Computational Physics
Fall 2020

Lalit Chaudhary

November 4, 2020

Nagel-Schreckenberg Model of Traffic Flow

Contents

List of Figures

1 Introduction

Cellular automata (CA) are models that are discrete in space, time and state variables. The discreteness makes CA efficient for computer simulations.

To describe the traffic flow on a street using a CA, the street is first divided into cells, typically of length 7.5 m. Each cell can now either be empty or occupied by exactly one car. Each car is characterized by its current velocity v which can take the values v=0,1,2,... ,v_{max}.

The temporal evolution of a given state, as per Nagel und Schreckenberg, is governed in 4 steps that have applied simultaneously to all the cars.

- **Acceleration**: All cars that have not already reached the maximal velocity v_{max} accelerate by one unit, i.e., $v \rightarrow v + 1$
- **Prevent Collision**: If a car has d empty cells in front of it and its velocity v (after step 1) is larger then d, then it reduces the velocity to d, i.e. $v \rightarrow min\,(d, v)$
- **Random Deceleration**: For each car with random probability p_c, decelerate it by 1 if p_c is smaller than dawdle probability p (usually 0.2). i.e, $v \rightarrow v - 1$ if $p_c < p$
- **Movement of Cars**: After above steps the new velocity v_n for each car n has been determined. The n^{th} cars is moved forward by v_n cells, i.e., $x_n \rightarrow x_n + v_n$

2 Setup for Simulation

For the specific case of ring shaped street, the Nagel-Schreckenberg Model was programmed. In all the space-time diagrams used in this report, the velocity value of -1 has been used to denote an empty cell on the street.

```
#Nagel-Schreckenberg model of traffic flow
def NaSc (S, n):

    #To store resulting matrix
    result=np.zeros(shape=(n, N))
    result[0] = S
    t = 0

    while (t < n):                  #iterate n times

        for i in range(len(S)):
            if(S[i] == inf):        #do nothing on empty cells
                continue

            if(S[i] < vmax):        #Accelerate if not at max speed
                S[i] = S[i]+1

            d = countEmpty(S, i)
            S[i] = min(d, S[i])     #Prevent Collision

        for i in range(len(S)):     #Randomization with dawdle probability
            if(S[i] == inf):
                continue
            prob = random()
            if(prob < p and S[i]>0):
                S[i] = S[i] - 1

        Snew = [inf]*len(S)         #Store updated values in new array

        for i in range(len(S)):
            if(S[i] == inf):
                continue
            if (i+S[i] < N):         #make street Circular
                Snew[i+S[i]] = S[i]
            else:
                Snew[i+S[i]-N] = S[i]

        S=Snew.copy()               #pass udated array for next iteration
        result[t] = Snew            #Store updated array in matrix
        t = t+ 1
    return result

#For plotting the results
def Plot(result, title):
    #get discrete colormap
    cmap = plt.get_cmap('Greys', np.max(result)-np.min(result)+1)
    mat = plt.matshow(result,cmap=cmap,vmin = np.min(result)-.5,
                                       vmax = np.max(result)+.5)
    cax = plt.colorbar(mat, ticks=np.arange(np.min(result),np.max(result)+1))

    plt.xlabel('Position on Road')
    plt.ylabel('Simulation Time')
    plt.title(title, loc='center')
    #plt.savefig(title)
```

Figure 1: Code Snippet for implemetation of Nagel-Schreckenberg Model

2.1 Validation of Code

For the validation of code, the program was run with 50 cars having randomized initial velocities. With the value of p set to 1, the result obtained after 100 iterations of the Nagel-Schreckenberg model is shown in figure (2).

