Data Analytics MSc Dissertation MTHM038, 2020/21

Disquisitiones Arithmeticæ

Project on traffic jams according to the Nagel-Schreckenberg model

Xinyu Wang, ID 200459387

Supervisor: Dr. Wolfram Just

A thesis presented for the degree of Master of Science in *Msc Data Analytics*

School of Mathematical Sciences

Queen Mary University of London

Declaration of original work

This declaration is made on September 19, 2021.

Student's Declaration: I Xinyu Wang hereby declare that the work in this thesis is my original work. I have not copied from any other students' work, work of mine submitted elsewhere, or from any other sources except where due reference or acknowledgement is made explicitly in the text, nor has any part been written for me by another person.

Referenced text has been flagged by:

1. Using italic fonts, **and**
2. using quotation marks "...", **and**
3. explicitly mentioning the source in the text.

Acknowledgements

Firstly, I am very grateful to my supervisor Dr. Wolfram Just, who carefully explained the underlying theory of the model, provided me with research ideas and helped me with any difficulties I encountered during the completion of this project. His knowledge and patience have been very supportive.
Secondly, I would like to thank my parents and my boyfriend. My parents, Mr Hongbin Wang and Ms Jing Gong have always been unconditionally tolerant and understanding, always encouraging me to pursue my dreams. When I wanted to quit my job to continue my studies, they did not put any pressure on me and supported my choice unconditionally. They provided me with financial support and eased my psychological burden to choose my future direction freely and comfortably. My boyfriend, Mr Yakuo Li, has always been quietly supportive of my choices and has put me first in planning our future. When I was confused about my future, He encouraged me to come to the UK to study and choose a subject I was passionate about. Although I was separated from him for a short year due to studying abroad, he was always there to support me and give me a lot of motivation. My parents and boyfriend have given me the utmost care during my year of study, and their positive support has always been a source of energy for me, even though we are miles apart.
Once again, I would like to thank my friends, Xinyu Wang (who has the same name as me) , Zhangling Li and Yuxuan Zhang, for helping me with my life and my studies. Especially when I was going through illness and isolation, they helped me buy medicine and encouraged me to be optimistic to help me get through that difficult time. Their kindness, optimism and positivity have always influenced me. They are some of the most precious friends in life.
Finally, I would like to say that I was unfortunate to have studied and been infected with Covid-19 when it was epidemic, which took a lot out of my learning experience. But I was also fortunate to feel the care of my school, family, lover and true friends at such a particular time. Life is always full of unknowns, but with their support I have been able to push on through.

Abstract

Traffic jams are a common phenomenon in life and occur not only on roads but also on conveyor belts, signal transmissions, etc. Algorithms that simulate traffic can increase our understanding of various traffic phenomena. This project is based on the Nagel Schreckenberg model, invented by Kai Nagel and Michael Schreckenberg in 1992, which takes imperfect driving behaviour into account, is compiled using python 3 ,and the results are visualised. Secondly, the simulation results were used to observe traffic congestion formation and the traffic setback that occurs during congestion. The relationship between traffic flow and traffic density was then analysed by quantifying the traffic flow by selecting the number of vehicles passing a particular station per unit time and the total vehicle speed. And the simulation results are compared with the actual situation to analyse at what vehicle density the traffic flow reaches its maximum. The project uses the random package to add random results and the matplotlib package to visualise the data collected during the simulation. This article includes the complete code and its explanation, including the logic of the model and the explanation of the variables individually.

Contents

Chapter 1

Introduction

As economic life has become better, vehicles have become the primary means of transport. At the same time, as cars make travel convenient, they inevitably cause traffic problems. Traffic jams are not only annoying, but they do have a high economic cost in terms of wasted energy, air pollution and reduced efficiency of urban operations. Traffic jams happen naturally, for example, during rush hour, but they are not confined to car traffic. Situations similar to traffic jams can also occur in life, for example, on the internet, on production lines, or even in our DNA (affecting our lives on a very fundamental level). Therefore, understanding and modelling traffic jams from a mathematical perspective have far-reaching implications. Algorithms that simulate traffic can increase our understanding of various traffic phenomena. A proper simulation model helps us to make predictions about traffic conditions in advance and to respond ahead of time to optimise the traffic system and improve its efficiency. An important model is the Nagel Schreckenberg model, invented by Kai Nagel and Michael Schreckenberg in 1992 in addition to simulating typical driving behaviour, innovatively takes into account the instability of human behaviour and explains the spontaneous formation of traffic jams.[5]We chose the Nagel Schreckenberg model as the basis for our project. To make the simulation simple enough and easy to understand, we wanted it to show in an easy-to-understand way the changes that occur in vehicles during traffic jams. Therefore, we compiled the project in python 3 and visualised the results. This project focuses on the relationship between traffic flow and traffic density. We choose suitable indicators to measure the traffic flow and observe how it changes as the traffic density increases. Through our analysis, we hope to find the point that maximises the traffic flow, which will help us to plan road traffic wisely in real life.
We will introduce the principles of the Nagel-Schreckenberg model in the first chapter, in-

cluding the basic vehicle actions of acceleration and deceleration, in addition to explaining how random human behaviour can affect traffic and how it incorporate into the model. An in-depth explanation of the code for the basic simulation model give in chapter 3. Specifically, all variables, functions and logic will be explained. The analysis of the simulation results, how to quantify the size of the traffic flow and how it relates to traffic density will be presented in chapter 4. The simulation results will be compared with the actual situation. We will eventually conclude in chapter 5.

Chapter 2

Theory of the model

2.1 Objectives of the model

The aim of this project is to build a simplified model through code that can simulate some traffic phenomena caused by realistic vehicles in motion. We can use the simulation results to research the relationship between traffic flow and traffic density, which can provide some inspiration for solving the problem of vehicle congestion in everyday life. When building the model, several issues need to consider. Firstly this simplified model must accomplish some simple behaviours of vehicles in the process of travellings, such as acceleration and deceleration. Secondly, the model must be stochastic to more closely resemble human behaviour in real-life scenarios. Finally, the model needs to be implemented in code, and the simulated traffic scenarios must be Straightforward and easy to understand. For this reason, we have chosen to use the Nagel-Schreckenberg model, which satisfies the above conditions.

2.2 Details of the Nagel-Schreckenberg model

The Nagel-Schreckenberg model defines a single lane of finite length where vehicles can only travel on the road in a forward and backward position, but not parallel to each other. This one-way lane is divided into many small adjacent areas, each of which is called a " site ". A vehicle moving along the road is constantly moving from one site to the next. Each site on the road can exist in two states, occupied by cars and not occupied by cars (empty).
The vehicle's velocity express in terms of the number of sites passed by a vehicle per unit of time. It ensures that the magnitude of the velocity is always an integer. This is because if the velocity is a decimal number, the vehicle is between two sites at a given unit time. However,

we can solve this problem by splitting the sites of the initial condition more narrowly, so that a decimal representation of velocity is meaningless. To prevent vehicles travelling at an infinite velocity out of all reality, there will be a maximum velocity limit. So the velocity of the vehicles on the road will be an integer between 0 and the maximum velocity.
As mentioned above, there are multiple vehicles travelling simultaneously on the whole road, at different sites and with different velocities. The unit time is defined as a time step, and for each time step, all vehicles will undergo a velocity and position adjustment, which we call a round of updates. The update of the position of the vehicles at a location is determined by the update of the velocity. The update of velocity divides into three cases acceleration, deceleration and random deceleration. The update rules are therefore divided into four steps.

2.2.1 Rules of update

1. *Acceleration*:
   In reality, a vehicle will only choose to accelerate when it is at a safe distance from the car in front of it. In addition, when a vehicle is travelling on an open and straight road, it tends to go faster and faster. Still, laws and regulations limit the maximum velocity (denoted by v_max below) for safety reasons depending on the type of road.The model specifies that the velocity (denoted by v below) increases by 1 over a time step, so when a vehicle updates its v it needs to be sure that the current v is more minor than v_max and that the distance to the vehicle in front is greater than the current velocity plus 1, at which point it can update v to v+1.

2. *Deceleration*:
   In an actual world situation, when a driver notices that the distance to the car in front is too small, they will choose to slow down to prevent a crash. So during the simulation, when the car's distance from the car in front (called d below) is less than or equal to the current speed v, it will slow down to a value where it will not be able to cause a collision with the car in front, at which point it updates v to d - 1.

3. *Randomisation*:
   In reality on the road, vehicles do not drive exactly as they usually think. For example, there are often situations where a vehicle moves slowly despite a considerable distance from the car in front, or where it suddenly brakes sharply. These situations do not follow the usual pattern of accelerating when the distance increases and slowing down when the length decreases, but they are not extremely unlikely accidents. It can be

due to a momentary distraction of the driver or to bad driving habits, so they are widespread. Therefore, such random situations have to be taken into account when modelling traffic conditions. For this situation, the model chooses to design a random probability P, which refers to the probability that the vehicle has P decelerating by 1, which means that v updates to v-1.[6]

4. *Movement*:
   The displacement of the vehicle is determined by the velocity. After the first three steps for the velocity update, the vehicle's position simply moves forward according to the velocity v.

With the vehicle position update completed, the information for a vehicle is updated. When each vehicle in the simulated environment has updated from start to finish, we have concluded a complete round of iterations. We can display the positions and speeds of all the vehicles with a suitable graphical representation. The four step update rule can then repeats to start a new round of iterations. When multiple iterations have completed, it will be possible to show the changes in traffic flow along the entire road.

