SIMULATION OF TRAFFIC FLOW

AUTHOR: CHENYUAN QU

STUDENT ID: 29975832

BSc report for PHYS6017
Computer techniques in Physics

Physics and Astronomy
University of Southampton
United Kingdom
15/04/2021

SIMULATION OF TRAFFIC FLOW

CHENYUAN QU

Abstract

This article did research on rules of traffic flow, by simulating *Nagel–Schreckenberg model*. The final state of this model was invested.By adjusting the parameters in *Nagel–Schreckenberg model*, we focused on the relationship between traffic flow and density, where some rules are founded. In the end, base on these rules, the prediction of traffic flow can be realized, and the referable optimization of the real traffic designing was proposed.

Contents

1 INTRODUCTION

Nowadays, traffic congestion is increasingly heavier especially in some metropolises, because that the development of traffic infrastructure and public transport is slowing down while the number of vehicles is increasing exponentially. Under this trend, *Intelligent Transportation System*[1] would be a great situation to deal with traffic jams and increase the efficiency of traffic infrastructure. It can also improve the safety of transport and reduce the traffic energy cost.[2] Traffic flow forecasting is the base of *Intelligent Transportation System*, which can provide the data foundation to schedule personal travel, control the traffic flow, program the traffic lights, and apply in many different fields. Meanwhile, the prediction of traffic flow needs massive data to analyse, which is unrealistic to access from real life. Therefore, constructing real and intelligent traffic models with evidence is significantly important.

2 THEORETICAL BACKGROUND

There are several kinds of models be popular currently, such as *vehicle-following model* [3], *cellular automaton traffic simulation* [4] and *fluid-dynamic traffic model*[5]. Among these theories, *cellular automaton traffic simulation* is one of programs which are most suitable for computer simulation because time, space and other quantities are assumed to be discrete in this kind of model. The most representative model of *cellular automaton traffic simulation* is called *Nagel–Schreckenberg model*
During the process of *Nagel–Schreckenberg model*, the action and movement of cars are both based on the set rules in one time cell. In this model, the road is a close circle, which is also called as periodic boundary conditions, and space and time are separated into many cells [6]. One cycle is run in one time cell and one space cell is empty or is occupied by one car. Related to the real world, one space cell may be treated as 7.5 meters and one time cell can be set as 1 second, and of course, they can be adjusted for the requirements. The key theory of *Nagel–Schreckenberg model* is highlighting two goals of drivers, arriving at the destination as soon as possible and trying to avoid hitting other cars. The rules are set absolutely depending on it while unexpected events are also considered. The specific rules will be mentioned later.

3 EXPERIMENTAL METHOD

For *Nagel–Schreckenberg model*, imagine that there is a circle road with some cars running on it. The road is divided into many sites, L, which is empty or be occupied by one car. The velocity of the car and time are also discrete in this model. Velocity can take an integer from 0 to speed limit, V_{max}, and time, t, also take integer steps. At the same time, the system updates by following four continuous rules within one time step. These rules are following:

1. Acceleration: Because every driver wants to arrive at the destination as quick as possible, the drive will accelerate the car as fast as they can. Therefore, the car will accelerate one velocity unit unless its speed has arrived at the maximum speed limit, V_{max}.
2. Avoiding collision: No car driver wants to hit the front car, so if the speed of one car is less than the distance between this car and its front car, this car will decelerate to the speed which is one space unit less than the distance.
3. Unexpected events: There are always unexpected events that happened during the driving period, which are maybe the view outside the window or chatting with the passenger. They will make the car slow down by one velocity unit. This rule is followed by the previous two rules.
4. Position update: The positions of cars will be updated based on the new speed of them.

As we can see, the first two rules show the key theory of *Nagel–Schreckenberg model* as I mentioned before, and the third law is a random noise which is trying to reproduce the accidents on the road.
At the beginning of the simulation, the cars will appear in random positions on the road and they are at rest. Then, they will follow the rules above and run it once after once until the system is stable, or called it the self-organised state, which will be ordered and conformed over all components of the system [7].
Then, we need to adjust the parameter of this model to start the simulation. Before it, we need to set the physical meaning for one time and space cell because we do not want that our parameters are too far away from reality. I would like to set one space cell to be 5 meters because one cell would only be

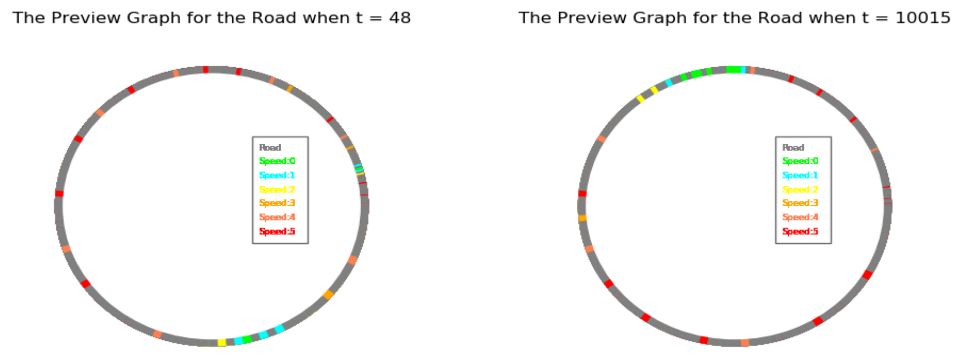
The Preview Graph for the Road when t = 48
The Preview Graph for the Road when t = 10015
Road
Speed:0
Speed:1
Speed:2
Speed:3
Speed:4
Speed:5

Figure 3.1: The preview graph for road in the beginning of simulation (left) and in the self-organised state(right)

filled by one car and the length of one normal car is between 4.2 meters to 4.8 meters. After considering the necessary safety distance, I supposed that 5 meters are suitable for one space unit. Turning to the time cell, I would like to set one time unit to be 2 seconds. The reason is that the reaction time of drivers needs to be considered.
After that, we can start to set the parameter. L = 200 would be suitable for the first try and the number of cars, N, can be regarded as 20, so the road will not be too busy or too free. The speed limit, V_{max}, can be 5 first, which is $45km/h$ like a city road. The probability of unexpected events can be 0.25. The road would be like Figure 3.1.