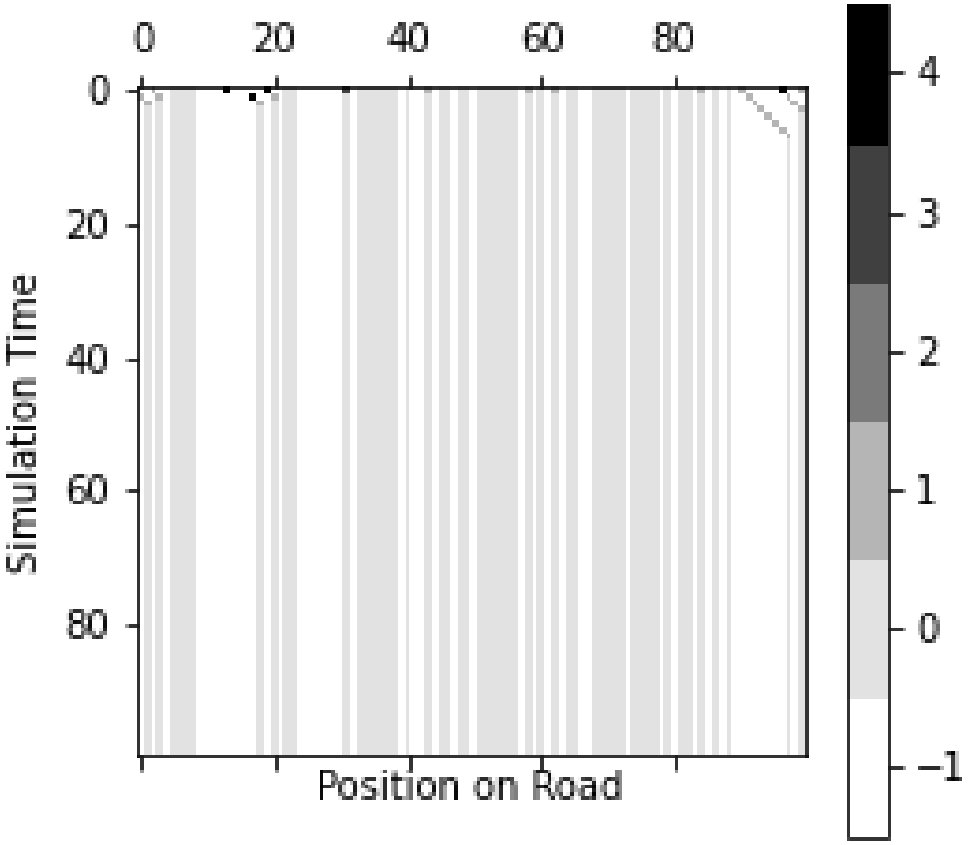

Figure 2: Time evolution of 50 cars with p=1

As expected, all the cars stop after initial few iterations, since the third step ensures the velocity of each car decreases in every iteration until they reach 0.

3 Case Specific Simulations

3.1 Traffic Jam with 6 cars and p = 0

With 18 total cars on the street and 6 in a jam initially, and $p = 0$, the result of 100 iterations is shown in figure (3).

