Traffic jams (the Nagel-Schreckenberg model)

Student: Yasin Butt-Shah 160226966

Supervisor: Dr. Wolfram Just

Contents

<u>Chapter 1: An overview of the Nagel-Schreckenberg model</u>

1.1Introduction

This research project is based on the Nagel-Schreckenberg model. Specifically, the aim of this project is to reproduce and discuss the results relating to the backwards movement of traffic. In this project I will produce code to simulate that of realistic traffic and discuss details as to how traffic builds up. I will do this with discussion based on simulations produced from my written code, as well as reference to the Nagel-Schreckenberg original research, in particular the fundamental diagram. We will discuss what causes traffic jams to move backwards and discuss what causes the "undisturbed motion"[1] of vehicles as well. The original model to simulate traffic jams was done using a boolean model built with Fortran, using arrays, similarly I have implemented code using arrays and a mixture of various functions and parameters using Python 3. The nature of the simulation involves using a set number of vehicles in a closed loop, this loop acts as a circular road with one lane motion. Vehicles in this road effectively hop to segments of the road known as sites. Using the original Nagel-Schreckenberg model I have defined a list of rules which all vehicles must abide by. These rules include when a car should accelerate and when a car should slow down. Very important to this simulation is the idea of randomness in order to emulate human behaviour. Without an element of randomness, the very nature of the model will behave differently, and the automation of the model will not emulate human behaviour. Finally, for this model to work all elements of acceleration, slowing down and randomisation will be performed in parallel before what is known as a time-step. After each time-step is carried out, the vehicles will move and thus a simulation of real-life traffic will be achieved. Also included in this project is an in-depth explanation of the programming which has been used. Specifically, all variables, functions and logic will be explained.

1.2 Summarisation of results

Within this project, simulations were done in two different ways. Firstly, a base case, which followed the same criteria as that of the original Nagel-Schreckenberg model and secondly the base case with one parameter changed. In all but one cases we can see a traffic jam building, and naturally what follows is the backward movement of traffic which is one of the fundamental results from the Nagel-Schreckenberg model. Some simulations when run on repeat do not produce traffic jams, but most of the outputs do.

Summarising the results, we see that most noticeable is the backwards movement of traffic when traffic jams occur. This specifically occurs when density has increased beyond what the road can effectively handle. We see this from the fundamental diagram, where peak flow is achieved at a certain density and anything beyond begins to reduce flow. It is at this reduction of flow where traffic builds up. Results also show one key principle in relation to undisturbed motion. Fundamentally little to no traffic occurs when either the length of the road is increased, or the number of vehicles deceased. We have shown this using half the number of cars than in the base case, cars flow at high velocities without encountering traffic. This result is crucial to apply to real life. We notice that in reality it is these same principles which can prevent or build traffic.

[1] Kai Nagel, Michael Schreckenberg. A cellular automaton model for freeway traffic. Journal de Physique I, EDP Sciences, 1992, 2 (12), pp.2221-2229. ff10.1051/jp1:1992277ff. ffjpa-00246697

Chapter 2: Details of the Nagel-Schreckenberg model

2.1 The model

The aim of our model is to set up a scenario which can simulate real world traffic. The very nature of this model must be simplified to emulate in code, but also realistic enough, that the results of tests produced by my code can be applied to real life. The real-world traffic emulated, will demonstrate the backward movement of traffic as we will see from results produced by my written code.

The model used is the Nagel-Schreckenberg model is defined to be on a "one-dimensional array of L sites."[2] Imagining a narrow infinite road, split up into many segments, the one-dimensional aspect of the model describes how vehicles cannot be next to each other, rather they can only be in front or behind one another. The many segments that would divide this 'infinite road' are known as 'sites' regarding our model. The nature of the model is that at any one given time, a site can have one of two different states. These states are that the site is occupied by a vehicle or the site is empty.

For every given vehicle which occupies a site, there has been assigned an "integer velocity with values between zero and v_{max}"[3] Integer velocities are for two reasons, for simplicity within the model and for an easily adjustable increase of velocity which is consistent. Say for instance that the velocity were not integer values. This would mean that when we increase a vehicles velocity by 0.5, the vehicle would need to move 0.5 sites in the next time-step. This would be problematic since we would then need to divide sites into further sub-sites. Also, it is important to understand in our simulation, velocity therefore represents the number of sites the car will advance in the next time-step. The maximum velocity for most simulations will be 5.

I have mentioned above the phrase "time-step," and I shall explain the importance of this. The very nature of this model will call for four different variables to be in constant progress during the simulation. These four different rules so to speak, will be explained in the next paragraph, but importantly it must be known that these variables change velocities in steps. We do not have a set number of seconds before velocities change and vehicles move, rather they simulate that of frames. Each 'frame' updates these variables in parallel, this means after each variable has adjusted simultaneously, the vehicles move accordingly. This is what we describe as a time-step.

