rightarrow f_4+f_0 + f_1+f_3+f_6 \geq 2$…………………………………(3)
For the string 00101, $f(100)+ f(001)+ f(010)+ f(101)+ f(010)\geq2$
$\Rightarrow f_4+f_1+2f_2 +f_5\geq 2$…………………………………....(4)
For the string 00111, $f(100)+ f(001)+ f(011)+ f(111)+f(110)\geq3$
$\Rightarrow f_4+ f_1+f_3+f_7+ f_6 \geq 3$…………………………………(5)
For the string 01011, $f(101)+ f(010)+ f(101)+ f(011)+f(110)\geq3$
$\Rightarrow f_2+f_3+2 f_5+ f_6 \geq 3$…………………………………(6)
For the string 01111, $f(101)+ f(011)+ f(111)+ f(111)+f(110)\geq4$
$\Rightarrow f_5+ f_3+f_7+f_7 +f_6 \geq 4$……………………………….....(7)
For the string 11111, $f(111)+ f(111)+ f(111)+ f(111)+f(111)\geq5$
$\Rightarrow f_7+ f_7+f_7++f_7 +f_7 \geq 5$……………………………….(8)
$\Rightarrow 5f_7\geq 5\Rightarrow f_7 \geq 1\Rightarrow f_7 =1$.
From equation (5), we get $f_4+ f_1+f_3+ f_6 \geq 2$.
From equation (7), we get $f_5+ f_3+ f_6 \geq 2$.
So we have to solve the inequalities, $f_0 \geq 0$, $2f_0 + f_1+f_2 +f_4\geq 1$, $f_0 + f_1+f_3+ f_4 +f_6 \geq 2$, $f_1+2f_2 + f_4+f_5\geq 2$, $f_1+f_3+ f_4+ f_6 \geq 2$, $f_2+f_3+2 f_5+f_6 \geq 3$, $f_5+ f_3+ f_6 \geq 2$………………………..(9)
Solving the above linear system of inequalities, we get the following possibility of the different neighborhoods. The corresponding rules are listed in following table.

Table-2: Shows different monotonically increasing CA rules

CA Rules	111(f_7)	110(f_6)	101(f_5)	100(f_4)	011(f_3)	010(f_2)	001(f_1)	000(f_0)
170	1	0	1	0	1	0	1	0
171	1	0	1	0	1	0	1	1
174	1	0	1	0	1	1	1	0
175	1	0	1	0	1	1	1	1
184	1	0	1	1	1	0	0	0
185	1	0	1	1	1	0	0	1
186	1	0	1	1	1	0	1	0

187	1	0	1	1	1	0	1	1
188	1	0	1	1	1	1	0	0
189	1	0	1	1	1	1	0	1
190	1	0	1	1	1	1	1	0
191	1	0	1	1	1	1	1	1
204	1	1	0	0	1	1	0	0
205	1	1	0	0	1	1	0	1
206	1	1	0	0	1	1	1	0
207	1	1	0	0	1	1	1	1
220	1	1	0	1	1	1	0	0
221	1	1	0	1	1	1	0	1
222	1	1	0	1	1	1	1	0
223	1	1	0	1	1	1	1	1
226	1	1	1	0	0	0	1	0
227	1	1	1	0	0	0	1	1
230	1	1	1	0	0	1	1	0
231	1	1	1	0	0	1	1	1
234	1	1	1	0	1	0	1	0
235	1	1	1	0	1	0	1	1
236	1	1	1	0	1	1	0	0
237	1	1	1	0	1	1	0	1
238	1	1	1	0	1	1	1	0
239	1	1	1	0	1	1	1	1
240	1	1	1	1	0	0	0	0
241	1	1	1	1	0	0	0	1
242	1	1	1	1	0	0	1	0
243	1	1	1	1	0	0	1	1
244	1	1	1	1	0	1	0	0
245	1	1	1	1	0	1	0	1
246	1	1	1	1	0	1	1	0
247	1	1	1	1	0	1	1	1
248	1	1	1	1	1	0	0	0
249	1	1	1	1	1	0	0	1
250	1	1	1	1	1	0	1	0
251	1	1	1	1	1	0	1	1
252	1	1	1	1	1	1	0	0
253	1	1	1	1	1	1	0	1
254	1	1	1	1	1	1	1	0
255	1	1	1	1	1	1	1	1

Similarly, it can be proved inductively that there are only 46 increasing rules for a CA of any finite length. Out of these rules, rules 170, 184, 204, 226, 240 are number conserving [9,15, 22].