4 THEORETICAL METHOD

To perform the road in the code, we can set an array x standing for it. The empty sites are -1, and the occupied sites are the speed of the cars. So firstly, an empty road need be created:

```
# create empty road
x = np.zeros(L, int)
```

```
for u in range(L):
    x[i] = -1
```

Then initialize the road by adding cars in random position:

```
import random
car_positions = random.sample(range(L),N)  # the position index of cars

for car_position in car_positions:
    x[car_position] = 0
```

After that, we need to follow the rules to run the simulation. For these laws,

$$\begin{aligned} v &= v+1 \qquad if\ v < V_{max}, \\ v &= d-1 \qquad if\ v \geq d, \\ v &= v-1 \qquad if\ p < p_{set}\ and\ v > 0, \\ i_{t+1} &= i_t + v, \end{aligned}$$

where v is speed of car, d is the distance between this car and front car, p_{set} is the set probability of unexpected events which is 0.25 in the first try, i is the index of one car in the road.
For graphical display, I used two pie charts superimposing together to get that effect. After converting the array of road, x, to the list of colour, color_show, which the colour is corresponded to the road and speed of vehicles, the road can be shown as:

```
# main body of the road
pie_eq = []
for i in range(L):
    pie_eq.append(1 / L)
plt.pie(pie_eq, radius=1, colors=color_show)
# decoration part, white circle to make the road be like a road
plt.pie([1, 0, 0], radius=0.95, colors='w')
```

Here, we plot a pie chart to show the road situation by colour first, then use another white pie chart to shade the inside of pie chart to make it look like a circle road.

5 DISCUSSION OF RESULTS

5.1 SELF-ORGANISED STATE

Let us look back to the Figure 3.1. We can see that the lighter color means slower speed. If there are many lime, cyan, yellow color around one place, we can say that there is a traffic jam there.
During the simulation, we can see that the road in the beginning is more like the left graph of Figure 3.1. The speed of cars is more random. The slow vehicles would appear many places in the road. For example, in the left graph, the bottom and the top right of the road are both gathering the slower cars.
Nevertheless, after the run of enough time, the system is much more ordered. The traffic jam is appeared in few places even the density of cars is huge. Turning to the graph on the right, it is observable that the traffic jam is on the top of road. The speed of cars there is all nearly 0 and 1, and the other cars would drive as fast as speed limit.

Therefore, we can regard the latter state as be self-organised state, which shows some patterns which we need to find out. Hence, the following data is all collected when the system is in self-organised state.

5.2 TRAFFIC FLOW VERSES TRAFFIC DENSITY

Traffic flow is defined as how many cars pass one given point, which I set it as end point, per time unit, and the traffic density is that how many cars per site on the road, N/L.
Traffic flow is the variable which we need to measure and the traffic density is the variable needed to change. There are two ways to change the traffic density. One is changing the length of road, L, and the other is changing the number of cars in road, N.
Meanwhile, for each case, the probability and speed limit would be adjusted. The probability is set as 0.25, 0.5 and 0.75, while the speed limit would be 5, 10, 15 and 20. Probability value will help us to find which probability fits real case better. The different value of speed limit is used to investing which speed limit is best in the real life. Speed limit of 20 is a little unrealised case because not all of cars can reach $180km/h$. It is more likely to the road without an official highway speed limit.

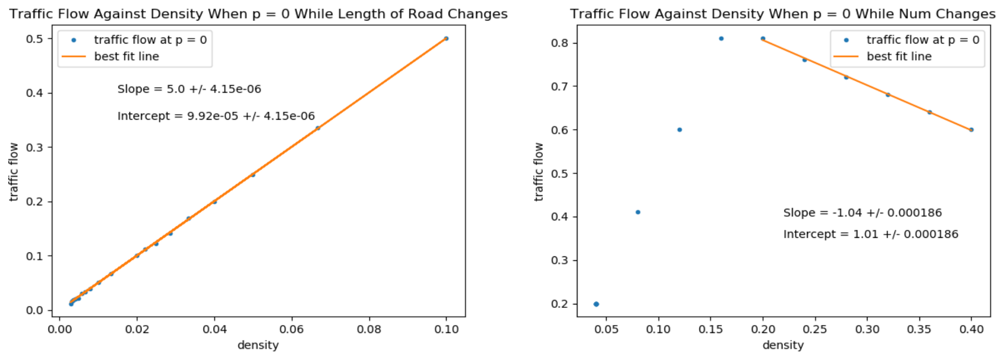

Figure 5.1: The graph between traffic flow and density when probability is 0 while the speed limit is 5. The blue spot is the traffic flow on corresponding density and orange line is the best fit line of the spots. The left one is while length of road is changing, and the right one is while number of cars is changing

5.2.1 CHANGING THE LENGTH OF ROAD

While changing the length of road, we need to keep number of cars be the same, like 20 cars. Then we record the traffic flow for length increasing by every 500 units from 500 to 10000. Next, use N/L to get the traffic density. If the probability equals to 0, the graph of it is shown as Figure 5.1(left).

From this graph, we can see that the trend between traffic density and flow is proportional, where intercept is neglect. The slope is 5 which is also value the speed limit, V_{max}. So when p = 0:

$$\text{Traffic flow} = V_{max} * \text{density} \qquad p = 0.$$

Repeat these steps with different probabilities and speed limits. Figure 5.2 and Figure 5.3 are shown.

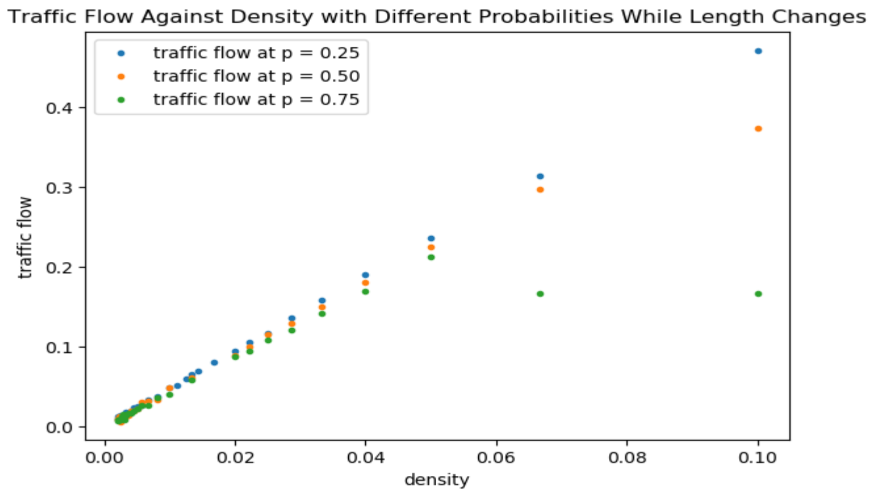

Figure 5.2: The graph between traffic flow and density with different probabilities when the length of road is changed. Blue spot is the traffic flow when probability is 0.25, orange spot is that when probability is 0.50, and green spot is that when probability is 0.75