2.2 Rules of the model

1) Acceleration: They are two conditions which tell the vehicle to advance its velocity. These two conditions are the current velocity of the vehicle and the distance between it and the next vehicle. (Distance in this case is referred to as the number of sites between the two vehicles, specifically in a forward motion) If the vehicles current velocity(v) is less than 5, and the distance between the next vehicle is larger than v+1, the speed is increased by one. Therefore, we can write that [$v \rightarrow v+1$].
2) Slowing down: With reference to sites (*i* and *j*) the following rules dictate the slowing down of a vehicle. If a vehicle at site *i* sees the next vehicle at site $i+j$ (with $j \leq v$) it reduces its speed to $j-1$and we can write that as [$v \rightarrow j-1$]. To explain this further, *j* represents the number of sites between the vehicle in discussion and the next vehicle. The model states that if the number of sites between the vehicle in discussion and the vehicle in front is smaller than the velocity, then of course in the next time-step, the vehicle will crash. To avoid this, the velocity becomes one less than the number of sites between the vehicle in discussion and the vehicle in front of it.
3) Randomisation: By a random probability *p* the velocity of each vehicle (if greater than zero) is decreased by one, and we can write that as [$v \rightarrow v-1$]. This variable within the code is purely

[2] *Ibid.*

[3] Kai Nagel, Michael Schreckenberg. A cellular automaton model for freeway traffic. Journal de Physique I, EDP Sciences, 1992, 2 (12), pp.2221-2229. ff10.1051/jp1:1992277ff. ffjpa-00246697

to simulate real world scenarios. Its purpose is to assure that the results are not robotic, rather include some element of human nature, hence the random probability will combat this issue. For the purposes of most of this study, I have selected that p be set to 0.2.

4) Car motion: Each vehicle is advanced by v sites. The purpose of this variable is to conclude all the adjustments that need to be made to acceleration, slowing down and velocities and implement apply them, therefore moving onto the next time-step.

Let us now discuss the 'closed system nature' of the model. The simulations run using the code I have written is based on a closed system, meaning there is not an infinite road, rather a closed loop upon which vehicles will move. This satisfies periodic boundary conditions. We can simulate scenarios with varying parameters up to very large numbers this way. Furthermore, the closed loop allows for traffic to build up in a realistic way, like that of "car races but only on a single lane."[4] If I were to use a straight infinite road, the frontmost car would never slow down since our model dictates slowing down depends on the car in front. Therefore, my simulation would be unsuccessful in showing traffic build up, and unsuccessful in showing the phenomena of traffic jams slowly moving backwards.

[4] *Ibid*.

Chapter 3: Explanation of the code

Before we explain the code it is important to note the data structures used to represent a vehicle and the site. A vehicle is simply represented as a tuple(pair) whereby we store the velocity as the first element and the position as the second element. The position in this case, refers to how far along the road the vehicle has travelled. The site is represented as a list of tuples whereby each tuple is a vehicle. We only store data for occupied sites not empty sites.

```
 1 ITERATIONS = 22
 2 SIZE = 100
 3 CARS = 20
 4 PROBABILITY = 20
 5 VMAX = 5
 6
 7
 8 starting = populate_sites_randomly()
 9 previous_iteration = starting
10
11 for _ in range(ITERATIONS):
12     this_iteration = sort_on_position(list(map(lambda car: rules(car, previous_iteration), previous_iteration)))
13     display(previous_iteration, this_iteration)
14     previous_iteration = this_iteration
```

We start of by defining global variables on lines 1-5. These variables are:

- Iterations-How many time-steps the simulation runs for
- Size-The number of sites (size is used since “sites” has been used for another purpose)
- Cars(vehicles)-The number of vehicles
- Probability-The probability of a vehicle reducing its velocity by 1
- V_{max}-The maximum velocity of a vehicle at any given time-step

Lines 8-9 define the starting position of each vehicle and velocity at a site.

Lines 11-14 runs each of the iterations. By using pythons map functions, we apply the “rule” function to each vehicle in the previous iteration and save this as the current iteration. After performing this calculation, we call the “display” function to print the time-step.

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

    # For each car
    for i in range(CARS):

        # Give this car a random velocity
        v = randrange(1, VMAX)

        while True:
            # Give this car a random position
            position = randrange(SIZE)

            # If position is not occupied, add car to the road
            if position_not_occupied(sites, position):
                sites.append((v, position))
                break

    return sort_on_position(sites)
```

The above functions purpose is to populate the number of assigned cars randomly to sites. This function considers whether a position is occupied before assigning a car to the site. This is to prevent multiple cars being placed on the same site. This function is invoked only once to create the starting positions. A few python standard library functions are used, these are “append” and “randrange,” they add to the end of a list and give a random number between two integers respectively. Finally, we sort based on the position.