Reversing the inequalities in (9), we can derive that the monotonically decreasing rules are 0, 2, 4, 8, 10, 12, 16, 24, 32, 34, 40, 42, 48, 56, 64, 66, 68, 72, 76, 80, 96, 98, 112, 128, 130, 132, 136, 138, 140, 144, 152, 160, 162, 168, 170, 176, 184, 192, 194, 196, 200, 204, 208, 224, 226, 240.

Thus all elementary CA rules can be divided into 3 classes as shown in table 3.

Table-3: Shows different class of CA rules based on their monotonic property

Different classes	CA rules

Class-1(Monotonically increasing rules)	170, 171, 174, 175, 184, 185, 186, 187, 188, 189, 190, 191, 204, 205, 206, 207, 220, 221, 222, 223, 226, 227, 230, 231, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255
Class-2(Monotonically decreasing rules)	0, 2, 4, 8, 10, 12, 16, 24, 32, 34, 40, 42, 48, 56, 64, 66, 68, 72, 76, 80, 96, 98, 112, 128, 130, 132, 136, 138, 140, 144, 152, 160, 162, 168, 170, 176, 184, 192, 194, 196, 200, 204, 208, 224, 226, 240
Class-3(Neither monotonically increasing nor decreasing rules)	1, 3, 5, 6, 7, 9, 11, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 25, 26, 27, 28, 29, 30, 31, 33, 35, 36, 37, 38, 39, 41, 43, 44, 45, 46, 47, 49, 50, 51, 52, 53, 54, 55, 57, 58, 59, 60, 61, 62, 63, 65, 67, 69, 70, 71, 73, 74, 75, 77, 78, 79, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 97, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 129, 131, 133, 134, 135, 137, 139, 141, 142, 143, 145, 146, 147, 148, 149, 150, 151, 153, 154, 155, 156, 157, 158, 159, 161, 163, 164, 165, 166, 167, 169, 172, 173, 177, 178, 179, 180, 181, 182, 183, 193, 195, 197, 198, 199, 201, 202, 203, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 225, 228, 229, 232, 233

Properties:-

1. Class-2 rules are conjugate rules of class-1.
2. The remaining 169 rules in class-3 are neither monotonically increasing nor decreasing.
3. If R_i, i=1,2,….,n is selected from class-3, then the non-uniform CA $<R_1, R_2, \ldots, R_n>$ is neither monotonically increasing nor decreasing rule.
4. If R_i, i=1,2,….,n is selected from class-1, then the non-uniform CA $<R_1, R_2, \ldots, R_n>$ is either monotonically decreasing or neither of the two.
5. If R_i, i=1,2,….,n is selected from class-2, then the non-uniform CA $<R_1, R_2, \ldots, R_n>$ is either monotonically decreasing or neither of the two.
6. If R_i is a monotone rule but not number conserving, then it must be an unbalanced rule.
   This property follows from the necessary condition of number conserving rule.
7. If R_i is monotonically increasing rule and even, then R_i+1 is monotonically increasing rule. [It is obvious as for even rule, 000…0. Next rule is odd so 000…1.If last increases weight, this rule again increases the weight or keep unchanged depending on structure of string.]
8. The intersection of class-1 and class-2 is the number conserving cellular automata(NCCA) rules as for these rules, $f(x_1,x_2,x_3)+f(x_2,x_3,x_1)+\ldots+f(x_n,x_1,x_2)=x_1+x_2+\ldots+x_n$.
   Hence the NCCA rules are 170, 184, 204, 226, 240.

Theorem-2: Three CA rules 239, 253 and 255 are strictly increasing rules for all binary strings of any length. Their conjugate rules 8, 64 and 0 are strictly decreasing rules for all binary strings of any length.

Proof:-

Table-4: Shows CA rules, all possible length 3 neighborhoods (nbhd) both in binary and decimal.