In Figure 5.2, we can find that, although the trend of first several spots is still proportional, the last a few spots are tending to be stable. For example, in the case of $p = 0.75$, the spots whose densities are greater than 0.05, the traffic flow are all 0.181, which means a seriously traffic jam is created. Meanwhile, the slope of trend of first several spots is also changed, which shows in the Table 5.1. From the table, we can find some patterns between probability and slope. The error is neglect and the previous law can be complement as:

$$\text{Traffic flow} = (V_{max} - v_{decrease} * p) * \text{density}, \tag{5.1}$$

p	slope	error of slope
0.25	4.75	4.98e-06
0.50	4.50	4.00e-05
0.75	4.24	5.10e-05

Table 5.1: Slopes and their error for different probabilities while the length of road is changing

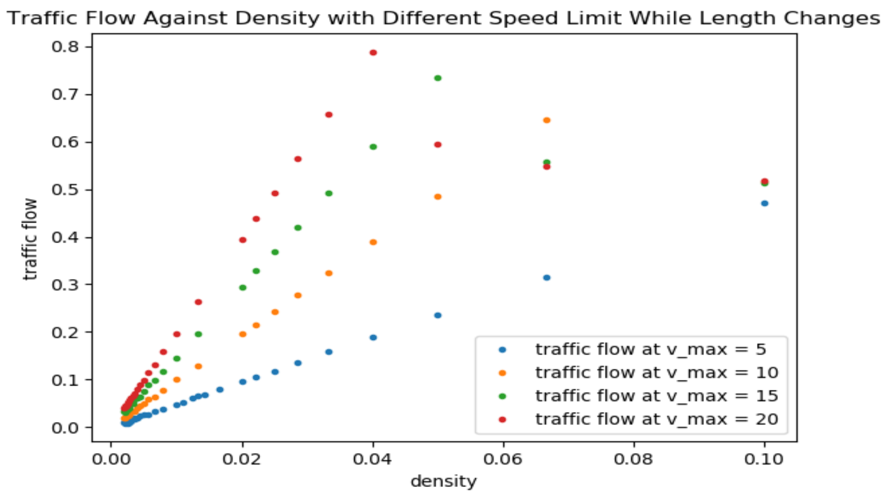

Figure 5.3: The graph between traffic flow and density with different speed limit when the length of road is changed while $p = 0.25$. Blue spot is the traffic flow when speed limit is 5, orange spot is that when speed limit is 10, green spot is that when speed limit is 15, and red spot is that when speed limit is 20.

V_{max}	slope	error of slope
5	4.75	4.98e-06
10	9.75	3.81e-06
15	14.75	6.66e-05
20	19.76	9.78e-04

Table 5.2: Slopes and their error for different speed limit while the number of car is changing

where p is the probability of unexpected events and $v_{decrease}$ is the decreased speed when unexpected events happened, when the data set is large enough.

We can conclude that the relationship between traffic flow and density is followed the equation above when the density of it is below the threshold value. When it is above the threshold value, the traffic flow would be stable at a constant value, which means that the traffic congestion has formed. When $p = 0$, the threshold value is not exist within 0 to 0.10 range, and it increases while the probability increases. Therefore, I suppose that probability of unexpected events is one of reason that traffic jam forms

Turning to Figure 5.3, we found similar pattern with the previous case. Besides at $V_{max} = 20$, the spots of all cases can be separated into two parts. First part shows the tendency be proportional again and second part tends to one constant. Let us see the gradient of first part first in Table 5.2. The answer fits the equation 5.1 properly, which shows the equation is general. Then, the second part is discussed. If we highlight the spot when density is 0.10, we found that the red, green and orange spots is in the same place, with traffic flow of 0.52. We look into the gradient of second parts of them, we can see when $V_{max} = 20$, the maximum traffic flow is highest, but after that, the dropping speed is fast as well. Because changing the length of road would not let density be quite large, we will discuss the second part mainly in next subsection.

5.2.2 CHANGING THE NUMBER OF CARS

We set the length of road be 500 unit this time, and record the traffic flow when number of cars, N, increase every 10 cars, stopping at 300 cars. Then, calculate traffic density and plot the graph with different probabilities and

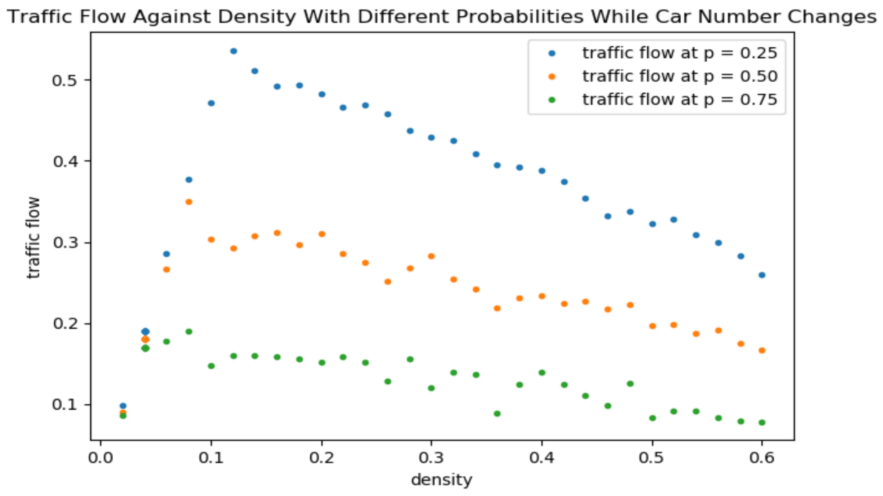

Figure 5.4: The graph between traffic flow and density with different probabilities when the number of cars is changed while $V_{max} = 5$. Blue spot is the traffic flow when probability is 0.25, orange spot is that when probability is 0.50, and green spot is that when probability is 0.75

speed limit again.