```python
def position_not_occupied(sites, position):
    # Checks if a position on the road is occupied by a car
    for site in sites:
        if site[1] == position:
            return False
    return True
```

This function is a convenience function that simply checks whether there is a vehicle at a site, using the position as a parameter to check.

```python
def sort_on_position(sites):
    # Takes unsorted sites and sorts based on the position
    return sorted(sites, key=lambda site: site[1])
```

This function is a convenience function that simply sorts the vehicles on the site based on position.

```python
def rules(car, sites):
    v = car[0]
    i = car[1]

    # 1. Acceleration
    if v < VMAX and position_not_occupied(sites, v + 1):
        v = v + 1

    # 2. Slowing down
    j = distance_to_next(sites, car)
    if j <= v:
        v = j - 1

    # 3. Randomization
    if v > 0:
        r = randrange(1, 100)
        if r <= PROBABILITY:
            v = v - 1

    # 4. Car motion
    i = i + v

    return v, i
```

This function takes a particular vehicle and applies the four set of rules. We only manipulate the vehicles velocity and position, and we return this new velocity and position as a tuple that represents the movement of the vehicle in this iteration.

```python
def distance_to_next(sites, current_site):
    next_site = sites[(sites.index(current_site) + 1) % len(sites)]
    distance = next_site[1] - current_site[1]
    if (distance < 0):
        distance = SIZE - abs(distance)
    return distance
```

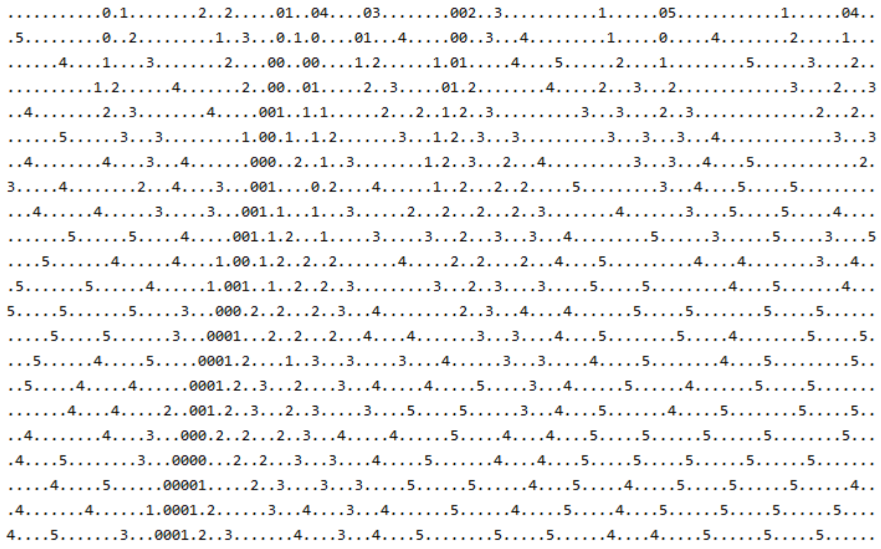
This function simply calculates the number of empty sites between a current vehicle and the next in a forward direction. The next vehicle ahead is obtained by using the current vehicles index in the sites list and then taking the remainder of dividing this by the length of sites. We must take the remainder to account for the cyclic nature of the road.

```python
def display(previous_iteration, this_iteration):
    zipped = zip(previous_iteration, this_iteration)

    sites_to_display = ["."] * SIZE

    for previous_site, next_site in zipped:
        position = previous_site[1]

        if position >= SIZE:
            position = position % SIZE

        velocity = next_site[0]

        sites_to_display[position] = str(velocity)

    print("".join(sites_to_display))
```

This function displays each iteration as a series of dots and numbers representing an empty site or as vehicle occupying that site with velocity “v” respectively. This function needs to merge the previous iteration and the current iteration together so that we can display the velocity of a vehicle from the next iteration in the position of the previous iteration. This is so that on a given iteration if the velocity says 3, then the vehicle will move 3 sites right by the next time-step.

Chapter 4: Simulations

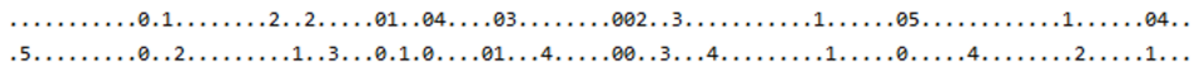
4.1 The base simulation