Rule/Nbhd	111(7)	110(6)	101(5)	100(4)	011(3)	010(2)	001(1)	000(0)
239	1	1	1	0	1	1	1	1
253	1	1	1	1	1	1	0	1
255	1	1	1	1	1	1	1	1

Since neighborhoods 010, 011, 110, 111 of all rules are assigned to 1 so weight of strings never decreases. Since neighborhood 111 of all rules is assigned to 1 so weight of string 111…11 remains same in all steps.
Now consider all strings except 111…11.
These strings must contains at least one 0. These strings must contain at least one template as 000, 001, 100 or 101 for all $n\geq3$.
Since all neighborhoods of rule 255 are assigned to 1 so weight of string increases by at least one.
Consider the rule 239. If the string contains at least one template as 000, 001 or 101, then weight increases by at least one because all these neighborhoods are assigned to 1.
Suppose a string contains a template 100. If n=3, then 100 has next state as 101. So weight increases by one. If $n\geq3$, the right position of the template (or leftmost position if 100 is towards end of the string) must be either 0 or 1. So the template will be either 1000 or 1001. The image of 00 is 01 so that weight increases by at least 1.
Consider the rule 253. If the string contains at least one template as 000, 100 or 101, then weight increases by at least 1 because all these neighborhoods are assigned to 1. Consider the string containing the template 001.
If n=3, the next state of 001 is 101 where weight increases by 1. If n=4, the string containing 001 is 0010 or 0011. Then their next states are 1011, 1011 where weight increases by 1.If n=5, then the string containing 001 at the beginning is 00100, 001001, 00110, 00111.The next states are 10111, 101101, 10111, 10111 where weight of the string increases by at least one. Inductively, the weight of the string increases by at least one for all values of n.
Similarly, we can apply similar logic to get strictly decreasing rules.

Let, Class-i=$\{x_1x_2...x_n \in \{0,1\}^n \mid w(x_1x_2...x_n)=i\}$ = set of all strings with weight-i.. For n=5, we will find some CA rules which will decrease or conserve the weight of half of the classes and do the reverse for another half of the strings.

Theorem-3: For n=5, the only rules which decreases or conserve the weight of the string in class-i where i=0, 1, 2 and increases or conserve the weight of the string in class-i where i=3, 4, 5 are Rules 170, 184, 204, 226, 232, 240.

Proof:- Applying equation (1) to the string 00000, f(000)+ f(000)+ f(000)+ f(000)+ f(000)=0

$\Rightarrow 5f_0= 0 \Rightarrow f_0= 0$……………………………….(1)
For the string 00001, f(100)+ f(000)+ f(000)+ f(001)+ f(010)$\leq$1
$\Rightarrow f_4+ f_1+f_2 \leq 1$………………………………….(2)
For the string 00011, f(100)+ f(000)+ f(001)+ f(011)+ f(110)$\leq$2
$\Rightarrow f_4+ f_1+f_3+f_6 \leq 2$……………………………….(3)
For the string 00101, f(100)+ f(001)+ f(010)+ f(101)+ f(010) $\leq$2
$\Rightarrow f_4+f_1+2f_2 +f_5\leq 2$……………………………..(4)
For the string 00111, f(100)+ f(001)+ f(011)+ f(111)+f(110)$\geq$3
$\Rightarrow f_4+ f_1+f_3+f_7+ f_6 \geq 3$………………………….(5)
For the string 01011, f(101)+ f(010)+ f(101)+ f(011)+f(110)$\geq$3
$\Rightarrow f_2+f_3+2 f_5+ f_6 \geq 3$………………………….(6)

For the string 01111, f(101)+ f(011)+ f(111)+ f(111)+f(110)≥4
$\Rightarrow f_5+ f_3+f_7+f_7 +f_6 \geq 4$…………………………(7)
For the string 11111, f(111)+ f(111)+ f(111)+ f(111)+f(111)=5
$\Rightarrow f_7+ f_7+f_7++f_7 +f_7=5 \Rightarrow f_7 =1$. ………………(8)
From equation (5), we get $f_4+ f_1+f_3+ f_6 \geq 2$.
From equation (7), we get $f_5+ f_3+ f_6 \geq 2$.
So we have to solve the inequalities, $f_1+f_2 +f_4 \leq 1$, $f_1+f_3+ f_4 +f_6 \leq 2$, $f_1+2f_2 + f_4+f_5 \leq 2$, $f_1+f_3+ f_4+ f_6 \geq 2$, $f_2+f_3+2 f_5+f_6 \geq 3$, $f_5+ f_3+ f_6 \geq 2$…………………(9)
Solving these linear system of inequalities, we get the CA rules as 170, 184, 204, 226, 232, 240.