Firstly, let us look back to Figure 5.1(right) and see the data when $p = 0$. The maximum data is between 0.16 to 0.20. Before it, the spots are proportional again with slope of $5 \pm 3.98e-6$, matching the equation before. After the peak, benefiting the range of density extends, we can see that the trend of second part is a decreasing straight line. The slope of it is -1.04 ± 0.000186, which is close to 1, and the intercept is 1.01 ± 0.000186, which close to 1 either. Therefore, we can predict the equation beyond the threshold value is:

$$\text{Traffic flow} = -1 * \text{density} \qquad p = 0.$$

Then we should try different probability to see how to make our equation more general (Figure 5.4)

The spots are still separated into two sectors. First one is followed the ruled we mentioned in 5.5.1, which the slopes are 4.75, 4.50, 4.25 separately. The

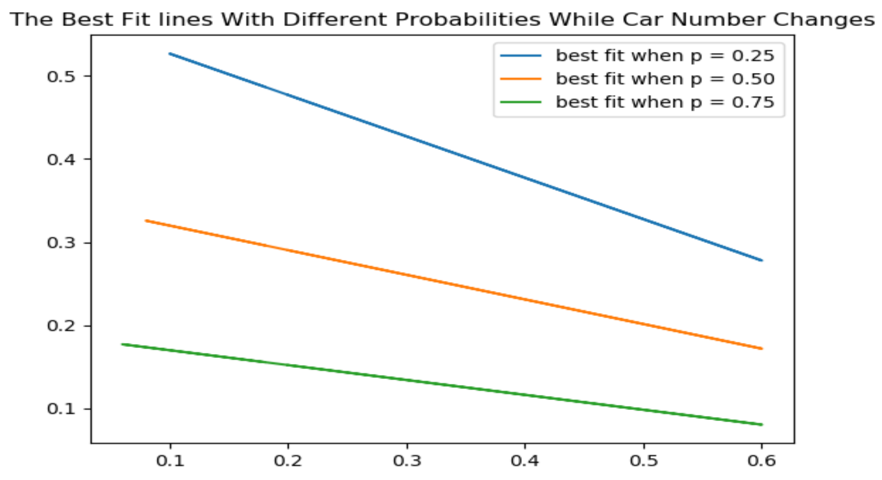

Figure 5.5: Best fit lines on traffic flow and density graph with different probabilities when the number of cars is changed while $V_{max} = 5$. Blue line is the best fit line at probability of 0.25, orange line is that at probability of 0.50, and green line is that at probability of 0.75

p	slope	error of slope	intercept	error of intercept
0.25	-0.498	0.00129	0.506	0.00129
0.50	-0.321	0.0088	0.335	0.0088
0.75	-0.187	0.00865	0.188	0.00865

Table 5.3: The data for best fit lines of Figure 5.5

second sector does also show an decreasing straight line. Let us ignore the spots and show the best fit lines for these three cases (Figure 5.5), and the data for these line is on Table 5.3.

From the table, we can see that the slope is tend to 0 while the probability is increasing. Meanwhile , the intercept is the opposite value of the gradient, but now we cannot decide whether speed limit, V_{max}, will in this equation, so we should plot the graph with different speed limit as Figure 5.6.

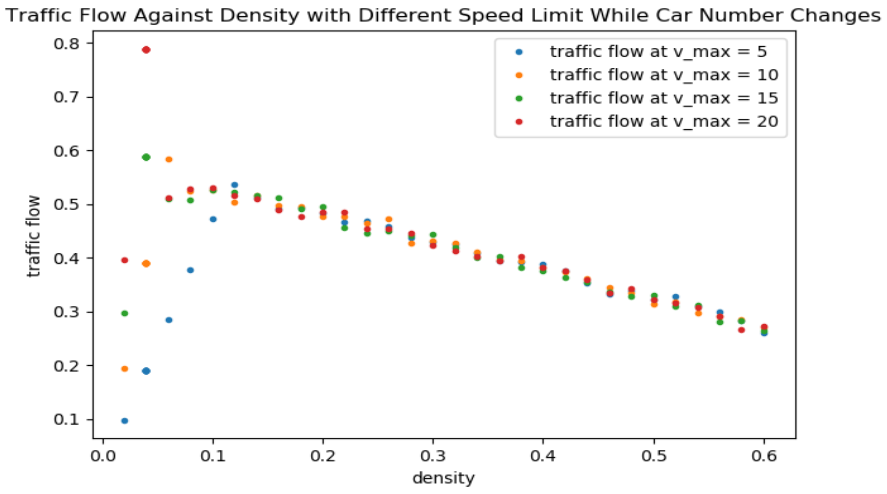

Figure 5.6: The graph between traffic flow and density with different speed limit when the length of road is changed while $p = 0.25$. Blue spot is the traffic flow when the speed limit is 5, orange spot is that when the speed limit is 10, green spot is that when the speed limit is 15, red spot is that when the speed limit is 20

This figure shows that that the gradients of decreasing parts are very close, which is round -0.5. Therefore, it tells us the speed limit would not involve in the equation:

$$\text{Traffic flow} = g - g * \text{density}, \tag{5.2}$$

where g is the absolute value of gradient increasing p.

Combine equation 5.1 and 5.2, we can get:

$$\begin{aligned} &\text{Traffic flow} = (V_{max} - v_{decrease} * p) * \text{density} \qquad \text{before threshold value,} \\ &\text{Traffic flow} = g - g * \text{density} \qquad \text{after threshold value.} \end{aligned} \tag{5.3}$$

5.3 PHYSICAL APPLICATION

We can apply Equation 5.3 to the real case. In the reality, country transport always wants that the number of cars pass the highway as many as possible, which means that the traffic flow need be as huge as possible. Once we know the real probability of unexpected events of local residents, we can look up the gradient of the second equation of Equation 5.3, and adjust the speed limit to get the slope of first equation. Combine both of them to meet the requirement. For example, if one highway needs to carry 50 cars per minute, we can calculate out the density. Set this density with some flexible space to the threshold value. The set the speed limit of the highway to meet the value. Or, on the other way, if one highway has a speed limit, we can use *Intelligent Transportation System* to do the traffic macro-control. Lead suitable number of cars to this highway to avoid congestion.