```
...........0.1........2..2.....01..04....03........002..3...........1......05............1......04..
.5.........0..2.........1..3...0.1.0....01...4.....00..3...4.........1.....0.....4........2.....1...
......4...1...3........2...00..00....1.2......1.01.....4....5......2....1.........5......3....2..
..........1.2......4.......2..00..01.....2..3.....01.2........4.....2...3...2.............3....2...3
..4........2..3........4.....001..1.1......2...2..1.2..3..........3...3....2..3..............2...2..
......5......3...3.........1.00.1..1.2.......3...1.2..3...3..........3...3...3...4............3...3
..4........4....3...4.......000..2..1..3........1.2..3...2...4..........3...3...4....5...........2.
3.....4........2...4....3...001....0.2....4......1..2...2..2.....5.........3...4....5.....5........
...4......4......3.....3...001.1...1...3......2...2...2...2..3........4.......3....5.....5.....4....
.......5......5.....4.....001.1.2...1.....3.....3...2...3...3...4.........5......3......5.....3....5
....5.......4......4....1.00.1.2..2..2.......4.....2..2....2...4....5..........4....4........3...4..
.5.......5......4......1.001..1..2..2..3.........3...2..3....3.....5.....5.........4....5.......4...
5.....5.......5.....3...000.2..2...2..3...4..........2..3...4....4.......5.....5........5.....5......
.....5.....5.......3...0001...2..2...2...4....4.......3...3....4....5.......5.....4........5.....5.
...5......4.....5.....0001.2....1..3...3.....3....4......3...3.....4.....5........4....5.........5..
..5.....4.....4......0001.2..3...2....3...4.....4.....5.....3...4......5......4.......5.....5.......
.......4....4.....2..001.2..3...2..3.....3....5.....5......3...4....5.......4.....5.........5.....5..
..4........4....3...000.2..2...2..3...4.....4......5.....4....4....5.....5......5......5........5...
.4....5.........3...0000...2..2...3...3....4.....5.......4....4....5.....5.....5......5......5.......
.....4.....5......00001.....2..3....3...3.....5......5......4....5.....4.....5.....5......5......4..
.4.......4......1.0001.2......3...4....3...4.......5......4.....5.....4....5......5.....5......5....
4....5.......3...0001.2..3.......4....3...4....5........5.....5......4....4.....5......5.....5......
```

Figure 1 – Base simulation code

Figure 1 represents output from my written code. We begin by explaining the format of what is being produced. Each line represents the road, in this case we emulate that of a closed loop by wrapping the end of the line with the beginning of the next line. Each new line represents a new iteration, effectively it is the next time-step. The road as previously mentioned consists of sites. These sites are either occupied by a vehicle or empty. Empty sites are displayed as a '.' and sites occupied by a vehicle are displayed with their corresponding velocity. We should note that each vehicle obeys the rules set out by the model. Theses rules include when to accelerate and when to slow down. For this specific base case, we have set the number of iterations to 22, the number of sites to 100, the number of vehicles/cars to 20, the probability of a car randomly reducing its speed by 1, to 20% and the maximum velocity of a vehicle to 5. These specific base parameters follow that of the original output produced by the Nagel-Schreckenberg model.