2.2.2 Conditions of Boundary

Consistent with real life, the lane defined by the model is of finite length, which means that the sites at the lane boundary require some unique settings. The Nagel-Schreckenberg model gives two different conditions, and an open boundary condition and a closed boundary condition. This project focuses on the relationship between traffic density and traffic flow, which is a measure of how many vehicles are on the road, and it can denote as ρ:

$$\rho = \frac{\text{Number of vehicles}}{\text{Length of lane}} = \frac{\text{Number of vehicles}}{\text{Total number of sites}} \tag{2.1}$$

From the above equation we can see that the value of ρ is highly dependent on the number of vehicles. Hence different boundary conditions can have a significant impact on ρ.

1. The open boundary condition:
   For the open boundary condition, it takes two parameters: α and β. α is the probability that a vehicle will enter the road when the first site is empty. β is also a probability value. When a vehicle travels to several sites near the end of the road, and its velocity is greater than the number of remaining sites, it has a probability β that it will leave

this lane. Obviously, due to the open boundary, the number of vehicles entering the lane and the number of vehicles leaving the lane will vary during the iteration, resulting in the total number of vehicles on the whole lane not being a constant. Hence the ρ will also fluctuate under open boundary condition.[4]

2. The closed boundary condition:
   For the closed boundary condition, it is assumed that the roads are connected at the beginning and end. This means that the vehicle will be approaching the first site after passing the last site. In this case, the road is still of finite length, but in theory, the lanes do not have a clear beginning and end, instead it is a closed-loop road. For ease of description we will still use the first site as the beginning of the road and the last site as the end of the road. We can also think of the closed boundary condition as a special case of the open boundary condition, where every time a car leaves the road, a car immediately enters it. In this case, the number of vehicles travelling on the lane will be a constant value. Then the traffic density will remain stable throughout the iterations in the closed boundary condition.[2]

For the convenience of the research, this project will be conducted under closed boundary conditions.

Chapter 3

Code explanation and Simulation

3.1 Explanation of the code

In the simulation we have chosen the data type tuple to store the vehicle information. Each tuple stores two parameters, the speed of the vehicle and the location of the vehicle, which is the site number. We sorted the tuples of all vehicles by position to form a list. This list then contains information on all occupied sites and the vehicles' velocities.

```
Iteration = 30
Size = 100
Cars = 30
P = 20
V_MAX = 5

initial_settings = arrange_sites()
pre_iter = initial_settings
for i in range(Iteration):
    this_iter = list(update_rules(car, pre_iter)for car in pre_iter)
    sites = simulation(pre_iter, this_iter)
    pre_iter = this_iter
```

The five main variables used in the simulation process are *Iteration, Size, Cars, P and V_max.*

- *Iteration*: the total time of the simulation run (each iteration is a time step).
- *Size*: the total number of sites, i.e. the total length of the simulated road.
- *Cars*: the number of vehicles running on the road.

- *P*: the probability that a vehicle's velocity decreases by one during its journey.
- *V_max*: the maximum velocity the vehicle can reach during its trip.

Lines 7 to 8 refer to the use of the *arrange_sites* function to assign a random velocity and starting site to each vehicle in the initial conditions. Afterwards, the list *pre_iter* is assigned to the list *initial_settings* as the initial value list for the first iteration.

Lines 10 to 13 run iterations based on each time step. At each iteration, the velocity and position of each vehicle update according to *update_rules*. As it is a single lane simulation, the situation of overtaking does not exist and therefore, only the distance between vehicles changes while the sequence is constant during the simulation.

```python
def position_sort(sites):
    # Sort vehicles by position
    sorted_sites = sorted(sites, key=lambda site: site[1], reverse = False)
    return sorted_sites
```

The function *position_sort* is to sort the cars according to the positions of the sites they occupy. It is applied at the beginning of the simulation so that it is convenient to subsequently update the velocity and position of each vehicle according to the order.

```python
def check_target_site(sites, target_site):
    # Check if the target site is occupied by a vehicle
    sites_occupied = list(site[1] for site in sites)
    # site is occupied
    if target_site in sites_occupied:
        return False
    # site is not occupied
    return True
```

The function above is designed to check if a station is empty, which means it is not occupied by a vehicle. It takes the location of the target site as a parameter to check if the site exists in the list of occupied sites.

```python
def arrange_sites():
    # Sites is a list representing the road
    sites = []

    # For each car
    for car in range(Cars):

        # Arrange a random speed between 0 and V_MAX
        v = randrange(1, V_MAX)

        while True:
            # Arrange a random site
            target_site = randrange(Size)

            # If the site is empty, arrange the car at this site
            if check_target_site(sites, target_site):
                sites.append((v,target_site))
                break
    return position_sort(sites)
```

The above function is designed to randomly assign speeds and locations to a specified number of vehicles. First, a list called *sites* is created to store a tuple of information about the speed and location of each vehicle. Afterwards, the *randrange* function is used to set a random velocity and location for each vehicle. Also, considering that a site can only be occupied by one car, the *check_target_site* function is used to prevent multiple cars from being assigned to the same site. Further, the velocity and location of each car are added to the sites list as a tuple using the *append* method. Finally, the *position_sort* function is used to sort the cars according to their position.

```python
def distance_to_front(sites, present):
    # Check the distance to the car in front
    front = sites[(sites.index(present) + 1) % len(sites)]
    d = front[1] - present[1]

    # On a circular road, when the car is at the end of the road and the
   car in front is at the beginning of the road,
    # the distance will be less than 0
    if (d < 0):
        d = Size + d

    return d
```

The distance between a vehicle and the vehicle in front can be easily calculated using the *distance_to_front* function. We use the *index* method to find the index of the specified vehicle in the list of *sites*, add one to get the index of the vehicle in front and then get the position information of the vehicle in front. Note that as the road is circular, the car in front of the car at the end of the simulated road is the first car at the beginning of the road. It is, therefore, necessary to use the remainder to determine the exact position of the car in front. Afterwards, the distance between the two cars is calculated from their position information. Similarly, on a circular road, when the car is at the end of the road, and the car in front is at the beginning of the road, So when the distance is less than 0, we need to correct it.

```python
def update_rules(car, sites):

    # Update vehicle velocity
    v = car[0]
    site = car[1]

    # 1. Acceleration
    if v < V_max and target_site_not_occupied(sites, site + v + 1):
        v = v + 1

    # 2. Deceleration
    d = distance_to_front(sites, car)

    # Slow down to distance minus one
    if d <= v:
        v = d - 1


    # 3. Ranomization
    # Vehicles may be randomly slowed down by 1
    if v > 0:
        r = randrange(1,101)
        if r < P:
            v = v - 1

    # Movement
    new_site = site + v

    return v, new_site
```

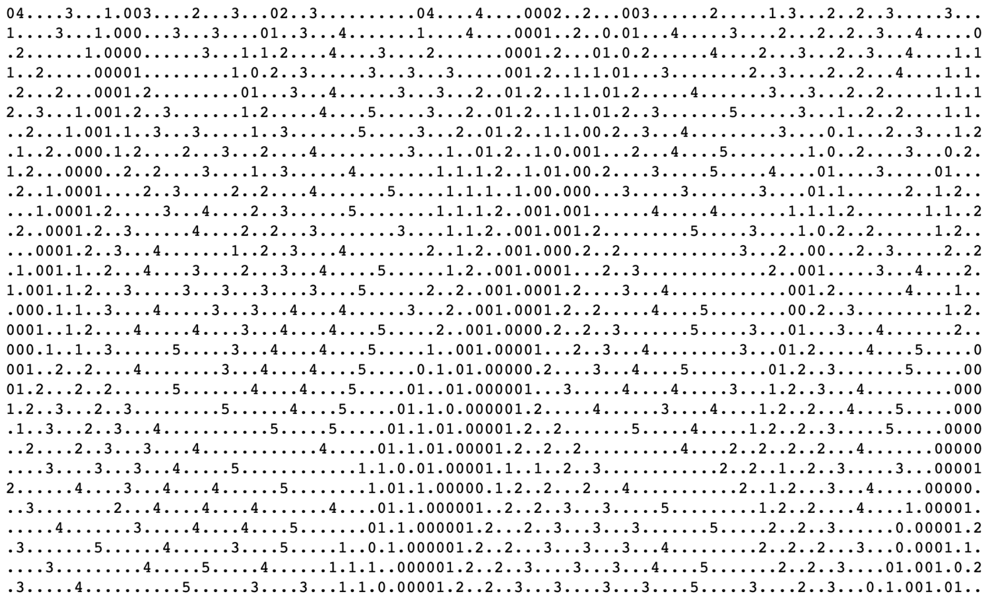
The *update_rules* function updates the velocity and position information for each vehicle according to the four rules described in the previous chapter. Input the vehicle's velocity and position as a tuple, return the new velocity and position as a tuple. Completing this input and output process for all vehicles represents the completion of a time step iteration.

```python
def simulation(pre_iter, this_iter):
    z = zip(pre_iter, this_iter)

    Simulated_sites = ["."]*Size

    for pre_site, this_site in z:
        site = pre_site[1]
#          print(site)

        if site >= Size:
            site = site % Size

        velocity = this_site[0]

        Simulated_sites[site] = str(velocity)

    print("".join(Simulated_sites))
    return Simulated_sites
```

This function displays each iteration as a series of dots (".") and numbers. "." represents an empty site, the number represents that this site is occupied by a car, and the number is the velocity of the car. Entering the previous iteration and the current iteration into the function will output the position of the previous iteration and the velocity of the current iteration. This allows us to simply extrapolate from the result of each iteration the distance the vehicle will move in the next time step. For example, if a site shows the number 4, the vehicle occupying that site will move four sites forward in the next iteration.