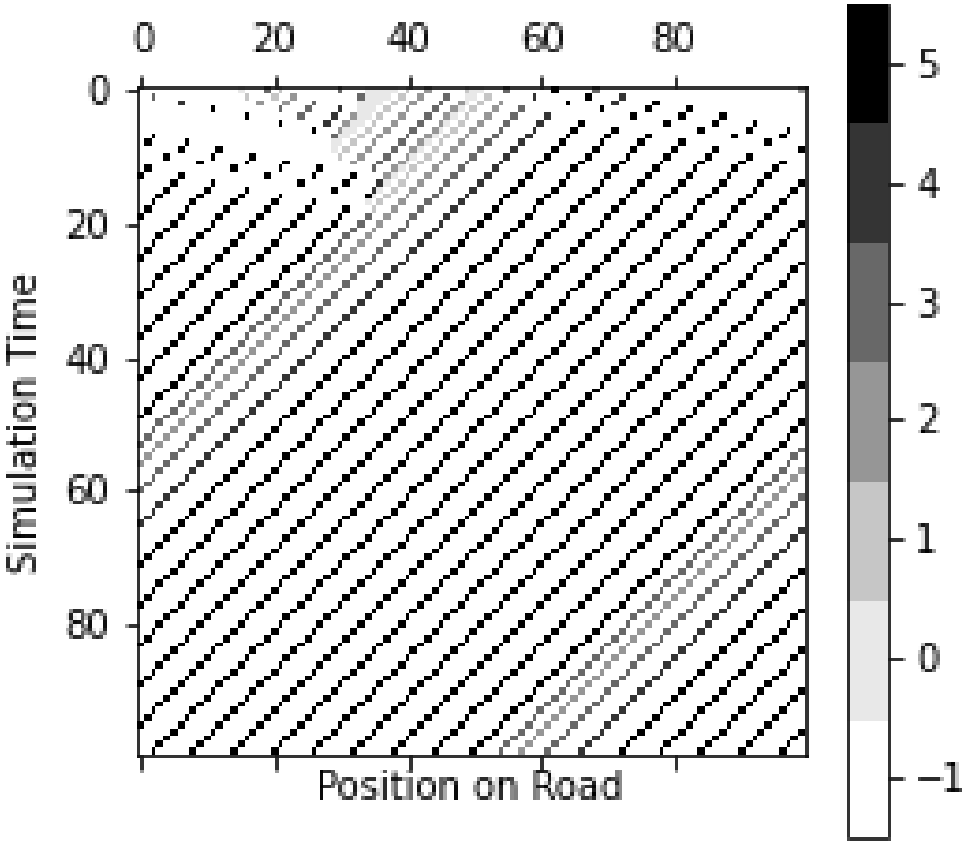

Figure 3: Time evolution of 18 cars with p = 0

3.2 Traffic Jam with 6 cars and p = 0.2

With 18 total cars on the street and 6 in a jam initially, and $p = 0.2$, the result of 100 iterations is shown in figure (4).