```
...........0.1........2..2.....01..04....03........002..3...........1......05............1......04..
.5.........0..2.........1..3...0.1.0....01...4.....00..3...4.........1.....0.....4........2.....1...
```

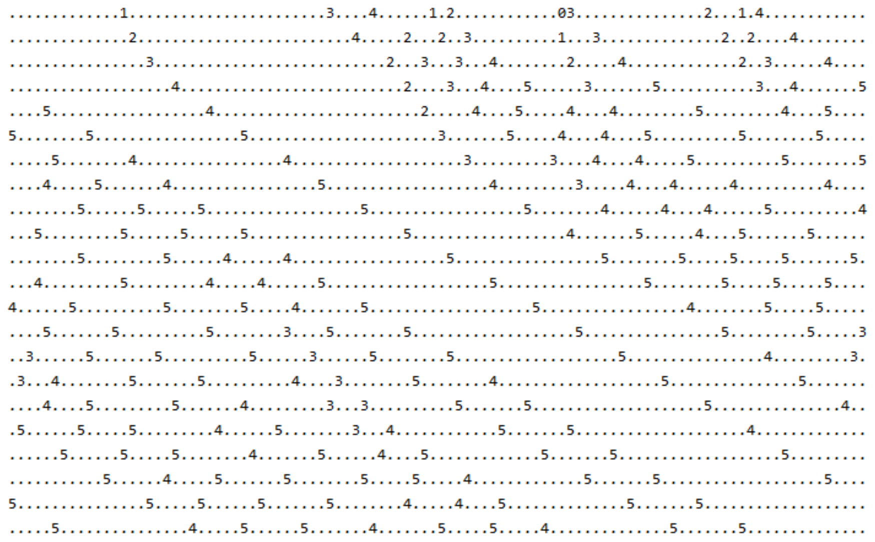
Figure 2 – First two iterations from base code

In figure 2 we analyse the first two lines of output. The first 0 on the very first line represents a vehicle at current velocity 0. We notice there is a vehicle in front, and so this vehicle decides not to increase its speed. On the next line we notice the very same vehicle has still not moved. It is important to note that the rule for accelerating relies on current velocity as well as the number of free sites ahead. On the second line we see that since two spaces are free which is larger than 0+1 (current velocity+1), this car should be ready to accelerate. However, there is the element of human nature which has been accounted for with a probability. This probability randomly reduces the speed of a vehicle by 1 to account for human nature, this could include a late reaction time or becoming distracted while stopped.

We notice very clearly where traffic begins to build up. This is displayed by clusters of 0's which represents vehicles that have stopped. It is very clear to see the backward movement of traffic as we go from one iteration to the next. The very purpose of this project is to display such a phenomenon. In the early iterations we see most vehicles to the right of the traffic build with speeds between 0-3. As we proceed to later iterations, we notice that most vehicles to the right of the traffic build up having speeds of 2-5. This shows us that since more cars are becoming jammed, there are more sites on the road which are empty, and so obeying the rules of acceleration, the vehicles increase speed.

4.2 Simulations with varying parameters

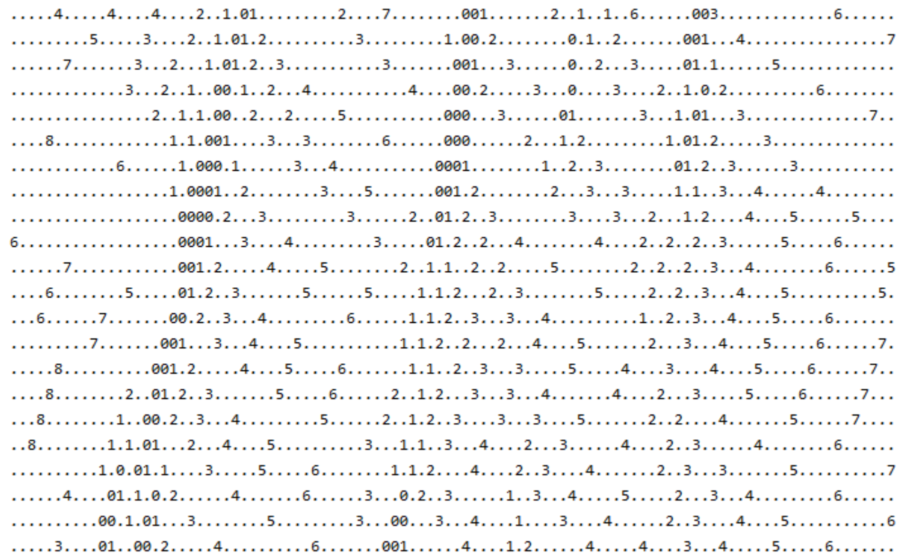
Within this section we discuss the results of simulations from varying parameters. These variable parameters include the number of vehicles, number of sites and the maximum velocity of a vehicle.