3.2 The base simulation

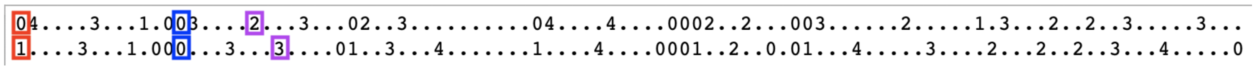
Figure 3.1 shows the results of the execution of the preceding code. In this simulation, the total number of sites is set to 100, the total number of vehicles travelling is set to 30, the probability of a random velocity reduction of one unit of velocity is 20%, and the maximum velocity that can be achieved per unit of time during the vehicle journey is 5.The above base parameter settings follow the original configuration of the Nagel-Schreckenberg model. First, let us explain what the diagram represents according to Figure 3.1. The lines in the figure represent roads, and as the simulation is an endless loop, the beginning and end of



```
04....3...1.003....2...3...02..3..........04....4....0002..2...003......2.....1.3...2..2..3.....3...
1....3...1.000...3...3....01..3...4.......1....4....0001..2..0.01...4.....3....2...2..2..3...4.....0
.2......1.0000......3...1.1.2....4....3....2.......0001.2...01.0.2......4....2...3...2..3...4....1.1
1..2.....00001.........1.0.2..3......3...3...3.....001.2..1.1.01...3........2..3....2..2...4....1.1.
.2...2...0001.2.........01...3...4......3...3...2..01.2..1.1.01.2.....4.......3...3...2..2.....1.1.1
2..3...1.001.2..3.......1.2.....4....5.....3...2..01.2..1.1.01.2..3.......5.....3...1..2..2....1.1.
..2...1.001.1..3...3.....1..3.......5.....3...2..01.2..1.1.00.2..3...4.........3....0.1...2..3...1.2
.1..2..000.1.2....2...3...2....4.........3...1..01.2..1.0.001...2...4....5........1.0..2....3...0.2.
1.2...0000..2..2....3....1..3......4........1.1.1.2..1.01.00.2....3.....5.....4....01....3.....01...
.2..1.0001....2..3.....2..2....4.......5.....1.1.1..1.00.000...3.....3.......3....01.1......2..1.2..
...1.0001.2.....3...4....2..3......5........1.1.1.2..001.001......4.....4.......1.1.1.2.......1.1..2
.2..0001.2..3......4....2..2...3........3....1.1.2..001.001.2.........5.....3....1.0.2..2......1.2..
...0001.2..3...4.......1..2..3....4........2..1.2..001.000.2..2............3...2..00...2..3.....2..2
.1.001.1..2...4....3....2...3...4.....5......1.2..001.0001...2..3.............2..001.....3...4....2.
1.001.1.2...3.....3...3...3....3....5......2..2..001.0001.2....3...4............001.2.......4....1..
.000.1.1..3....4.....3...3...4....4......3...2..001.0001.2..2.....4....5........00.2..3.........1.2.
0001..1.2....4.....4....3...4....4....5.....2..001.0000.2..2..3.......5.....3...01...3...4.......2..
000.1..1..3......5.....3...4....4....5.....1..001.00001...2..3...4.........3...01.2.....4.....5.....0
001..2..2....4........3...4....4....5.....0.1.01.00000.2....3...4....5.........01.2..3.......5.....00
01.2...2..2......5.......4....4....5.....01..01.000001...3.....4....4.....3...1.2..3...4.........000
1.2..3...2..3.........5......4....5.....01.1.0.000001.2.....4......3....4....1.2..2...4....5.....000
.1..3...2..3...4...........5.....5.....01.1.01.00001.2..2.......5.....4.....1.2..2..3.....5.....0000
..2....2..3...3....4............4.....01.1.01.00001.2..2..2..........4....2..2..2..2...4.......00000
....3....3...3...4.....5............1.1.0.01.00001.1..1..2..3............2..2..1..2..3.....3...00001
2......4....3...4....4......5........1.01.1.00000.1.2..2...2...4...........2..1.2...3...4.....00000.
..3........2...4....4....4.......4....01.1.000001..2..2..3...3.....5.........1.2..2....4....1.00001.
.....4.......3.....4....4....5.......01.1.000001.2...2..3...3...3.......5.....2..2..3......0.00001.2
.3.......5......4......3....5.....1..0.1.000001.2..2...3...3...3...4.........2..2..2...3...0.0001.1.
....3.........4.....5.....4......1.1.1..000001.2..2..3....3...3...4....5.......2..2..3....01.001.0.2
.3.....4..........5......3....3...1.1.0.00001.2..2..3...3....3...3....5.....3....2..3...0.1.001.01..
```

Figure 3.1: Results of the code simulation

each line represent two adjacent sites. The vehicle runs from left to right and when it passes the last site it will rejoin the first site to continue its journey. Each line consists of a series of dots and numbers, with each character representing the location of a site. One hundred sites indicate 100 dots and numbers in total. Where a "." means that the stop is empty, and not occupied by a vehicle. The number means that the site is occupied and the number represents the velocity of the vehicle. The speed of each vehicle is set according to the rules, as described in Chapter 2, which stipulate that vehicles accelerate and decelerate. Thus each line represents the current state of traffic on the road.



```
04....3...1.003....2...3...02..3..........04....4....0002..2...003......2.....1.3...2..2..3.....3...
1....3...1.000...3...3....01..3...4.......1....4....0001..2..0.01...4.....3....2...2..2..3...4.....0
```

Figure 3.2: Results of the first two iterations of the simulation

For the next step, let us analyse the path of the vehicles based on the results of the first two lines of iterations in Figure 3.2. First, we consider the first line with the first number 0 (marked by the red box), which indicates that there is currently a vehicle parked at the first stop. Because it is preceded by a vehicle with a distance of 0, it has chosen to stay in place for this iteration. From the second line, we can see that the vehicle in front of it (marked by the red box) has left, so it decides to accelerate forward by 1 in the next iteration. We then look at the vehicle at site 20 in the first line (marked by the purple box), meaning that the

vehicle passes the current site at a speed of 2 and arrives at site 22 after one time step. We notice that on reaching the 22nd site, its velocity increases by 1, and since the vehicle in front of it is more significant than three sites away, it can accelerate. It will therefore complete the next iteration at a speed of 3. Finally, we can observe the vehicle in the middle of the first line, with a velocity of 0 (marked by the blue box). It is the same as the first one because there is a car in front of it and therefore stays in place during this iteration. It is worth noting that in the second row, the vehicle in front of it is 3 sites away, but instead of choosing to accelerate, it continues to stay in place. Firstly, according to the velocity update rules, its speed is less than its maximum speed, and it is much further away from the car in front of it than its speed. It should therefore accelerate by 1. Then, due to the random deceleration setting in the rules, the car's velocity is reduced again by 1 to 0. This, therefore, leads to a confusing situation where the velocity appears to remain the same without updating. This situation simply simulates what often happens in reality, such as a driver not accelerating in time due to a slow reaction, or sudden deceleration due to an unexpected event.
In the end, let us once again consider the entire simulation shown in Figure 3.1, where we can see that it appears that vehicles are at velocity 0 in some areas for several consecutive iterations. This occurs in the first, middle and rear sections of the road, representing three instances of temporary vehicle clustering at these three locations. It is worth noting that during the first seventeen iterations, no congestion occurred at the end of the road. Considering that the vehicles are travelling on a continuous road, it is not difficult to find that the congestion at the back of the road is due to the backward movement of traffic from the first section of the road after several iterations, which in turn affects the flow of vehicles at the back.[3] To summarise, in this simulation, there are two periods of congestion on the entire road. The relationship between traffic flow and traffic density will be analysed in later chapters. In addition, there is an initial period of congestion. There is a large concentration of traffic on some sections and a sparse concentration of traffic on the remaining areas. It can be observed that during this period, most of the vehicles travelling on the sparse section can reach speeds of 4 to 5 and provide more room for acceleration on the congested section. Thus after a while the congestion is gradually reduced, and the distribution of vehicles on the whole stretch of road becomes even, and the speed remains more or less constant at 2 to 3.

Chapter 4

Relationship between traffic flow and traffic density

It is essential that we clarify the definitions of traffic flow and traffic density before the relationship research begins. We define traffic density in 2.1 of the boundary conditions using the number of vehicles and the total number of sites. Traffic flow is primarily intended to be an indicator describing the state of vehicle movement on the road. It can be divided into two primary states, jam and flow. When the road is thoroughly jammed, we can observe that the velocity v of almost all vehicles is 0. When the road is flowing, the vehicles are moving at nearly the same velocity, and the vehicles are evenly distributed along the whole road. Maximum traffic flow is achieved when all vehicles can reach their maximum velocity, and are evenly distributed throughout the road. Of course both of these states often occur on the road at the same time, with some areas of road jammed and some areas flowing. They exist at the same time and change position over time.

Next allow us to consider what happens to the traffic flow under a variety of parameters.