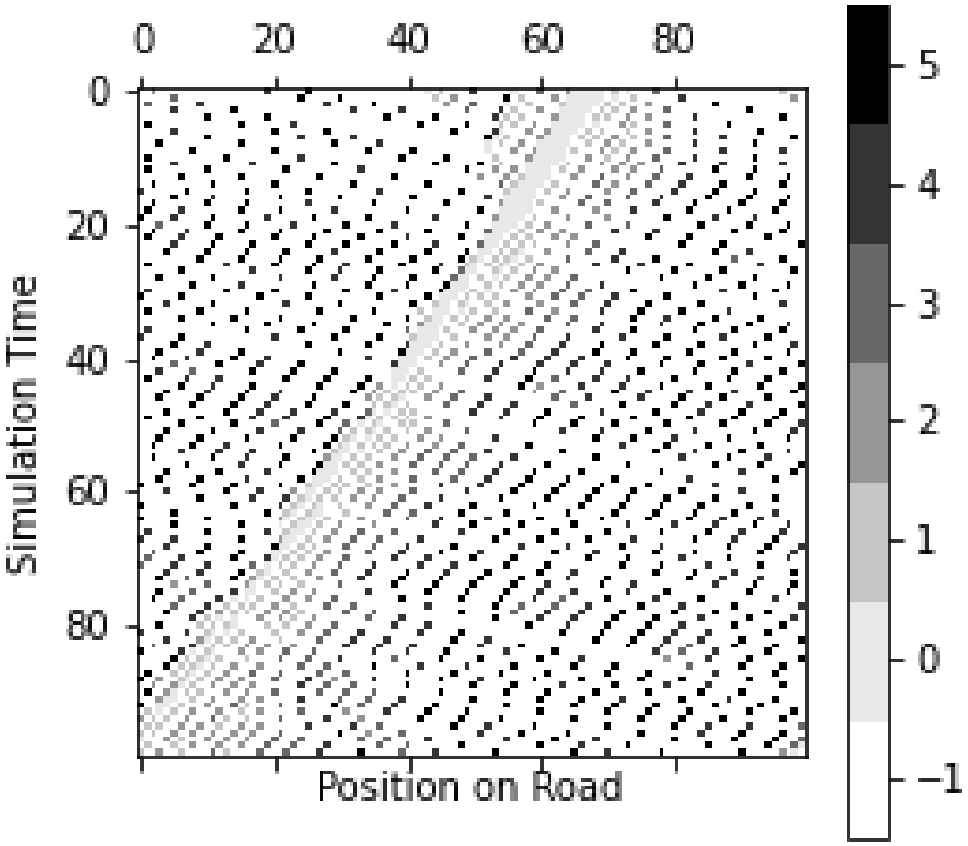

Figure 4: Time evolution of 18 cars with p = 0.2

3.3 Traffic Jam with 12 cars and p = 0

With 36 total cars on the street and 12 in a jam initially, and $p = 0$, the result of 100 iterations is shown in figure (5).

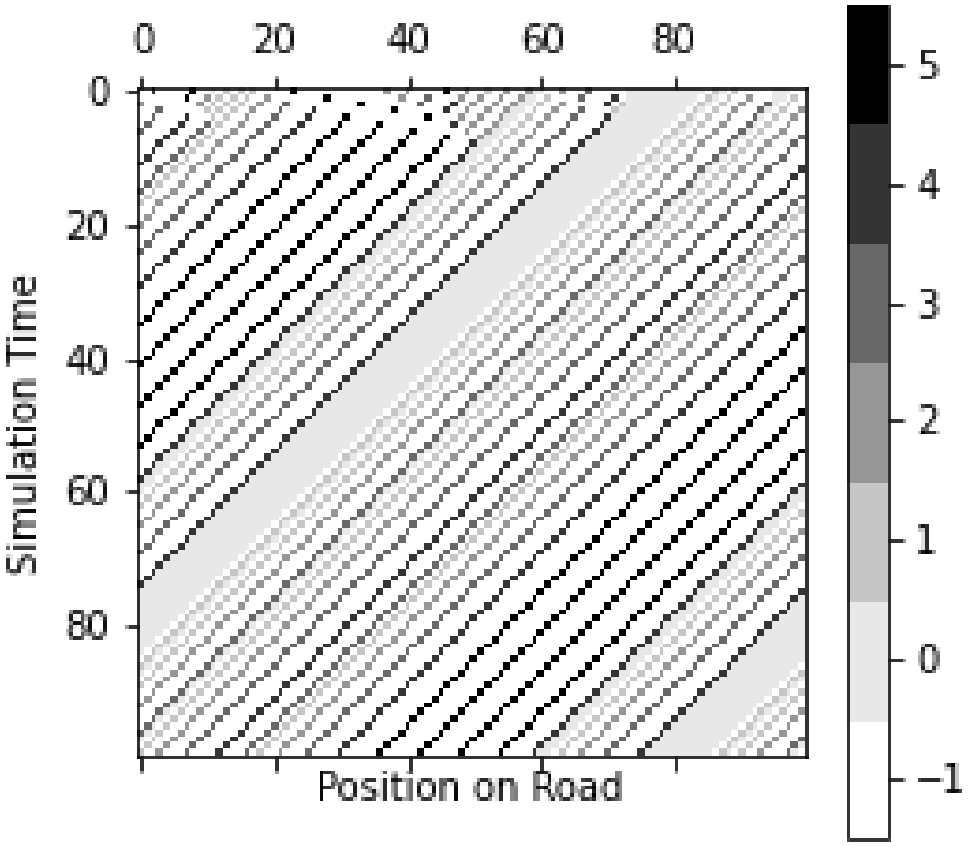

Figure 5: Time evolution of 36 cars with p = 0

3.4 Traffic Jam with 12 cars and p = 0.2

With 36 total cars on the street and 12 in a jam initially, and $p = 0.2$, the result of 100 iterations is shown in figure (6).