6 CONCLUSION

1. In *Nagel–Schreckenberg model*, the system would run from a disordered state to a self-organised state. After enough time of the run, the system would show some interesting patterns like traffic congestion and the following rules.

2. The relationship between traffic flow and density was invested. The trend of spots can be separated into the increasing part and decreasing part. We can change the density by adjusting the length of road to focus

on the increasing part. The increasing part shows the proportional property. The slope is equal to V_{max} when the probability of unexpected events,p, equals 0. Then we can modify the density by number of cars on the road to zoom in the decreasing part. The trend of it is a decreasing line with the slope of -1 when $p = 0$. This phenomenon shows that even there is no external factor, the traffic jam will still emerge because of crowded road.

3. After adding different probabilities and speed limits, we found that the probability and speed limits both affect the slope of increasing part either. After some predictions and verification, the slope of increasing data is $V_{max} - v_{decrease} * p$, where V_{max} is the speed limit and $v_{decrease}$ is the decreased speed when unexpected event happens. We also noticed that the slope of decreasing data is only affected by probabilities but not affected by the speed limit. When probability is getting greater, the gradient is getting smaller.

4. This model can be used as refer in daily life. The equation above can be used to calculate the threshold carry capacity or setting the speed limit to avoid traffic jams.

References

[1] Monahan, Torin
"War Rooms" of the Street: Surveillance Practices in Transportation Control Centres
The Communication Review. 10 (4): 367–389. CiteSeerX 10.1.1.691.8788.(2007)
doi:10.1080/10714420701715456. S2CID 44101831.

[2] SIOR, Social Impact Open Repository
Reducing delay due to traffic congestion. [Social Impact]. ITS. The Intelligent Transportation Systems Centre and Testbed"
http://sior.ub.edu/jspui/cris/socialimpact/socialimpact00438
(05/09/2017)

[3] Chenggang Li, Xiaobei Jiang, Wuhong Wang, Qian Cheng, Yongjun Shen

A Simplified Car-Following Model Based on the Artificial Potential Field
https://core.ac.uk/download/pdf/82041059.pdf (2016)

[4] University of Delaware, Dept. of Civil Engineering
Cellular Automaton Traffic Simulation
http://udel.edu/ mm/traffic/ca.html (10/04/2021)

[5] Gabriella Bretti, Roberto Natalini, Benedetto Piccoli
A Fluid-Dynamic Traffic Model on Road Networks
Archives of Computational Methods in Engineering 14(2):139-172
DOI:10.1007/s11831-007-9004-8 (June 2007)

[6] K Nagel and M Schreckenberg
Journal de Physique I: vol 2, page 2221-2229
(Dec 1992)

[7] Ashby, W. R. *Principles of the self-organizing system*, pp. 255–78 in Principles of Self-Organization. Heinz von Foerster and George W. Zopf, Jr. (eds.) U.S. Office of Naval Research.(1962)