When only 10 cars are travelling on a lane with a length of 100 sites(as in Figure 4.1), it is clear that no traffic jams occur and almost all cars are unaffected by the distance to the car in front. After several iterations, the cars almost always reach a maximum speed of 5. Only random deceleration reduces the speed of the vehicles. This shows that at low densities, traffic conditions are very smooth. This phenomenon is called "undisturbed motion" and is since the number of cars is much smaller than the number of sites. Although the whole road is unobstructed, too many stations are permanently unoccupied, the cars are not evenly distributed, and the traffic flow is low. Increasing the number of cars in this situation will help to increase traffic flow.

```
.........4....2......3....................3...2..5.........1..2......4........3.....................
.............2..3.......4....................2..3.....4.....2...3........5.......4..................
...............3...4........5..................3...4......3...3....4..........5......5..............
..................4....5.........5................4....4.....3...4.....5...........4......5.........
......................5.....5.........5...............4....4....4....5......5..........5.......5....
4..........................4.....4.........5..............4....4....5.....5......5..........5.......
....5..........................5.....5..........5.............4....5.....5.....4......5..........5..
..5......4..........................4.....5..........4............5.....5.....4....5.......4........
.......5.....5..........................5......5.........4.............5.....4....5.....5......5....
5...........5.....5..........................5......5........5..............4....5.....4.....4......
.....4...........4.....5..........................5......5........5.............5.....4....5.....5..
..5......5...........4......4..........................5......5........5.............3....5.....4...
5......4......5..........5......4...........................5......5........5...........4......4....
.....4.....5.......5..........5.....5............................5......5........5..........5......4
...5.....5......5.......5..........5.....4............................5......5........5..........5..
..5.....4.....5......4.......5..........4....5.............................4......5........5........
.......3....5......5.....5........5.........5.....5............................5.......5........4...
4.........3......4......5.....5........4.........5.....5............................5.......4.......
....5........4.......5.......5.....5.......5..........4.....4............................5......5...
.5.......5.......5........5.......5.....5.......5.........5.....4.............................5.....
......5.......4.......5........5.......5.....5.......4.........4....5..............................5
....5......5......5........5........5.......5.....4......5.........5.....5..........................
.........5......5......5........5........5.......4....5.......5.........5.....4.....................
..............5......5......5........4........4......5.....5.......5.........4....5.................
...................5......4......5.......4........5.......4.....5.......5........4.....5............
........................5.....5.......5......4.........5......5......5.......5.......5......5.......
.............................5.....5.......4.....5..........5......5......5.......5.......5......5..
..5...............................5.....5......5......5..........5......5......5.......5.......5....
5......5...............................5.....5......5......5..........5......5......5.......5.......
.....5......5...............................5.....4......5......5..........5......5......4.......5..
```

Figure 4.1: Cars = 10

When the number of vehicles increases to 20(as shown in Figure 4.2), the road starts to

```
.3..........4......01.4.......3.......1..3....3.....1.2...1.01.02..05.............1..3..............
....4...........2..1.1....5......4.....2....4....2...2..2..01.00..00.....5.........1....3...........
........5.........1.1.2........5.....2...2......1..2...2..00.001..00..........5.....2......4........
.............5.....0.2..3...........2..2...3.....2...3...001.01.1.01...............1..3........5....
4.................01...3...4..........2..3....4....3....001.01.0.01.2...............2....3..........
....4.............1.2.....4....5........3...4.....3...1.01.01.01.1.2..3...............2.....3.......
........4..........2..3.......5.....5......4....4....1.01.01.01.1.1..2...4..............3......4....
............5........3...4.........4.....4.....3....0.00.01.01.1.1.2...3.....5.............4.......5
....5............4......4....5.........4.....4....1.1.01.1.00.1.1.2..3....4.......4............5....
5........5...........5......5.....5........5.....1.1.00.1.001..1.2..3...3.....5.......5.............
.....5........5...........5......5.....5........0.1.001..001.2..2..3...3...4.......5.......5........
..........5........5...........4......4.....3...1..001.1.00.2..2..3...3...4....5........5.......5...
.4.............5........5..........4......3....1.1.01.1.001...1..3...2...4....4.....5........5......
.....5..............5........5.........5.....2..0.01.1.001.2...2....2..3.....4....4......5........4.
..5.......5..............5........4.........2..01.1.1.001.2..2...3....3...4......4....4.......5.....
.......4.......5..............5.......5.......01.1.1.001.2..2..3....3....4....5......4....4........5
....5......5........5..............4.......2..1.1.1.000.2..1..3...3....4.....5.....5.....4....5.....
.........5......5........5.............5.....1.1.1.0001...1.2....3...4.....5......5.....4....5.....5
....5.........5......5........5.............0.1.1.0001.2...2..2.....4....5......4......4....5.....5.
...5.....5.........4......5........4........1..1.0000.2..2...2..3.......4.....4.....5......5.....5..
..4.....5.....5........5.......5.......5.....2..00001...1..3...2...4........5.....5......5......4...
4.....4......4.....5........5.......5.......2..00001.2...2....1..3.....5.........5.....5......5.....
....5.....5......4......4........5.......4...00000.1..3...3...1....4.......4.........4.....5.......4
...4.....4.....5.....5......5.........5......000000..1....2...1.2.......4.......5.........5......5..
..3....5.....4......5.....5......5.........1.000001...2.....1..2..3.........5........5.........5....
4....3......3....4.......4.....5......5.....000000.1....3....2...3...4...........5........5.........
....3...3......3.....5.......5......4......0000001..2......3...3....4....5............5........5....
5......3...4......4.......5.......5.....1..000001.2...3.......3...3.....5.....5............5........
.....4....4....5......4........5.......1.1.00001.2..3....4.......2...4.......5.....4............4...
5........4....4.....5.....5.........2...1.00001.2..3...4.....4.....3.....5........3....5............
```

Figure 4.2: Cars = 20

show a section of the road with multiple vehicles in close proximity and zero speed in succession. This indicates that the road is experiencing a traffic jam. Cars can still generally reach speeds of 4 or 5 on non-blocked roads, with large distances between vehicles and smooth traffic flow. On blocked roads cars slow down or even stop in their tracks. However, the traffic does not stop, and we can observe that as the iteration progresses, the position of the congested section gradually moves backwards, a phenomenon known as traffic backward movement. This phenomenon is known as 'traffic backwards' because the fast moving cars in the non-jammed section create space for the cars in the jammed section to move forward, while the cars in front of the jammed section can slowly accelerate until they leave the jammed section. As the cars in front of the blockage require time to accelerate and can only increase their velocity by 1 per time step, the cars coming from behind at high speed can slow down to 0 in one time step, so we can see the traffic in front of the blockage gradually clearing, while the traffic behind continues to increase the traffic congestion and the location of the jam is moving backwards.

As the number of vehicles increases to 70as shown in Figure 4.3, it is clear that most of

```
000002..000001.2..4.....1.00001.0.002..1.1.0000000002..0000002..0.001.0001.01.00001.000.00002..001.0
00001..000001.2..3....1..00001.00.01..1.1.0000000001..0000000..01.01.0001.01.00001.0001.0001..001.00
0001.0.00001.2..3...2..1.0001.001.1.2..1.0000000001.0.0000000..1.01.0001.01.00001.0001.0001.1.01.000
001.01.0001.2..3...2..1.0001.001.1.2..1.0000000000.00.0000001...01.0000.01.00001.0001.0001.1.01.0000
00.01.0001.1..3...2..1.0001.001.1.2..1.00000000001.01.000000.2..1.00001.1.00001.0001.0001.1.01.00000
01.1.0001.1.2....2..1.0000.001.1.1..1.00000000001.01.0000001...1.00001.1.00001.0001.0001.1.01.000000
1.0.0001.0.2..3....1.00001.01.1.1.1..00000000001.01.0000000.2...00000.1.00001.0001.0001.0.01.0000000
.01.001.01...3...2..00001.01.1.1.1.1.0000000001.00.00000001...1.00001..00001.0000.0001.01.0.00000001
01.001.01.2.....1..00001.01.1.0.1.1.0000000001.000.0000001.2...00001.1.0000.00001.000.00.01.0000001.
0.001.01.2..2....0.0001.01.1.00..1.0000000001.0001.000001.2..1.0001.0.00001.0001.0001.01.1.0000000.0
1.01.01.2..2..2..1.001.01.0.001...000000001.0001.000000.2..1.0001.01.0000.0001.0001.01.1.00000001.0
.01.01.2..2..2..1.001.01.01.01.1..00000001.0001.0000000...1.0001.01.00000.001.0000.01.1.0000001.00
.1.01.2..2..2..1.001.00.01.01.0.1.00000001.0001.00000000....0000.00.000001.00.00001.0.1.0000001.001
1.01.1..2..2..0.001.001.1.00.01..00000001.0000.000000001....0001.01.00001.000.0001.01..0000001.000.
.01.1.1...2..01.01.001.1.001.1.1.0000001.00001.0000001.2...001.01.00001.0001.001.01.1.000001.0001.
.1.1.1.2....00.01.001.1.001.0.1.0000001.00001.00000000.2..1.01.00.00001.0001.001.01.1.000001.0001.1
1.1.0.2..2..01.0.001.1.000.01..0000001.00001.000000001...1.01.001.0001.0001.000.01.1.000001.0001.1.
.1.01...2..01.01.01.0.0000.0.0.000001.00001.00000001.2...01.001.0001.0001.0001.1.1.000001.0001.1.0
..01.2....01.01.01.01.0001.1.0.00001.00001.000000000.2..1.1.001.0001.0001.0001.1.1.000001.0001.1.01
1.1.1..2..1.00.01.01.0001.0.00.0000.00001.000000001...1.1.001.0001.0000.0001.1.1.000001.0001.1.00.
.0.1.2...1.001.0.01.0001.00.00.0000.0001.000000001.2...1.001.0000.00001.001.1.0.000001.0001.1.001.
.1..2..1..001.01.1.0001.001.01.0001.000.0000000000.1..2..000.00000.0001.001.0.01.000001.0000.1.001.1
1.2...1.1.00.01.1.0000.001.00.0001.0001.000000001..2...0001.00001.001.001.01.0.000001.00001..001.0.
.1..2..1.001.1.1.00001.01.001.000.0001.000000001.1...1.001.00001.001.001.01.01.00001.00001.1.01.01.
..2...1.000.1.1.00000.01.001.0000.001.000000001.1.2...001.00001.001.001.01.01.00000.00001.1.01.00.2
.1..2..0001..1.000001.0.000.00001.00.000000001.1.2..1.01.00001.001.001.01.01.000001.0001.1.01.001..
..1...0000.2..000001.01.001.0001.001.000000001.0.2..0.01.00001.000.000.00.01.000001.0001.1.01.001.1.
...2..0001...000001.00.001.0001.000.00000001.00...01.0.00001.0000.001.01.1.000001.0001.1.01.000.1.2
.3...0000.2..00001.001.00.0001.0000.0000001.001...1.01.0001.00000.01.01.1.000001.0001.0.01.0001..2.
3...00001...00001.001.000.001.00001.0000000.000.1...01.0001.000001.1.01.0.000001.0001.01.1.0001.2...
```