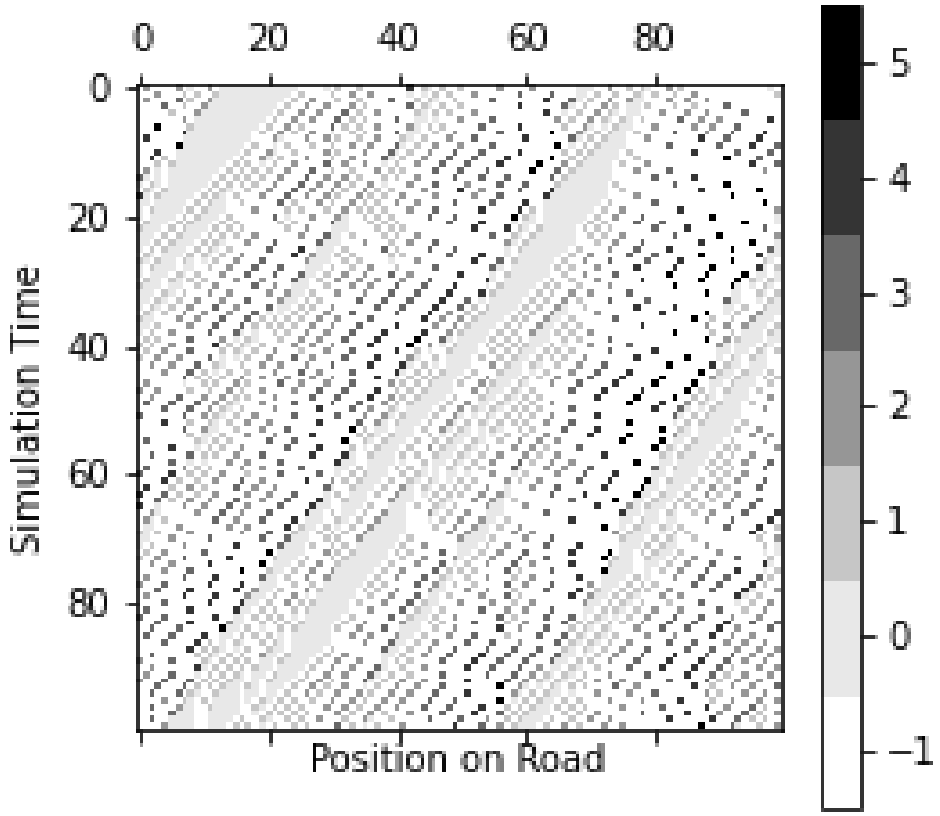

Figure 6: Time evolution of 36 cars with p = 0.2

4 Comparison and Conclusion

Figure (3) and figure (5) describe smooth flow of traffic compared to figure (4) and figure (6).This describes the fact that with $p = 0$, there is null to only a slim chance of traffic jam as the only possibility of cars slowing down is due to density of cars. Figure (4) and (6) describe rather realistic scenarios with $p = 0.2$. Herein, some cars slow down randomly and hence creating patches of traffic jams. This is more evident in figure (6) where the density of cars is more on the street than in figure (4).

A
Source Code

The general outline of the code looked like the following

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Lalit Chaudhary
Computational Physics
Project 4
"""
from pylab import *
import numpy as np
import matplotlib.pyplot as plt
from random import seed
from random import random
from random import randint
from datetime import datetime

N = 100          #No. of Cells
size = 7.5       #Size of each cell
vmax = 5         #Max speed of car
inf = -1         #Denotes empty cell
p = 0.2          #dawdle probability
n = 100          #No. of iterations to perform

#To generate random numbers
seed(datetime.now())

#Set up a street with N cells
def createStreet(N):
    S = []
    for i in range(N):
        S.append(inf)
    return S