Observations: Out of 6 rules, 5 rules are number conserving and the only nontrivial rule is 232.

Regarding strictly increasing and strictly decreasing CA rules the experimental outcome shows that there are no such rules which strictly increase or decrease the decimal value of binary strings for all CA rules of size n (4 <= n <=9). However there is only a single uniform CA rule which conserve the decimal value of all binary strings of size n which is the identity rule 204. The generic proof for any length configuration is trivial as the total possible strings are finite and strictly increasing for all strings requires a domain whose cardinality is at least one more according to pigeon hole principle. It is not clear to us that relaxing the strictness for one, two or more strings out of 2^n possible strings whether any CA rule exists in the rule space and this can be another experiment for our future study.

5. Conclusion and future research directions

In this report an effort has been given to implement the task of 1-D elementary CA rules which are non-uniform in both space and time using python. This particular rule space is computationally infeasible to handle due to its exponential complexity in nature. Therefore, the experiments were carried out over a feasible computing environment with a small sample size from which few inferences are observed. Further, the inferences are mathematically proved for its validity in general for arbitrary sample size.

Our next effort is to search rule vectors (perhaps using evolutionary techniques) which are number conserving over the space of all CA rules by utilizing different parameters of CA and this will be helpful in many applications but rarely studied due to its computational infeasibility of the rule space. We will also extend this 1-D work to two-dimension and three-dimension and their application towards density classification task for higher domain as well.

.

References:

[1] Bocaro N., 2007. "Eventually Number-conserving cellular automata", *International Journal of Modern Physics C*, Vol. 18, No. 1, pp. 35- 42.

[2] Boccara, N., Fukś, H., 2002. "Number Conserving Cellular Automaton Rules", *Fundamenta Informaticae*, 52, pp. 1-13.

[3] Boccara, N., Fukś, H., 1998. "Cellular Automaton Rules Conserving the Number of Active Sites", *Journal of Physics A: Mathematical and General*, 31(28), pp. 6007-6018.

[4] Chau, H. F., Siu, L.W., Yan, K. K., 1999. "One dimensional n-ary density classification using two cellular automaton rules", *International Journal of Modern Physics C*, 10(5), pp. 883-889.

[5] Capcarrere, M. S., Sipper, M., Tomassini, M., 1996. “Two-state, r =1 Cellular Automaton that Classifies Density”, *Physical Review Letter,* 77(24), 4969-4971.

[6] Capcarrere, M. S., Sipper, M., 2001. “Necessary conditions for density classification by cellular automata”, *Physical Review E*, Vol. 64, pp. 036113/1–036113/4.

[7] Choudhury, P. P., Sahoo S., Hasssan S., Basu S., Ghosh D., Kar D., Ghosh A., Ghosh A., Ghosh A. K., 2010. “Classification of Cellular Automata Rules Based on Their Properties”, *International Journal of Computational Cognition*, Vol. 8, No.4, pp. 50-54.

[8] Das, A.K., Sanyal, A., Pal Chaudhuri, P., 1991: “On Characterization of Cellular Automata with Matrix Algebra” *Information Science* 61(3), 251 (1991).

[9] Durand, B., Formenti, E., Róka, Z., 2003. “Number conserving cellular automata I: decidability”, *Theoretical Computer Science*, 299, pp. 523-535.

[10] Das, R., Mitchell, M., Crutchfield, J. P., 1994. “A genetic algorithm discovers particle-based computation in cellular automata, Parallel Problem Solving from Nature - PPSN III” *Lecture Notes in Computer Science*, 866, pp. 344-353.

[11] de Oliveira, P. P. B., 2013. “Conceptual connections around density determination in cellular automata”, *Proc. of the 19th International Workshop on Cellular Automata and Discrete Complex Systems: Lecture Notes in Computer Science*, 8155, pp. 1-14.