Figure 4.3: Cars = 70

the road appears to be jammed, with vehicles entering the next jam section almost as soon as they emerge from a jam. On the non-congested sections, the speed of the vehicles stays at 1. One or two vehicles can accelerate to 2 or 3, but only for about one time step before slowing down to 1 again. Under these conditions, the traffic flow on the whole road slows down. From this, we can deduce that as the number of cars gradually increases closer to the number of stops, the traffic flow drops. At this point, the maximum speed limit for the road is no longer meaningful, as the slow traffic flow simply cannot support vehicles travelling at high speeds. It is certainly possible to still observe a large amount of traffic moving backwards, which indicates that the traffic is still moving overall, just very slowly. However, there are many traffic jams and many vehicles are at a standstill.

Based on the above characteristics, we can roughly deduce that when traffic density is at a low level or a high level, the traffic flow is minor. Therefore, as the traffic density increases, the traffic flow should be on an upward and then downward trend. Two different indicators have been chosen to measure traffic flow. One is the number of vehicles passing a particular site per unit time, and the other is the total velocity of all vehicles on the road. The higher the number of vehicles passing a site per unit time, the faster the traffic flow on the road, and the lower the number of vehicles, the slower the traffic flow. The higher the total velocity of all vehicles on the road, the faster the traffic flow, and the lower the total velocity, the slower the traffic flow. We will then investigate the relationship between traffic flow and traffic density using these two statistical indicators separately.

4.1 Number of vehicles passing per unit time

4.1.1 Counting rules

We refer to the site being counted as the count point, located at site i. The number of vehicles passing the count point is denoted as n. First, let's consider how to count the number of vehicles passing the count point per unit of time. We define a vehicle to be deemed to have passed a count point only if it leaves the count point at the next iteration. This means that if a vehicle has a speed of 0 at this count point, it will remain where it is at the next iteration, and we will consider that it has not passed this count point and it will not be counted. With this in mind, we will split the number of cars counted into two cases, when the count point is occupied and when it is not.

1. If the count point is occupied, we divide it into two cases according to the velocity v of the vehicle.
   When v = 0 (as shown in Figure 4.4 below), the vehicle will not leave the count point in the next iteration according to our initial setting and will therefore not be disregarded.

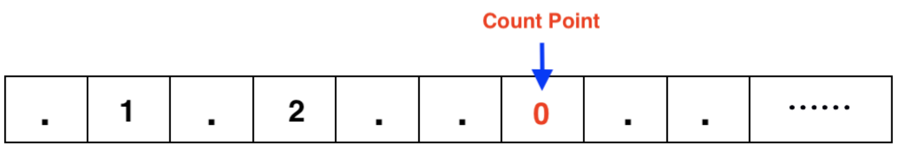


Figure 4.4: The count point is occupied and the car's velocity is 0.

When v>0 (as shown in Figure 4.5 below), the vehicle leaves the count point at velocity 3 in the next iteration, that is, passes the count point. At this point, n $\rightarrow$ n+1.

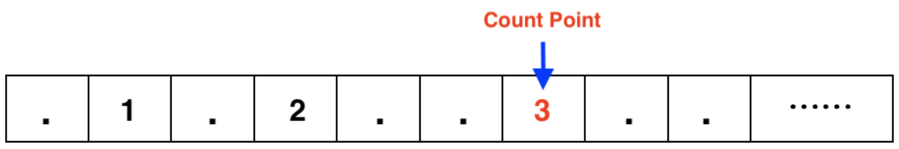


Figure 4.5: The count point is occupied and the car's velocity is greater than 0.

2. If the count point is not occupied now, we need to consider whether the following vehicle will be able to pass the count point on the next iteration.
   As shown in Figure 4.6 below, the first vehicle behind the count point locates at site i-3, which has a velocity of 2, and its distance from the count point is 3. After one iteration it will reach site i-1 and will not pass the count point, so we have no vehicles passing the count point during this iteration.

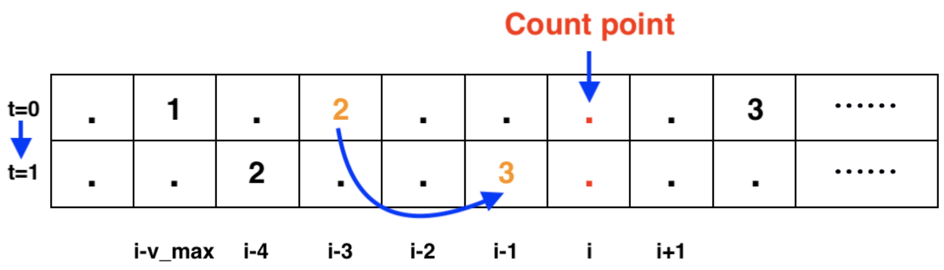


Figure 4.6: The count point is not occupied and the first car behind the count point is at position i-3 with a speed of 2.

Figure 4.7 shows below that the first car behind the count point is at site i-3, and the car has a velocity of 3 at this point and a distance of 3 from the count point. after one iteration it will be at exact site i with a velocity of 3. Due to our initial setup, it does not complete the action of leaving the count point in this round of iterations, so we cannot take it into account.

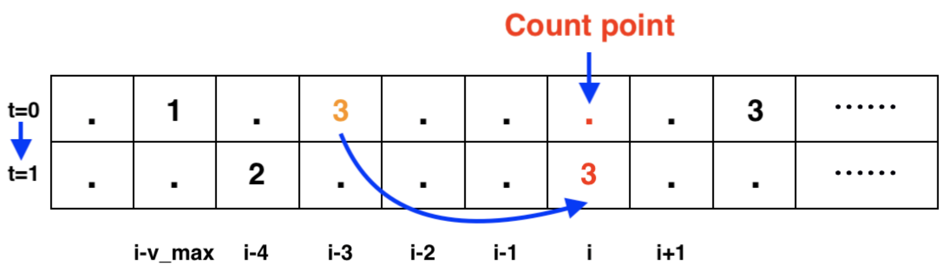


Figure 4.7: The count point is not occupied and the first car behind the count point is at position i-3 with a speed of 3.

According to Figure 4.8, the first vehicle behind the count point is at site i-3 and has a velocity of 4, which is greater than the distance between it and the count point. After one iteration, the vehicle will pass the count point and finally arrive at site i+1 with velocity 3. At this point, we can update n to n+1. It should be noted that after the iteration the orange vehicle's velocity at site i+1 is 3 instead of 4 or 5. This is because the vehicle in front of it had a velocity of 3 before the iteration and should be located precisely four sites away from it after the iteration. So to prevent a crash in the next iteration, the orange vehicle chooses to reduce its velocity to 3.0 after reaching site i+1.

In summary, in the event that a count point is not occupied, the only way it can

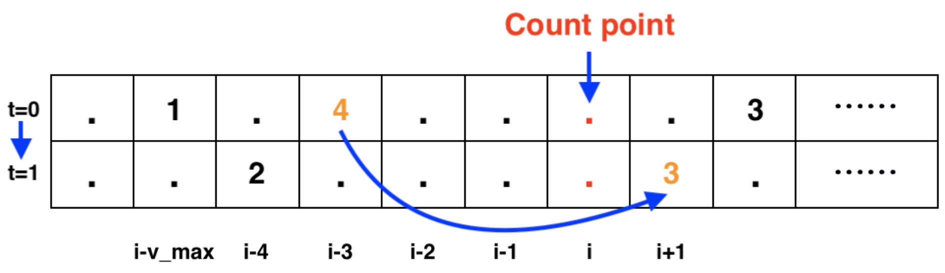


Figure 4.8: The count point is not occupied and the first car behind the count point is at position i-3 with a speed of 4.

be counted as having passed the count is if the velocity of the first vehicle behind the count point is greater than the distance between it and the count point. Also, because of the maximum velocity limit for vehicles, in this project, the top speed is 5. It is only necessary to look backwards up to 4 sites to determine if a vehicle may have passed the count point. A vehicle located more than 4 sites behind a count point is not considered. Because its velocity must be less than or equal to 5, it must not pass the count point.