7 APPENDIX

7.1 SIMULATION AND ANALYSIS CODE

```
# Import Section
import numpy as np
import random
import os
import re
from matplotlib import pyplot as plt
from matplotlib.patches import Rectangle

plt.ion()  # Turn the interactive mode on


# Function Section
################# Running Part ###################
def create_empty_road():
    """
    Create an empty road
    :return: an empty road
    """
    x = np.zeros(L, int)
    for i in range(L):
        x[i] = -1
    return x


def initialize_road():
    """
    Initialize the road by using parameters
    :return: initial road
    """
    x = create_empty_road()

    car_position = random.sample(range(L), N)  # set the cars' position randomly

    for car in car_position:
```

```python
            x[car] = 0
    return x


def accelerate_to_speed_limit(x):
    """
    First rule: every car will accelerate up to the speed limit
    :param x: the road and cars with initial speed
    :return: the road and cars with accelerated speed
    """
    x_new = []
    for cell in x:
        if cell != -1 and cell < V_max:  # if the speed of the car
                                         # does not reach the maximum speed,
            cell += 1                    # the car will accelerates.
        x_new.append(cell)
    return x_new


def distance_detect(x, index_cell):
    """
    Detect the distance between current car and front car
    :param x: the road and cars
    :param index_cell: the index of current car
    :return: the distance between current car and front car
    """
    distance = 1
    while distance:
        front_index = index_cell + distance
        if front_index >= L:  # if the index is beyond the maximum length,
            front_index -= L  # it will return to the beginning
                              # because the road is a closed circle
        if x[front_index] != -1:  # if the front car is detected,
            break                 # jump out the loop
        distance += 1
    return distance
```

```python
def avoid_hit(x):
    """
    Second rule: the drivers do not want to hit the car in front
    :param x: the road and cars with initial speed
    :return: the road and cars with speed which may be adjusted
    """
    x_new = []
    for index_cell, cell in enumerate(x):
        if cell != -1:
            # get the distance between this car and front car
            distance = distance_detect(x, index_cell)
            if cell >= distance:  # if front car is close,
                cell = distance - 1  # the car will decelerate to avoid hitting.
        x_new.append(cell)
    return x_new


def unexpected_events(x):
    """
    Third rule: After the former two rules, unexpected events cause
    slowing down.
    :param x: the road and cars with initial speed
    :return: the road and cars with speed under unexpected events
    """
    x_new = []
    for cell in x:
        if cell > 0 and random.random() <= p:  # if the speed of car is not 0,
            cell -= 1  # it is possible to slow down due to unexpected events.
        x_new.append(cell)
    return x_new


def move_forward(x):
    """
    In the end of one cycle, the car will move forward based on their speed.
    :param x: the road and cars at the initial positions
    :return x_new: the road and cars at the final positions;
    :return tf_count: the count for traffic flow when the cars pass
```

```
                      the end point.
    """
    x_new = create_empty_road()
    tf_count = 0

    for index_cell, cell in enumerate(x):
        if cell != -1:
            new_index = index_cell + cell
            if new_index >= L:  # because the road is a closed circle again
                new_index -= L
                tf_count += 1
            x_new[new_index] = cell
    return x_new, tf_count


def one_cycle(x):
    """
    The whole process during one unit time
    :param x: the initial road and cars with speed
    :return x_new: the new road an cars with speed
    :return tf_count: the count for traffic flow when the cars pass
                      the end point.
    """
    x_accelerate = accelerate_to_speed_limit(x)
    x_avoid = avoid_hit(x_accelerate)
    x_unexpected = unexpected_events(x_avoid)
    x_new, tf_count = move_forward(x_unexpected)
    return x_new, tf_count


def change_to_color(x):
    """
    Change the array to the colour to paint the graph
    :param x: the array of road and cars
    :return: the list of colour with same order of cars
    """
    color_show = []
    for item in x:
        if item == -1:
            color_show.append('gray')  # road color is gray
        elif item == 0:
            color_show.append('lime')  # speed 0 color is lime
        elif item == 1:
            color_show.append('cyan')  # speed 1 color is cyan
        elif item == 2:
            color_show.append('yellow')  # speed 2 color is yellow
        elif item == 3:
            color_show.append('orange')  # speed 3 color is orange
        elif item == 4:
            color_show.append('coral')  # speed 4 color is coral
        elif item == 5:
            color_show.append('red')  # speed 5 color is red
    return color_show


def plot_section(x, t):
    """
    The function to plot the road and cars with colours
    :param x: the array of the road and cars
    :return: True if all clear
    """
    color_show = change_to_color(x)  # change the array to colour parameter

    title = 'The Preview Graph for the Road when t = ' + str(t)
    plt.title(title)

    # label part
    plt.text(0.3, 0.4, 'Road', color='gray', fontsize=6)
    plt.text(0.3, 0.3, 'Speed:0', color='lime', fontsize=6)
    plt.text(0.3, 0.2, 'Speed:1', color='cyan', fontsize=6)
    plt.text(0.3, 0.1, 'Speed:2', color='yellow', fontsize=6)
    plt.text(0.3, 0, 'Speed:3', color='orange', fontsize=6)
    plt.text(0.3, -0.1, 'Speed:4', color='coral', fontsize=6)
    plt.text(0.3, -0.2, 'Speed:5', color='red', fontsize=6)

    # plot the road
    # main body of the road
    pie_sep = []
    for i in range(L):
        pie_sep.append(1 / L)
    plt.pie(pie_sep, radius=1, colors=color_show)
    # decoration part, white circle to make the road be like a road
    plt.pie([1, 0, 0], radius=0.95, colors='w')

    ax = plt.gca()  # decoration part,
    ax.add_patch(Rectangle((0.26, -0.26), 0.35, 0.76, color='dimgray', fill=False

    plt.pause(0.1)  # After showing a graph, the graph pauses for 0.1 second.

    return True


def record_traffic_flow(x):
    """
    Record the traffic flow for each cycle.
    :param x: road and cars with speed
    :return: True if successive
    """
    t = 0
    traffic_flow = []  # create an empty list to save the data of traffic flow

    while not t == 1000:  # run the simulation 1000 times to reduce the error
        t += 1
        x, tf_count = one_cycle(x)
        traffic_flow.append(tf_count)

    text_file_name = (str(L) + '_' + str(N) + '_' + str(V_max) +
                      '_' + str(p) + '_' + 'tf.txt')
    # create the txt file with name
    traffic_flow_text_file = open(text_file_name, 'w+')
    # write the average value on the first line
    traffic_flow_text_file.write(str(np.mean(traffic_flow)))

    # record the data for each run in case of further research
```

```python
    for one_traffic_flow in traffic_flow:
        traffic_flow_text_file.write(str(one_traffic_flow))
        traffic_flow_text_file.write('\n')

    traffic_flow_text_file.close()

    return True


################# Analysis Part ##################
def analysis():
    """
    Analysis main function
    :return: True if successive
    """
    plt.ioff()
    filelist = findall_file()  # find all file in current path

    # plot the graph of the density against traffic flow
    # with different probability when length of road is changing.
    length_change_with_different_probability(filelist)
    # with different speed limit when length of road is changing.
    length_change_with_different_speed(filelist)
    # with different probability when number of cars is changing.
    car_number_change_with_different_probability(filelist)
    # with different speed limit when number of cars is changing.
    car_number_change_with_different_speed(filelist)