```
.............1.......................3....4......1.2............03...............2...1.4............
..............2.........................4.....2...2..3..........1...3..............2..2....4........
................3...........................2...3...3...4........2.....4.............2..3......4....
...................4..........................2....3...4....5......3.......5...........3...4.......5
....5..................4........................2.....4....5.....4....4.........5.........4....5....
5........5.................5......................3.......5.....4....4....5..........5........5.....
.....5........4.................4....................3.........3....4....4.....5..........5........5
....4.....5.......4.................5...................4.........3.....4....4......4..........4....
........5......5......5..................5..................5........4......4....4......5..........4
...5.........5......5......5..................5..................4.......5......4....5.......5......
........5.........5......4......4..................5.................5........5.....5.....5.......5.
...4.........5.........4.....4......5...................5.................5........5.....5.....5....
4......5..........5........5.....4.......5...................5.................4........5.....5.....
....5.......5..........5........3....5........5...................5................5.........5.....3
..3......5.......5..........5......3......5........5...................5................4.........3.
.3...4........5.......5..........4....3........5........4...................5...............5.......
....4....5.........5.......4.........3...3..........5.......5....................5...............4..
.5......5.....5.........4......5........3...4............5.......5....................4.............
......5......5.....5........4.......5......4....5.............5.......5...................5.........
...........5......4.....5.......5........5.....5.....4.............5.......5...................5....
5...............5.....5......5.......5........4.....4....5..............5.......5...................
.....5...............4.....5......5.......4.......5.....5.....4..............5.......5..............
```

Figure 3 – Simulation with vehicles reduced to 10 (Undisturbed motion)

This simulation in figure 3 was done with half the number of vehicles occupying the road at any one time (10) As we can see there is no traffic build up, and not any backward movement of traffic. Within the Nagel-Schreckenberg model, this is described as "undisturbed motion."[5] This is due to low a low overall density. With reference to the fundamental diagram, when the number of cars is less than the number of sites, the flow of vehicles has not yet reached its maximum state. In this case since we have halved the number of vehicles, we have reduced the overall density thus reducing the flow.

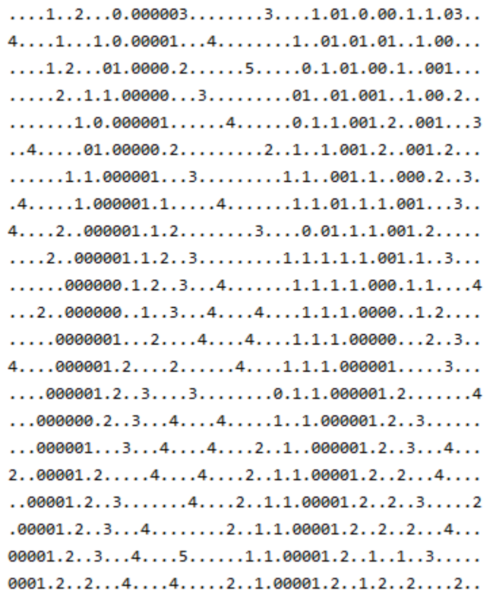


```
.....4.....4....4....2..1.01.........2....7........001.......2..1..1..6......003.............6......
.........5.....3....2..1.01.2..........3.........1.00.2........0.1..2.......001...4................7
......7.......3...2...1.01.2..3...........3.......001...3......0..2...3.....01.1......5.............
.............3...2..1..00.1..2...4...........4....00.2.....3...0....3....2..1.0.2..........6........
................2..1.1.00..2...2.....5...........000...3......01.......3...1.01...3..............7..
....8.............1.1.001....3...3........6......000......2...1.2.........1.01.2.....3..............
............6......1.000.1......3...4...........0001........1..2..3........01.2..3......3...........
..................1.0001..2........3....5.......001.2........2...3...3.....1.1..3...4......4........
...................0000.2...3.........3......2..01.2..3........3....3...2...1.2....4....5......5....
6..................0001...3....4.........3.....01.2..2...4........4....2..2..2..3......5.....6......
......7............001.2.....4.....5........2..1.1..2..2.....5........2..2..2..3...4........6......5
....6........5.....01.2..3.......5......5.....1.1.2...2..3........5.....2..2..3...4....5..........5.
...6......7.......00.2..3...4.........6......1.1.2..3...3...4..........1..2..3...4....5.....6.......
.........7.......001...3...4....5...........1.1.2..2...2...4....5.......2...3...4....5.....6......7.
.....8..........001.2.....4....5.....6.......1.1..2..3...3.....5.....4....3....4....5.....6......7..
....8........2..01.2..3.......5.....6......2..1.2...3...3...4.......4....2...3.....5.....6......7...
...8........1..00.2..3...4.........5......2..1.2..3....3...3....5.......2..2....4.......5......7....
..8........1.1.01...2...4....5..........3...1.1..3...4....2...3......4....2..3......4........6......
..........1.0.01.1....3.....5.....6........1.1.2....4....2..3....4.......2..3...3.......5..........7
......4....01.1.0.2......4.......6......3...0.2..3......1..3...4.....5.....2...3...4........6.......
..........00.1.01...3........5.........3...00...3...4....1....3....4......2..3....4....5...........6
.....3....01..00.2.....4..........6.......001......4....1.2......4.....4....3...4.....5.....6.......
```

Figure 4 – Simulation with maximum velocity 10

[5] Kai Nagel, Michael Schreckenberg. A cellular automaton model for freeway traffic. Journal de Physique I, EDP Sciences, 1992, 2 (12), pp.2221-2229. ff10.1051/jp1:1992277ff. ffjpa-00246697

In figure 4 we see the results from a simulation where the maximum velocity of vehicles has been increased to 10 and the number of vehicles remains at 20. We notice a few things. Firstly, the velocity of 10 is never achieved, this is because the road is simply not large enough to allow any vehicle to accelerate this much, the vehicle will encounter another vehicle ahead before it has time to increase its speed to10. Secondly we notice what seems to be 2 different traffic jams building up, although the second is less noticeable. The first traffic jam also evidently shows the backward movement of traffic. Secondly we note although traffic jams don’t frequently build up, vehicles are much closer together with lower speeds compared to the base simulation. This is due to vehicles catching up to the next a lot faster than in the base simulation.