#Count no. of empty cells after current position
def countEmpty(S, i):
    count = 0
    if (i+1 == N):
        i = -1
    while (S[i+1] == inf):
        i = i+1
        count = count+1
        if (i+1 == N):
```

```
            i = -1
    return count

#Nagel-Schreckenberg model of traffic flow
def NaSc (S, n):

    #To store resulting matrix
    result=np.zeros(shape=(n, N))
    result[0] = S
    t = 0

    while (t < n):                  #iterate n times

        for i in range(len(S)):
            if(S[i] == inf):        #do nothing on empty cells
                continue

            if(S[i] < vmax):        #Accelerate if not at max speed
                S[i] = S[i]+1

            d = countEmpty(S, i)
            S[i] = min(d, S[i])     #Prevent Collision

        for i in range(len(S)):     #Randomization with dawdle probability
            if(S[i] == inf):
                continue
            prob = random()
            if(prob < p and S[i]>0):
                S[i] = S[i] - 1

        Snew = [inf]*len(S)         #Store updated values in new array

        for i in range(len(S)):
            if(S[i] == inf):
                continue
            if (i+S[i] < N):         #make street Circular
                Snew[i+S[i]] = S[i]
            else:
                Snew[i+S[i]-N] = S[i]

        S=Snew.copy()               #pass udated array for next iteration
        result[t] = Snew            #Store updated array in matrix
        t = t+ 1
    return result

#For plotting the results
```

```
def Plot(result, title):
    #get discrete colormap
    cmap = plt.get_cmap('Greys', np.max(result)-np.min(result)+1)
    mat = plt.matshow(result,cmap=cmap,vmin = np.min(result)-.5,
                                        vmax = np.max(result)+.5)
    cax = plt.colorbar(mat, ticks=np.arange(np.min(result),np.max(result)+1))

    plt.xlabel('Position on Road')
    plt.ylabel('Simulation Time')
    plt.title(title, loc='center')
    plt.show()
    #plt.savefig(title)


#Traffic jam with 6 cars; p = 0
S = createStreet(N)
p = 0.0
pos = randint(0,80)             #position of traffic jam
for i in range(pos, pos+6):
    S[i] = 0

#12 other cars at random position
for a in range (12):
    x = randint(0, 99)
    while (S[x] != inf):
        x = randint(0, 99)
    S[x] = randint(0, 5)

r1 = NaSc(S, n)
Plot(r1, '18 Cars with p = 0')


#Traffic jam with 12 cars; p = 0
S2 = createStreet(N)
p = 0.0
pos = randint(0,80)             #position of traffic jam
for i in range(pos, pos+12):
    S2[i] = 0

#24 other cars at random position
for a in range (24):
    x = randint(0, 99)
    while (S2[x] != inf):
        x = randint(0, 99)
    S2[x] = randint(0, 5)
```

```
r2 = NaSc(S2, n)
Plot(r2, '36 Cars with p = 0')



#Traffic jam with 6 cars; p = 0.2

S3 = createStreet(N)
p = 0.2
pos = randint(0,80)             #position of traffic jam
for i in range(pos, pos+6):
    S3[i] = 0

#12 other cars at random position
for a in range (12):
    x = randint(0, 99)
    while (S3[x] != inf):
        x = randint(0, 99)
    S3[x] = randint(0, 5)

r3 = NaSc(S3, n)
Plot(r3, '18 Cars with p = 02')

#Traffic jam with 12 cars; p = 0.2
S4 = createStreet(N)
p = 0.2
pos = randint(0,80)             #position of traffic jam
for i in range(pos, pos+12):
    S4[i] = 0

#24 other cars at random position
for a in range (24):
    x = randint(0, 99)
    while (S4[x] != inf):
        x = randint(0, 99)
    S4[x] = randint(0, 5)

r4 = NaSc(S4, n)
Plot(r4, '36Cars with p = 02')
```