    return True


def findall_file():
    """
    Find all file names in current path
    :return: list of all files
    """
    filepath = "./"  # current path
    filelist = []
    for _, _, file in os.walk(filepath):
            filelist.append(file)
    return filelist


def pattern_fit_files(filelist, pattern):
    """
    Find out the files which is fitted to the pattern
    :param filelist: list of all files
    :param pattern: string which is the pattern need to be fitted
    :return: list of all files we need
    """
    file_needed = []
    for type in filelist:  # because os.walk classifies the files
                           # with their file type
        for item in type:  # match for each files
            m = pattern.findall(item)
            if not m == []:
                file_needed.append(m[0])
    return file_needed


def get_tf_value(file):
    """
    Open the file and return the average traffic flow in it.
    :param file: the name of the file we needed
    :return: average traffic flow
    """
    f = open(file)
    contain = []
    for line in f.readlines():
        contain.append(float(line.replace('\n', '')))
    return contain[0]  # average traffic flow is on first line


def round_to_3_sf(data):
    """
    Standardize the data to 3 significant figures
    :param data: float of raw data
    :return: string with 3 significant figures
    """
    return '%s' % float('%.3g' % data)


def plot_fitting_line_length(density, tf, range, label):
    """
    Plot the best fit line within the range
    :param density: cars per site on the road
    :param tf: traffic flow, cars passing the end point
    :param range: list with 2 elements, first one is the starting point of
                  best fit line, second one is the end point.
    :param label: label for best fit line
    :return slope_text: string for the slope of best fit line
    :return intercept_text: string for the intercept of best fit line
    """
    density_in_range = []
    tf_in_range = []
    for index in range(len(density)):
        # we need the density only within the range to plot a best fit line
        if range[0] < density[index] < range[1]:
            density_in_range.append(density[index])
            # we need to record the corresponding traffic flow data
            tf_in_range.append(tf[index])

    # get the data and error for the best fit line
    fitting_data, fitting_error = np.polyfit(density_in_range, tf_in_range,
                                             1, cov=True)
    fitting_y = []
    for one_desity in density_in_range:
        fitting_y.append(one_desity * fitting_data[0] + fitting_data[1])
    plt.plot(density, fitting_y, label=label)

    # create the text for showing the slope and intercept with their error
    slope_text = ('Slope = ' + round_to_3_sf(fitting_data[0]) + ' +/- ' +
        round_to_3_sf(abs(min(fitting_error[0][0], fitting_error[0][1]))))
    intercept_text = ('Intercept = ' + round_to_3_sf(fitting_data[1]) + ' +/- '
        + round_to_3_sf(abs(min(fitting_error[1][0], fitting_error[1][1]))))
```

```python
    return slope_text, intercept_text


def get_density_and_tf_length(file_needed, pattern):
    """
    Return the density and traffic flow while length of road is changing.
    :param file_needed: name of the file which is waiting to be processed
    :param pattern: string which is the pattern need to be fitted
    :return: the density and traffic flow
    """
    density = []
    tf = []
    for item in file_needed:
        one_tf_value = get_tf_value(item)  # get the average traffic flow
        tf.append(one_tf_value)            # in this file
        length = item.replace(pattern, '')  # get the corresponding
        density.append(20 / int(length))    # length of road

    return density, tf


def get_density_and_tf_car_number(file_needed, pattern):
    """
    Return the density and traffic flow while number of cars
    in the road is changing.
    :param file_needed: name of the file which is waiting to be processed
    :param pattern: string which is the pattern need to be fitted
    :return: the density and traffic flow
    """
    density = []
    tf = []
    for item in file_needed:
        one_tf_value = get_tf_value(item)  # get the average traffic flow
        tf.append(one_tf_value)            # in this file
        car_number_raw = item.replace('500_', '')  # get the corresponding
        car_number = car_number_raw.replace(pattern, '')  # length of road
        density.append(int(car_number) / 500)
    return density, tf


def plot_density_tf_graph_probability(density_025, density_050, density_075,
                                      tf_025, tf_050, tf_075):
    """
    Plot the traffic flow against density graph with different probability.
    :param density_025: density at probability of 0.25
    :param density_050: density at probability of 0.50
    :param density_075: density at probability of 0.75
    :param tf_025: traffic flow at probability of 0.25
    :param tf_050: traffic flow at probability of 0.50
    :param tf_075: traffic flow at probability of 0.75
    :return: True if successive
    """
    plt.plot(density_025, tf_025, '.', label='traffic flow at p = 0.25')
    plt.plot(density_050, tf_050, '.', label='traffic flow at p = 0.50')
    plt.plot(density_075, tf_075, '.', label='traffic flow at p = 0.75')
    plt.xlabel('density')
    plt.ylabel('traffic flow')

    return True


def plot_density_tf_graph_speed(density_5, density_10, density_15,
                                density_20, tf_5, tf_10, tf_15, tf_20):
    """
    Plot the traffic flow against density graph with different speed limit.
    :param density_5: density at speed limit of 5
    :param density_10: density at speed limit of 10
    :param density_15: density at speed limit of 15
    :param density_20: density at speed limit of 20
    :param tf_5: traffic flow at speed limit of 5
    :param tf_10: traffic flow at speed limit of 10
    :param tf_15: traffic flow at speed limit of 15
    :param tf_20: traffic flow at speed limit of 20
    :return: True if successive
    """
    plt.plot(density_5, tf_5, '.', label='traffic flow at v_max = 5')
    plt.plot(density_10, tf_10, '.', label='traffic flow at v_max = 10')
    plt.plot(density_15, tf_15, '.', label='traffic flow at v_max = 15')
    plt.plot(density_20, tf_20, '.', label='traffic flow at v_max = 20')
    plt.xlabel('density')
    plt.ylabel('traffic flow')

    return True


def length_change_with_different_probability(filelist):
    """
    Main function to plot the traffic flow against density graph with
    different probability
    while the length of road is changing.
    :param filelist: list of all files
    :return: True if successive
    """
    # Find all file names with matched
    pattern_025 = re.compile(r'\d+_20_5_0.25_tf.txt')
    file_needed_025 = pattern_fit_files(filelist, pattern_025)
    pattern_050 = re.compile(r'\d+_20_5_0.5_tf.txt')
    file_needed_050 = pattern_fit_files(filelist, pattern_050)
    pattern_075 = re.compile(r'\d+_20_5_0.75_tf.txt')
    file_needed_075 = pattern_fit_files(filelist, pattern_075)

    # get the density and traffic flow for each of them
    density_025, tf_025 = get_density_and_tf_length(file_needed_025,
                          '_20_5_0.25_tf.txt')
    density_050, tf_050 = get_density_and_tf_length(file_needed_050,
                          '_20_5_0.5_tf.txt')
    density_075, tf_075 = get_density_and_tf_length(file_needed_075,
                          '_20_5_0.75_tf.txt')

    # Plot the graph
    title = ('Traffic Flow Against Density with Different Probabilities
            While Length Changes')
    plt.title(title)
    plot_density_tf_graph_probability(density_025, density_050, density_075,
                                      tf_025, tf_050, tf_075)
    # if best_fitting_para is True:
        # plot_fitting_line_length(density, tf)
    plt.legend()
    plt.show()

    return True


def length_change_with_different_speed(filelist):
    """
    Main function to plot the traffic flow against density graph with
    different speed limit
    while the length of road is changing.