```
....1..2...0.000003........3....1.01.0.00.1.1.03..
4....1...1.0.00001...4........1..01.01.01..1.00...
....1.2...01.0000.2......5.....0.1.01.00.1..001...
.....2..1.1.00000...3.........01..01.001..1.00.2..
.......1.0.000001......4......0.1.1.001.2..001...3
..4.....01.00000.2.........2..1..1.001.2..001.2...
......1.1.000001...3.........1.1..001.1..000.2..3.
.4.....1.000001.1.....4.......1.1.01.1.1.001...3..
4....2..000001.1.2........3....0.01.1.1.001.2.....
....2..000001.1.2..3.........1.1.1.1.1.001.1..3...
......000000.1.2..3...4.......1.1.1.1.000.1.1....4
...2..000000..1..3...4....4....1.1.1.0000..1.2....
.....0000001...2....4....4....1.1.1.00000...2..3..
4....000001.2....2......4....1.1.1.000001.....3...
....000001.2..3....3........0.1.1.000001.2.......4
...000000.2..3...4....4.....1..1.000001.2..3......
...000001...3...4....4....2..1..000001.2..3...4...
2..00001.2.....4....4....2..1.1.00001.2..2...4....
..00001.2..3.......4....2..1.1.00001.2..2..3.....2
.00001.2..3...4........2..1.1.00001.2..2..2...4...
00001.2..3...4....5......1.1.00001.2..1..1..3.....
0001.2..2...4....4.....2..1.00001.2..1.2..2....2..
```

Figure 5 – Simulation with sites reduced to 50

In figure 5 we see the results from a simulation where the only varied parameter is the number of sites, which has been reduced from 100 to 50. We see not only are there two traffic jams, but both traffic jams display a backward movement. The traffic jams when compared to all other cases also have more vehicles stationary. This due to the size of the road being reduced by half, since there is less sites on there road, there is less space for cars to move. This shows the length of the road can impact the probability of a traffic jam building.

Chapter 5: The fundamental diagram

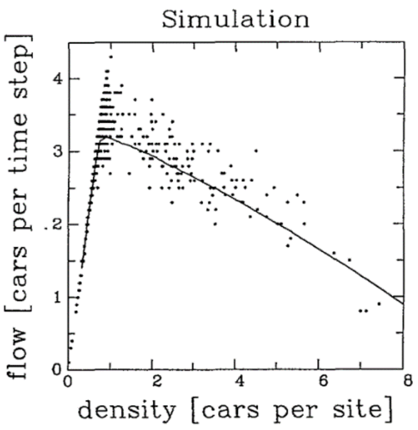


Figure 6 – Simulated Fundamental diagram (Taken from "A cellular automaton model for freeway traffic")

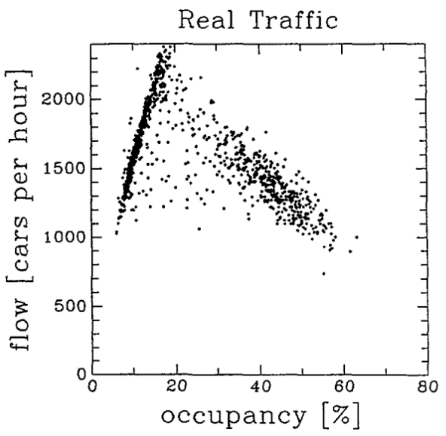


Figure 7 – Real traffic Fundamental diagram (Taken from "A cellular automaton model for freeway traffic")

Following simulating the backward movement of traffic, one can also obtain what is known as the fundamental diagram as shown in figure 6. First we must understand the two variables used in this plot. The first is the independent variable density, which in the model is measured by cars per site. To accurately plot density, we must take the number of cars used in the simulation and divide it by the number of sites in our simulation. Therefore, we determine a result measured by cars per site. The next variable is the dependent variable flow, measured in cars per time step. This variable is dependent on the density and measures the number of cars flowing through a site within one time-step.

We should note a few observations when studying this plot. First, we note that when density is 0 the flow is 0. This is because if there are 0 cars per site, then there will be 0 cars per time step, i.e. there will no cars passing through a site at any given time. Secondly we notice that as density increases, the flow increases. There is a direct positive correlation between the two variables up to approximately a density of 1. This is to say that when density (Number of cars divided by number of sites) remains 1, we are at peak flow. The reasoning for this is that when density is 1, the number of cars is equal to the number of sites and so maximum flow is achieved as there is precisely enough sites for each car to constantly move and not have to slow down or build up traffic. Thirdly we note the decline in flow as the density exceeds 1. The reduction of flow is not as prominent as the initial increase in flow, rather it is a steady decline. The result is a negative correlation between flow and density when density increases beyond 1.

We note that "the line represents averages over 10^6 time-steps."[6] This explains that the more iterations of this simulation we consider the more the data begins to resemble the line. When we compare the simulation to figure 7 (which represents real traffic), we notice they resemble similar shapes. A sharp increase in flow between approximately 8% and 19% occupancy, and maximum flow is achieved when the occupancy of the roads increases to approximately 19%. Beyond 19% occupancy the flow begins to decrease. Summarising we notice flow is optimum when density rests at a specific density, both in real traffic and in the simulation. The decrease in flow is representative of traffic jams occurring, i.e. the more cars occupying a road after optimum flow, the more traffic is likely to build up. Finally, it is important to note in both simulation and real traffic, flow does not reach 0, indicating that although traffic builds up, the vehicles in both cases eventually move and pick up speed.

[6] Kai Nagel, Michael Schreckenberg. A cellular automaton model for freeway traffic. Journal de Physique I, EDP Sciences, 1992, 2 (12), pp.2221-2229. ff10.1051/jp1:1992277ff. ffjpa-00246697

Chapter 6: Conclusion

To conclude this project, I have successfully reproduced simulations in Python 3 to that of the simulations within the Nagel-Schreckenberg model. We conclude with some facts and analytics we have discovered throughout this project.

Firstly, within traffic jams, indeed a phenomenon occurs whereby traffic jams themselves move backwards. This is evident in our simulations and also representative of reality. In motorways, traffic builds up after faster from behind than vehicles accelerating in front due to human nature (an example would be reflex time, although multiple real natured variables affect this). I have emulated this delayed responsiveness by allocating a randomness to the movement of vehicles within the code.

Secondly we have analysed and discussed the plot of the fundamental diagram. This plot was produced within the Nagel-Schreckenberg model, and we have discussed its usefulness in explaining how density can affect the likelihood of traffic jams. Most notably within the fundamental diagram, the flow of cars reaches their peak at a specific density. In reality, this specific density can be measured by the number of cars occupying the road. With this information we can conclude that traffic jams depend on the ratio between the number of cars and the space available on the road.

Finally, the compilation of code within Python has allowed me to modify and adjust parameters to simulate many scenarios. Although not all possible scenarios have been observed within this project, one could easy adjust variables independent of one another and analyse simulations produced. This has most definitely been the advantage of using modern programming and has allowed for fast compilation and run times through the python interpreter. Using Python 3 I have been able to run simulations with all rules in parallel, and the final code can be analysed within the final chapter.

Chapter 7: References

1) Kai Nagel, Michael Schreckenberg. A cellular automaton model for freeway traffic. Journal de Physique I, EDP Sciences, 1992, 2 (12), pp.2221-2229. ff10.1051/jp1:1992277ff. ffjpa-00246697

Chapter 8: Appendix

8.1 The complete code