[12] de Oliveira, P. P. B., 2014. “On Density Determination With Cellular Automata: Results, Constructions and Directions”, *Journal of Cellular Automata*, 9(5-6), pp. 357-385.

[13] de Oliveira, P. P. B., Faria F., Zanon R. and Leite R.M., 2015. “Computational Evidence against the Possibility of Density Determination with Fixed Arrangements of Non-Local Elementary Rules”, *International. Journal of Unconventional Computing*, 11(3-4), pp. 325-343.

[14] Fukś H., 1997. “Solution of the density classification problem with two cellular automata rules”, *Physical Review E*, 55(3), R2081-R2084.

[15] Fukś , H., 2004. “Probabilistic cellular automata with conserved quantities”, *Nonlinearity*, 17, pp. 159–173.

[16] Gacs, P., Kurdyumov, G. L., Levin, L. A., 1978. “One dimensional uniform arrays that wash out finite islands”, *Problemy Peredachi Informatsii*, 14, pp. 92–98.

[17] Goles E., Moreira A., Rapaport I., 2011. “Communication complexity in number-conserving and monotone cellular automata”, *Theoretical Computer Science,* 412, pp. 3616–3628.

[18] J. v. Neumann., 1966. “The Theory of Self-Reproducing Automata”, A. W. Burks (ed), *Univ. of Illinois Press*, Urbana and London.

[19] Land, M., Belew, R. K., 1995. “No perfect two-state CA for density classification exists”, *Physical Review Letters*, 74(25), pp. 5148-5150.

[20] Li W., Packard N. H., Langton G. C., 1990. "Transition Phenomena in cellular automata rule space", *Physics D*, Vol 45, pp. 77-94.

[21] Maiti, N., Munshi, S., Chaudhuri, P. P., 2006. "An Analytical formulation for Cellular Automata (CA) based Solution of Density Classification task (DCT)", *Lecture Notes in Computer Science*, 4173, pp. 147–156.

[22] Moreira A., Boccaro N., Goles E., 2004. " On conservative and monotone one-dimensional cellular automata and their particle representation", *Theoretical Computer Science,* 325, pp.285 – 316.

[23] Naskar, N., Das, S., 2012. "Characterization of non-uniform cellular automata having only two point states - 0^n and 1^n", *Proc. of the 18th International Workshop on Cellular Automata and Discrete Complex Systems*: *Exploratory Papers*, Rapport de Recherche I3S - ISRN: I3S/RR-2012-04-FR, pp. 29–37.

[24] Packard, N. H., 1988. "Adaptation towards the edge of chaos", *Dynamic Patterns in Complex Systems, World Scientific*, Singapore, pp. 293–301.

[25] Pal S., Sahoo S., Nayak B. K., 2016. "Deterministic Computing Technique for Perfect Density Classification", arXiv:1607.06909 [nlin.CG].

[26] Reynaga, R., Amthauer, E., 2003. "Two-dimensional cellular automata of radius one for density classification task ρ = ½", *Pattern Recognition Letters,* 24(15), pp. 2849–2856.

[27] Redeker, M., 2015. **"**Density Classification Quality of the Traffic-Majority Rules", *Journal of Cellular Automata*, 10(3-4), pp. 195-234.

[28] Sipper, M., Capcarrere, M. S., E. Ronald, 1998. "A simple cellular automaton that solves the density and ordering problems", *International journal of modern Physics C*, 9(7), pp. 899-902.

[29] Sahoo, S., Choudhury P. P., Pal, A., Nayak, B. K., 2013. "Solutions on 1D and 2D density classification problem using programmable cellular automata", *Journal of cellular automata*, 9(1), pp 59-88.

[30] S. Wolfam., 2002. "A New Kind of Science", *Wolfram Media, Inc.*

[31] S. Wolfram, 1986."*Theory and Applications of Cellular Automata"*, Singapore: World Scientific.

[32] Wolz, D. and de Oliveira, P. P. B., 2008. "Very effective evolutionary techniques for searching cellular automata rule spaces", *Journal of Cellular automata*, 3(4), pp. 289 -312.