    :param filelist: list of all files
    :return: True if successive
    """
    # Find all file names with matched
    pattern_5 = re.compile(r'\d+_20_5_0.25_tf.txt')
    file_needed_5 = pattern_fit_files(filelist, pattern_5)
    pattern_10 = re.compile(r'\d+_20_10_0.25_tf.txt')
    file_needed_10 = pattern_fit_files(filelist, pattern_10)
    pattern_15 = re.compile(r'\d+_20_15_0.25_tf.txt')
    file_needed_15 = pattern_fit_files(filelist, pattern_15)
    pattern_20 = re.compile(r'\d+_20_20_0.25_tf.txt')
    file_needed_20 = pattern_fit_files(filelist, pattern_20)

    # get the density and traffic flow for each of them
    density_5, tf_5 = get_density_and_tf_length(file_needed_5,
                      '_20_5_0.25_tf.txt')
    density_10, tf_10 = get_density_and_tf_length(file_needed_10,
                        '_20_10_0.25_tf.txt')
    density_15, tf_15 = get_density_and_tf_length(file_needed_15,
                        '_20_15_0.25_tf.txt')
    density_20, tf_20 = get_density_and_tf_length(file_needed_20,
                        '_20_20_0.25_tf.txt')
    # Plot the graph
    title = ('Traffic Flow Against Density with Different Speed Limit
            While Length Changes')
    plt.title(title)
    plot_density_tf_graph_speed(density_5, density_10, density_15,
                                density_20, tf_5, tf_10, tf_15, tf_20)
    # if best_fitting_para is True:
        # plot_fitting_line_length(density, tf)
    plt.legend()
    plt.show()

    return True


def car_number_change_with_different_probability(filelist):
    """
    Main function to plot the traffic flow against density graph with
    different probability
    while the number of cars is changing.
    :param filelist: list of all files
    :return: True if successive
    """
    # Find all file names with matched
    pattern_025 = re.compile(r'500_\d+_5_0.25_tf.txt')
    file_needed_025 = pattern_fit_files(filelist, pattern_025)
    pattern_050 = re.compile(r'500_\d+_5_0.5_tf.txt')
    file_needed_050 = pattern_fit_files(filelist, pattern_050)
    pattern_075 = re.compile(r'500_\d+_5_0.75_tf.txt')
    file_needed_075 = pattern_fit_files(filelist, pattern_075)

    # get the density and traffic flow for each of them
    density_025, tf_025 = get_density_and_tf_car_number(file_needed_025,
                                                         '_5_0.25_tf.txt')
    density_050, tf_050 = get_density_and_tf_car_number(file_needed_050,
                                                         '_5_0.5_tf.txt')
    density_075, tf_075 = get_density_and_tf_car_number(file_needed_075,
                                                         '_5_0.75_tf.txt')
    # Plot the graph
    title = 'Traffic Flow Against Density With Different Probabilities
    While Car Number Changes'
    plt.title(title)
    plot_density_tf_graph_probability(density_025, density_050, density_075,
                                      tf_025, tf_050, tf_075)
    # if best_fitting_para is True:
        # plot_fitting_line_length(density, tf)
    plt.legend()
    plt.show()

    return True


def car_number_change_with_different_speed(filelist):
    """
    Main function to plot the traffic flow against density graph
    with different speed limit
    while number of cars is changing.
    :param filelist: list of all files
    :return: True if successive
    """
    # Find all file names with matched
    pattern_5 = re.compile(r'500_\d+_5_0.25_tf.txt')
    file_needed_5 = pattern_fit_files(filelist, pattern_5)
    pattern_10 = re.compile(r'500_\d+_10_0.25_tf.txt')
    file_needed_10 = pattern_fit_files(filelist, pattern_10)
    pattern_15 = re.compile(r'500_\d+_15_0.25_tf.txt')
    file_needed_15 = pattern_fit_files(filelist, pattern_15)
    pattern_20 = re.compile(r'500_\d+_20_0.25_tf.txt')
    file_needed_20 = pattern_fit_files(filelist, pattern_20)

    # get the density and traffic flow for each of them
    density_5, tf_5 = get_density_and_tf_car_number(file_needed_5,
                                                    '_5_0.25_tf.txt')
    density_10, tf_10 = get_density_and_tf_car_number(file_needed_10,
                                                      '_10_0.25_tf.txt')
    density_15, tf_15 = get_density_and_tf_car_number(file_needed_15,
                                                            '_15_0.25_tf.txt')
    density_20, tf_20 = get_density_and_tf_car_number(file_needed_20,
                                                            '_20_0.25_tf.txt')

    # Plot the graph
    title = 'Traffic Flow Against Density with Different Speed Limit
            While Car Number Changes'
    plt.title(title)
    plot_density_tf_graph_speed(density_5, density_10, density_15,
                                density_20, tf_5, tf_10, tf_15, tf_20)
    # if best_fitting_para is True:
        # plot_fitting_line_length(density, tf)
    plt.legend()
    plt.show()

    return True


# Setting Section
"""
PLEASE READ

L, N, V_max, p and t_pre_run_max are adjustable parameters for simulation,
which are also preset.

If you want to see the process of simulating the model from beginning,
text_para_pre_run or/and graphical_para_pre_run need be set to True.
graphical_para_pre_run is recommended for accessing the process of model
visually.

If you want to see the process when the model is in the stable state,
graphical_para_final need be set to True.

If you want to save traffic flow data for further research,
traffic_flow_save_para need to be set to True.
Maybe you need to run the simulation several time with adjusting the
simulation parameters to access different data.
```

```
If you want to analysis data,
analysis_para need to be set to True.
only_analysis_para is provided to skip the simulation process.
If you also need best fit lines in your plotting graph, set
best_fitting_para be True.
"""""""""""""""""""""

# simulation parameters
global L, N, V_max, p
L = 200  # length of road
N = 30  # number of cars in the road
V_max = 5  # maximum speed of cars
p = 0.25  # probability of unexpected events
t_pre_run_max = 10000  # the maximum time of pre-run

# text and graphical output
global text_para_pre_run, graphical_para_pre_run, graphical_para_final
text_para_pre_run = False  # whether show the array of road when it is
pre-run, show is True.
graphical_para_pre_run = False  # whether show the graph of road when it is
pre-run , show is True.
graphical_para_final = True  # whether show the graph of road when
the system is in the final state, show is True.

# Data save
traffic_flow_save_para = False  # whether record the data of traffic flow

# Analysis parameter
analysis_para = False # if need analysis, show is True
only_analysis_para = False  # if only need analysis and not run the
simulation, show is True
if only_analysis_para is True:
    analysis_para = True

global best_fitting_para
best_fitting_para = True  # if we need best fit line, show is True
# Main Body
if __name__ == '__main__':
```

```
if only_analysis_para is False:  # See whether need to run the simulation
    t = 0  # time is 0 in the beginning
    x = initialize_road()  # create an initial road
    if graphical_para_pre_run is True:  # whether show the graph
        plot_section(x, t)
    if text_para_pre_run is True:  # whether show the array
        print(x)

    while not t == t_pre_run_max:
    # stop when the time reaches the maximum time for pre-run
        t += 1  # time increases one each cycle
        x, _ = one_cycle(x)  # run the logic

        if graphical_para_pre_run is True:  # whether show the graph
            plot_section(x, t)
        if text_para_pre_run is True:  # whether show the array
            print(x)

    if graphical_para_final is True:
        for time in range(100):
            t+=1
            x, _ = one_cycle(x)
            plot_section(x, t)
if traffic_flow_save_para is True:
# see whether need to save the traffic flow
    # save the data is better, because it does not require us
    # to run the simulation again and again.
    record_traffic_flow(x)

if analysis_para is True:  # see whether need to run analysis part
    analysis()  # use the data we saved to do analysis

exit()
```