4.1.2 Code and simulation results

```
................................................................................................-1..................
...............................................................................................-1...................
..............................................................................................-1....................
.............................................................................................-1.....................
............................................................................................-1......................
...........................................................................................-1.......................
..........................................................................................-1........................
.........................................................................................-1.........................
........................................................................................-1..........................
.......................................................................................-1...........................
......................................................................................-1............................
.....................................................................................-1.............................
....................................................................................-1..............................
...................................................................................-1...............................
..................................................................................-1................................
.................................................................................-1.................................
................................................................................-1..................................
...............................................................................-1...................................
..............................................................................-1....................................
.............................................................................-1.....................................
............................................................................-1......................................
...........................................................................-1.......................................
```

Figure 4.9: Simulation results when the number of Cars is 1

In both simulations in this section and the next, we have not included simulations where the number of vehicles is 0 and 1. The absence of cars on the road is not very realistic for our project study; we cannot study what happens to traffic flow on an empty road, and there is no doubt that the absence of cars means that there is no traffic flow. We exclude the case of only one car because of my model itself. The deceleration part of our code in the update rule function requires that the distance d between the vehicle and the car in front of it be calculated. That v be reduced to d-1 when d is less than or equal to the vehicle speed v. Thus, when there is only one vehicle on the entire road, due to our closed boundary condition, the car in front of this vehicle is itself, d will always be equal to 0 and v will always be d-1 = -1. This results in the simulation shown below, where each The vehicle will go backwards at -1 for each iteration. For these reasons, we exclude the cases where the number of vehicles is 0 and 1 from the simulation.

```python
Cars_list = list(range(2,Size))
density_list=[]
flow_list=[]
count_point_1 = randrange(0,Size)
count_point_2 = randrange(0,Size)
```

Based on the original model, to be able to count the number of cars passing a given site per unit time (below called n) and to study its relationship with the traffic density (ρ), we need to add some parameters to the code to improve the calculation and recording of the data. Firstly, as we want to simulate the traffic flow at different traffic densities, we have to change the fixed-parameter Cars to a list *Car_list* from 2 to 99. Furthermore, two empty lists, *density_list* and *flow_list*, are created to record the different values of ρ and their corresponding n during the simulation. Finally, to obtain universal results, we used the *randrange* function to randomly select two count points, hoping to increase the accuracy by calculating their average.

```
1  for Cars in Cars_list:
2      initial_settings = arrange_sites_randomly()
3      pre_iter = initial_settings
4      counts_1 = 0
5      counts_2 = 0
6      for i in range(Iteration):
7          this_iter = list(update_rules(car, pre_iter)for car in pre_iter)
8  
9          sites = simulation(pre_iter, this_iter)
10 
11         counts_1 += pass_or_not(count_point_1, sites)
12         counts_2 += pass_or_not(count_point_2, sites)
13 
14         pre_iter = this_iter
15     density = compute_density(Cars, Size)
16     flow = (counts_1+counts_2)/(Iteration*2)
17     print(density, flow)
18     density_list.append(density)
19     flow_list.append(flow)
```

The code above takes 30 iterations of a road with different traffic densities and records the corresponding data. Thus a total of $98 \times 30 = 2940$ simulations were performed. Two variables are created on lines 4 and 5 to record how many vehicles passed through each of the two observation points during the iteration. Lines 12 to 14 use the *pass_or_not* function to calculate whether a car passes the count point at each iteration and record the results in *counts_1* and *counts_2*. After each traffic density simulation has completed 30 iterations, we first calculate the traffic density using the *compute_density* function. The number of vehicles passing through the two counts after this iteration is then summed and divided by the number of iterations to obtain the total number of vehicles passing through the two points per unit time, and we divide this number by two to obtain the number of vehicles passing through a given count point per unit time. At the end we record these two values in the *density_list* and *flow_list* using the *append* method.

```python
def compute_density(cars, size):
    # Calculation of traffic density
    return cars/size
```

compute_density is a convenience function for calculating traffic density.

```python
def pass_or_not(count_point, sites):
    # If there is no vehicle at the count point
    if sites[count_point] is str("."):
        for i in list(range(count_point - V_MAX, count_point))[::-1]:
            if sites[i] is not str(".") and int(sites[i]) > (count_point -
    i):
                return 1
        return 0

    # If there is a vehicle at the count point and the velocity is greater
    than 0.
    elif int(sites[count_point]) > 0:
        return 1
    # If there is a vehicle at the count point and the velocity equals 0.
    else:
        return 0
```

pass_or_not is a function used to count whether a car has passed a count point at each iteration according to the counting rules in the previous section. Due to the initial setup and update rules of our model, there is no overtaking of cars, so at most one car passes by each station at each iteration. We input the location of the count point and the result of the road, outputting 1 when a car passes the count point and 0 when it does not.

Based on our code, a dot plot Figure 4.10 was drawn to see the effect of traffic density on the number of vehicles passing a count point per unit time. As the independent variable traffic density (ρ) increases, the number of cars on the road gradually increases. The dependent variable the number of vehicles passing a given site per unit time (n), first gradually increases from less than 0.1 to nearly 0.6. As ρ approaches 20%, the maximum amount of traffic the road can handle reaches and n begins to fall. Until ρ approaches 100% and n is almost wholly zero, meaning that the entire road is wholly congested with cars and all

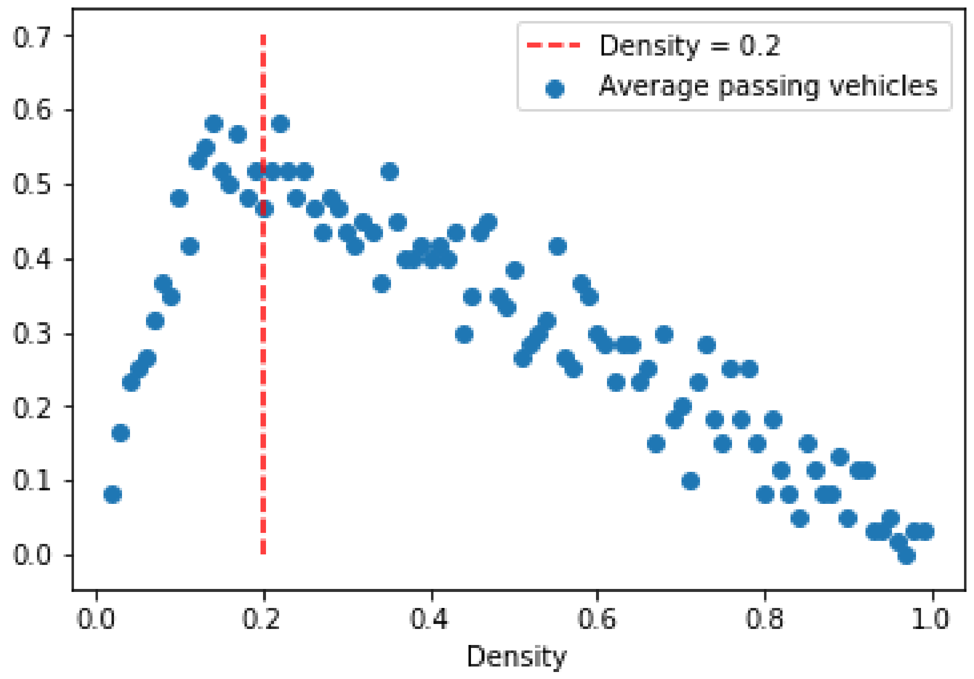


Figure 4.10: Relationship between the number of vehicles passing a count point per unit time and traffic density.

vehicles stop where they are.

4.2 total velocity of all vehicles on the road

4.2.1 Counting rules

It is much easier to calculate the total velocity than calculating the number of vehicles passing a station per unit time. We are actually calculating an average total velocity, obtained by adding up the total velocity after each iteration and dividing it by the number of iterations. Considering that our model initially arranges the position and speed for the vehicle randomly, the average total speed is used to exclude its influence on the vehicle operation. The formula 4.1 is shown below, where M denotes the total number of iterations and C denotes the number of vehicles.$v_{m,n}$ denotes the speed of the vehicle. For example $v_{2,3}$ denotes the speed of the third vehicle after the second iteration.

$$\text{Total velocity} = \frac{1}{m}\sum_{m}^{M}\sum_{c}^{C} v_{m,n} \tag{4.1}$$

In addition, the average speed of vehicles at different traffic densities was calculated to see if the average speed of vehicles changed as the traffic density increased. The formulae are given below.

$$\text{Average velocity} = \frac{1}{c}\frac{1}{m}\sum_{m}^{M}\sum_{c}^{C} v_{m,n} \tag{4.2}$$

4.2.2 Code and simulation results

```
1  Cars_list = list(range(2,Size))
2  density_list = []
3  velocity_list = []
4  average_velocity_list = []
```

Like the previous code, we create two lists, *density_list* and *velocity_list*, to the original code to record the traffic density and the total velocity of the vehicles for each round of the simulation, respectively. In addition to this, a list *average_velocity_list* has been added to record the average velocity of the vehicles.

```
1  for Cars in Cars_list:
2      initial_settings = arrange_sites_randomly()
3      pre_iter = initial_settings
4      velocity = 0
5      for i in range(Iteration):
6          this_iter = list(update_rules(car, pre_iter)for car in pre_iter)
7          sites = simulation(pre_iter, this_iter)
8  
9          velocity += compute_total_vehicle_speed(sites)
10         pre_iter = this_iter
11     density = compute_density(Cars, Size)
12     total_velocity_per_timestep = velocity/Iteration
13     average_velocity = total_velocity_per_timestep/Cars
14     print(density,"",total_velocity_per_timestep)
15     density_list.append(density)
16     velocity_list.append(total_velocity_per_timestep)
17     average_velocity_list.append(average_velocity)
```

In the fourth line, we create a variable *velocity* to record the sum of all speeds throughout the simulation. The iterative process of running the simulation remains the same as the base model, where in the ninth line, we use the *compute_total_vehicle_speed* function to calculate the total speed of the vehicle on the road after each iteration and add it to the parameter *velocity*. After each traffic density iteration, we divide *velocity* by the number of iterations to obtain the total speed of all vehicles on the road and store it in the *velocity_list*. In addition, we divide the total velocity by the number of vehicles to obtain the average velocity at the current density and record it in *average_velocity_list*. We also calculated the traffic density again using the *compute_density* function.

```python
def compute_total_vehicle_speed(sites):
    # Calculate the sum of the velocities of the vehicles on the entire
    road at each iteration
    v = 0
    for i in sites:
        if i is not str("."):
            v += int(i)
    return v
```

The *compute_total_vehicle_speed* formula allows us to use the results of the simulation to calculate the speed of all vehicles on the road after each round of iteration.

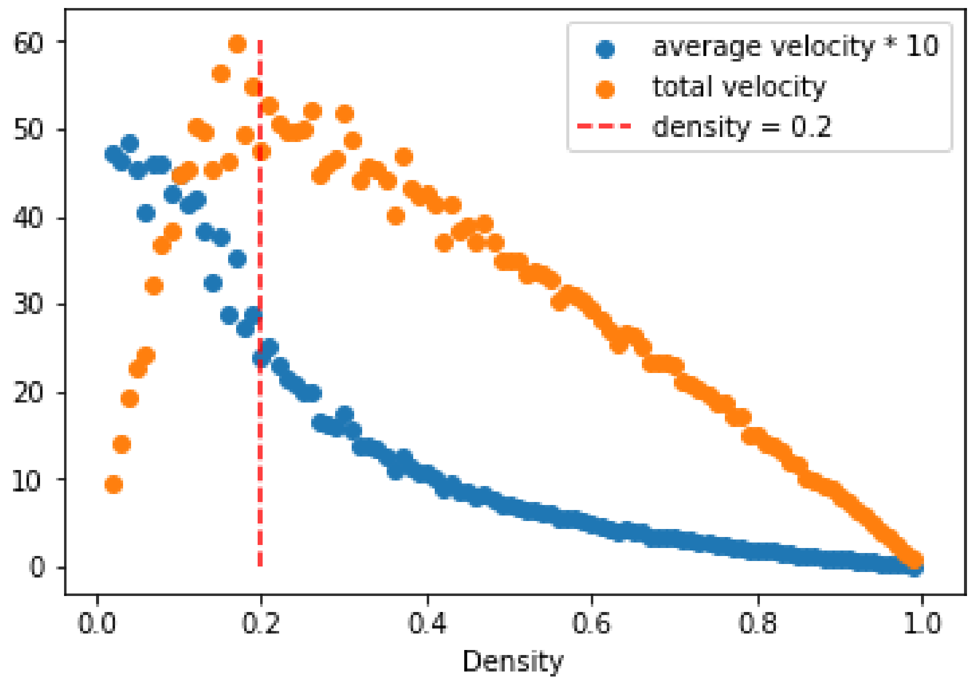
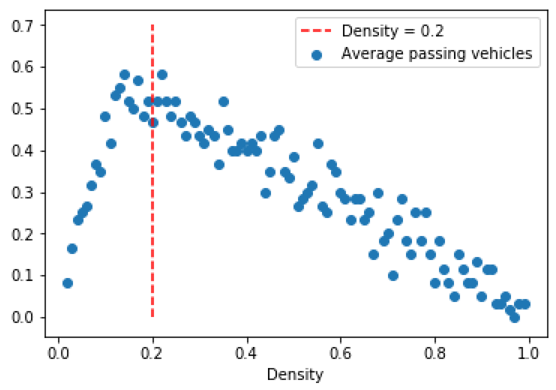


Figure 4.11: average velocity V.S. traffic density and total velocity V.S. traffic density.

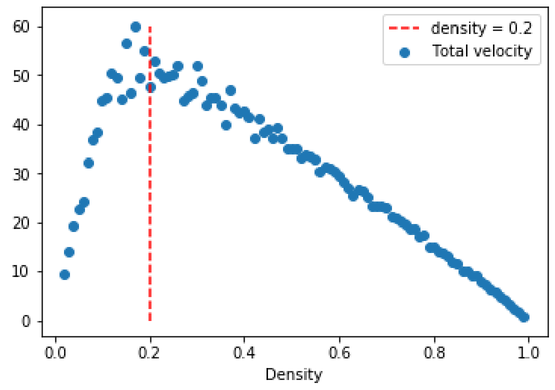
Figure 4.11 shows the average speed multiplied by 10 together with the overall speed data. We can see that the trend for the orange points is basically the same as the trend we discussed in the previous section, both rising and then falling with the increase in density. This indicates that the overall velocity peaks when the traffic flow also peaks, after which the overall velocity starts to decrease as the traffic density increases and the congested roadway begins to increase, and the traffic flow begins to decrease. At the same time, we can find that the average velocity of cars decreases as the traffic density increases, while the rate of decrease does not remain stable. The average speed decreases slowly until the traffic volume peaks. After the traffic volume peaks, the average speed decreases rapidly as the traffic density increases and eventually settles down again. While acceleration is a slow process in the model, with each time step only adding up to 1, deceleration is a speedy process. In the event of a traffic jam, vehicles coming from behind can reduce their existing speed directly to 0 within a single time step. The rapid decline phase is caused by the increase in traffic after the traffic flow reaches its maximum, and the rise in traffic jams caused by the rise in the number of vehicles on the road, with the original high-speed traffic flow triggered by one. The presence of a traffic jam triggers the speed of several vehicles to drop directly to 0, thus rapidly pulling down the average value. As the traffic density increases further, the traffic congestion in the road increases and the distance between two congested sections becomes shorter. In this case, the vehicle has just left a blockage. It will not be able to accelerate to high speed before it encounters another blockage, so the volume of traffic on non-blocked roads is at a low speed, and, therefore, the average speed decreases gradually.

4.3 Simulation results and reality

Based on the results of the previous two simulations, we know that as the traffic density increases, the traffic flow goes through increasing and then decreasing. Looking at Figures 4.12(a) and 4.12(b) we see that the traffic flow peaks when the traffic density approaches 20%. These are due to the fact that the total number of points is 100 and the maximum speed of the vehicle is 5. When the condition of random speed reduction is ignored, the distance between two vehicles is 6. The rear vehicle can continue to drive at high speed without slowing down. This means that when all vehicles on the road have a distance of 6, all vehicles can reach a maximum speed of 5 and be evenly distributed across the lane. This just about satisfies the maximum traffic flow on the road. We can find that 100 divided by 6 (the stops occupied by vehicles plus the 5 stops ahead) equals approximately 16.67%, which means that theoretically, the traffic flow reaches its maximum when the traffic density is at



(a) Relationship between the number of vehicles passing a count point per unit time and traffic density.



(b) Relationship between total velocity and traffic density.

Figure 4.12:

16.67%. We can summarize the following equation to calculate the optimal traffic density,

$$\rho^* = \frac{\frac{\text{Number of sites}}{(v_{max}+1)}}{\text{Number of sites}} = \frac{1}{(v_{max}+1)} \times 100\% \tag{4.3}$$

Allow us then to take the random slow-down situation into account. When a vehicle randomly decelerates by 1. It closes the distance between it and the car behind it, reducing the overall average speed, it leaves more room to allow vehicles to enter the flow of traffic. Assuming that all vehicles on the road have reduced their velocity by 1, in other words, the maximum vehicle speed is 5 - 1 = 4, then according to equation 4.3 the maximum flow on the road will occur at a traffic density of 1/(5-1+1) = 20%. Of course, it is not possible for all vehicles to slow down at random, so the maximum flow on the whole road should be between 16.67% and 20%. The results of the simulations in Figures 4.12(a) and 4.12(b) show that the traffic density at which the maximum traffic flow reached is about 18%, which is right in this interval.

Figure 4.13 shows a scatter plot of the statistics obtained for real traffic conditions. When comparing the results of the simulation in Figure 3.1 with Figure 4.13, we will notice that the scatter distribution is highly similar in shape and position. In real traffic the volume of traffic rises sharply to a peak as the traffic density increases from 5% to 19% and then begins to fall. On the whole, traffic always reaches a peak and then drops, so when traffic flow on the road is low it can be caused by two opposite reasons. One is that the vehicle density is too low, with too few vehicles passing through per unit time, which leads to low

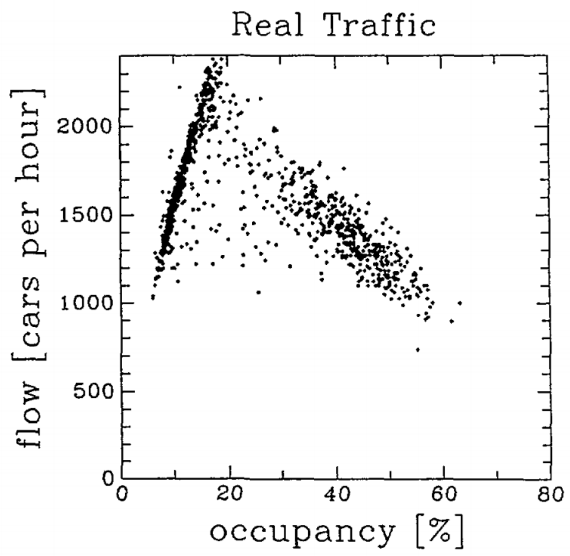


Figure 4.13: Real traffic Fundamental diagram (Taken from “A cellular automaton model for freeway traffic”)

traffic flow. The second is that the vehicle density is too high. The more cars on the road, the more likely it is that a traffic jam will form. When the vehicle density is higher than the optimal density, traffic jams will turn out to be more and more on the road. The vehicles on the blocked sections are stalled, and as the number of blocked sections increases, the speed of the vehicles on the non-blocked sections is gradually reduced. This leads to a reduction in the overall speed of vehicles and a slow forward movement of vehicles, resulting in low traffic flow.[1]

Chapter 5

Conclusion

In conclusion, a traffic simulation model with closed boundary conditions based on the Nagel-Schreckenberg model has been built using python. The random package was used for the random deceleration of the vehicles, and the matplotlib package was used for the visualisation of the collected data. The use of python code allows us to modify and debug variables to simulate different traffic conditions. Although the effects of all the variables are not analysed individually, the use of the code allows us to simplify the debugging process and to visualise the results of the simulation. The following conclusions are drawn from the analysis of the simulation results.

First, we visualise the road changes visualised by the code, and we can see that when the road is congested, there is a phenomenon of traffic backward movement. This is because the vehicles at the front of the congested road need time to slowly accelerate away, while the vehicles at the back of the road, travelling at high speed, slow down to zero and line up at the end of the queue for a short period of time. This causes the position occupied by the congested vehicles to change, gradually moving backwards. Secondly, we chose to use two indicators, the number of vehicles passing a given site per unit of time and the total velocity, to measure the volume of traffic flow. When comparing these two indicators with traffic density, we observed the same results. Both indicators rise rapidly to a peak with increasing traffic density and then fall to 0. The traffic density at which the indicator reaches its peak is the optimal traffic density we want to obtain. At the optimum traffic density, the road will get its maximum traffic flow.

Finally, we compare the simulation results with the real traffic, the trends are generally consistent and we find that the optimal traffic density is related to the maximum velocity allowed to travel and the random deceleration. When there is no random speed reduction, the optimal traffic density is inversely proportional to the maximum velocity plus one. When the random speed reduction is taken into account, the traffic density where the maximum flow occurs shifts to the right and becomes larger.

Bibliography

[1] Michael Blank. Hysteresis phenomenon in deterministic traffic flows. *Journal of Statistical Physics*, 120(3/4):627 – 658, 2005.

[2] Tsang-Jung Chang, Kao-Hua Chang, and Hong-Ming Kao. A new approach to model weakly nonhydrostatic shallow water flows in open channels with smoothed particle hydrodynamics. *Journal of Hydrology*, 519:1010–1019, 11 2014.

[3] Xin-Gang Li, Zi-You Gao, Bin Jia, and Rui Jiang. Deceleration in advance in the nagel–schreckenberg traffic flow model. *Physica A: Statistical Mechanics and its Applications*, 388(10):2051 – 2060, 2009.

[4] Yinyi Ma, Jan van Dalen, Chris de Blois, and Leo Kroon. Estimation of dynamic traffic densities for official statistics: Combined use of data from global positioning system and loop detectors. *Transportation Research Record*, 2256(1):104–111, 2011.

[5] Kai Nagel and Michael Schreckenberg. A cellular automaton model for freeway traffic. *Journal de physique I*, 2(12):2221–2229, 1992.

[6] H. B. ZHU, H. X. GE, and S. Q. DAI. A new cellular automaton model for traffic flow with different probability for drivers. *International Journal of Modern Physics C: Computational Physics Physical Computation*, 18(5):773 – 782, 2007.

Appendix A

Project Code

```python
from random import randrange
import matplotlib.pyplot as plt

def position_sort(sites):
    # Sort vehicles by position
    sorted_sites = sorted(sites, key=lambda site: site[1], reverse = False
   )
    return sorted_sites

def check_target_site(sites, target_site):
    # Check if the target site is occupied by a vehicle
    sites_occupied = list(site[1] for site in sites)
    # site is occupied
    if target_site in sites_occupied:
        return False
    # site is not occupied
    return True

def arrange_sites():
    # Sites is a list representing the road
    sites = []

    # For each car
    for car in range(Cars):

        # Arrange a random speed between 0 and V_MAX
        v = randrange(1, V_MAX)

        while True:
            # Arrange a random site
            target_site = randrange(Size)

            # If the site is empty, arrange the car at this site
            if check_target_site(sites, target_site):
                sites.append((v,target_site))
                break
    return position_sort(sites)

def distance_to_front(sites, present):
    # Check the distance to the car in front
    front = sites[(sites.index(present) + 1) % len(sites)]
    d = front[1] - present[1]

    # On a circular road, when the car is at the end of the road and the car in front is at the beginning of the road,
    # the distance will be less than 0
    if (d < 0):
        d = Size + d

    return d

def update_rules(car, sites):

    # Update vehicle velocity
    v = car[0]
    site = car[1]

    # 1. Acceleration
    # When the speed of the vehicle is less than V_MAX and
    # the position after the move is not currently occupied by a vehicle
    if v < V_MAX and check_target_site(sites, site + v + 1):
        v = v + 1

    # 2. Deceleration
    # When the distance to the vehicle in front is less than the speed of the vehicle,
    # the vehicle needs to slow down.
    d = distance_to_front(sites, car)

    # Slow down to distance minus one
    if d <= v:
        v = d - 1

    # 3. Ranomization
    # Vehicles may be randomly slowed down by 1
    if v > 0:
        p = randrange(1,101)
        if p <= P:
            v = v - 1

    # 4. Movement
    new_site = site + v

    return v, new_site

def simulation(pre_iter, this_iter):
    z = zip(pre_iter, this_iter)

    Simulated_sites = ["."]*Size

    for pre_site, this_site in z:
        site = pre_site[1]
#          print(site)

        if site >= Size:
            site = site % Size

        velocity = this_site[0]

        Simulated_sites[site] = str(velocity)

    print("".join(Simulated_sites))
    return Simulated_sites

def compute_density(cars, size):
    # Calculation of traffic density
    return cars/size

def pass_or_not(count_point, sites):

    # If there is no vehicle at the count point
    if sites[count_point] is str("."):
        for i in list(range(count_point - V_MAX, count_point))[::-1]:

            if sites[i] is not str(".") and int(sites[i]) > (count_point -
    i):

                return 1
        return 0
    # If there is a vehicle at the count point and the velocity is greater
    than 0.
    elif int(sites[count_point]) > 0:
        return 1
    # If there is a vehicle at the count point and the velocity equals 0.
    else:
        return 0

def compute_total_vehicle_speed(sites):
    # Calculate the sum of the velocities of the vehicles on the entire
   road at each iteration
    v = 0
    for i in sites:
        if i is not str("."):
            v += int(i)
    return v

Iteration = 30
Size = 100
Cars = 30
P = 20
V_MAX = 5

initial_settings = arrange_sites()
pre_iter = initial_settings
for i in range(Iteration):
    this_iter = list(update_rules(car, pre_iter)for car in pre_iter)
    sites = simulation(pre_iter, this_iter)
    pre_iter = this_iter

# Simulation of the relationship between the number of cars passing a
   particular station per unit of time and density.
```

```python
Cars_list = list(range(2,Size))
density_list=[]
flow_list=[]

count_point_1 = randrange(0,Size)
count_point_2 = randrange(0,Size)
for Cars in Cars_list:
    initial_settings = arrange_sites()
    pre_iter = initial_settings
    counts_1 = 0
    counts_2 = 0
    for i in range(Iteration):
        this_iter = list(update_rules(car, pre_iter)for car in pre_iter)

        sites = simulation(pre_iter, this_iter)

        counts_1 += pass_or_not(count_point_1, sites)
        counts_2 += pass_or_not(count_point_2, sites)

        pre_iter = this_iter
    density = compute_density(Cars, Size)
    flow = (counts_1+counts_2)/(Iteration*2)
    print(density, flow)
    density_list.append(density)
    flow_list.append(flow)

plt.xlabel("Density")
plt.vlines([0.2], 0,0.7, linestyles='dashed', colors='red',label = '
    Density = 0.2')
plt.scatter(density_list, flow_list, label = 'Average passing vehicles')
plt.legend()
plt.show;

# Simulation of the relationship between the average velocity of a vehicle
     on the road and density.
Cars_list = list(range(2,Size))
density_list = []
velocity_list = []
average_velocity_list = []

for Cars in Cars_list:
    initial_settings = arrange_sites()
    pre_iter = initial_settings
    velocity = 0
    for i in range(Iteration):
        this_iter = list(update_rules(car, pre_iter)for car in pre_iter)
        sites = simulation(pre_iter, this_iter)

        velocity += compute_total_vehicle_speed(sites)
        pre_iter = this_iter
    density = compute_density(Cars, Size)
    total_velocity_per_timestep = velocity/Iteration
    average_velocity = total_velocity_per_timestep/Cars
    print(density,"",total_velocity_per_timestep)
    density_list.append(density)
    velocity_list.append(total_velocity_per_timestep)
    average_velocity_list.append(average_velocity)

plt.xlabel("Density")
plt.vlines([0.2], 0,60, linestyles='dashed', colors='red',label = 'density
    = 0.2')
plt.scatter(density_list, velocity_list, label="Total velocity")
plt.legend()
plt.show();

plt.xlabel("Density")
plt.scatter(density_list, list(i*10 for i in average_velocity_list),label
    = 'average velocity * 10')
plt.scatter(density_list, velocity_list, label = 'total velocity')
plt.vlines([0.2], 0,60, linestyles='dashed', colors='red',label = 'density
    = 0.2')
plt.legend()
plt.show;

# Special cases: Cars = 1
Cars = 1
initial_settings = arrange_sites()
pre_iter = initial_settings
for i in range(Iteration):
    this_iter = list(update_rules(car, pre_iter)for car in pre_iter)
    sites = simulation(pre_iter, this_iter)
    pre_iter = this_iter
```