```python
from random import randrange


def sort_on_position(sites):
    # Takes unsorted sites and sorts based on the position
    return sorted(sites, key=lambda site: site[1])


def position_not_occupied(sites, position):
    # Checks if a position on the road is occupied by a car
    for site in sites:
        if site[1] == position:
            return False
    return True


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

    # For each car
    for i in range(CARS):

        # Give this car a random velocity
        v = randrange(1, VMAX)

        while True:
            # Give this car a random position
            position = randrange(SIZE)

            # If position is not occupied, add car to the road
            if position_not_occupied(sites, position):
                sites.append((v, position))
                break

    return sort_on_position(sites)


def distance_to_next(sites, current_site):
    next_site = sites[(sites.index(current_site) + 1) % len(sites)]
    distance = next_site[1] - current_site[1]
    if (distance < 0):
        distance = SIZE - abs(distance)
    return distance


def rules(car, sites):
    v = car[0]
    i = car[1]

    # 1. Acceleration
    if v < VMAX and position_not_occupied(sites, v + 1):
        v = v + 1

    # 2. Slowing down
    j = distance_to_next(sites, car)
    if j <= v:
        v = j - 1

    # 3. Randomization
    if v > 0:
        r = randrange(1, 100)
        if r <= PROBABILITY:
            v = v - 1

    # 4. Car motion
    i = i + v

return v, i



def display(previous_iteration, this_iteration):
    zipped = zip(previous_iteration, this_iteration)

    sites_to_display = ["."] * SIZE

    for previous_site, next_site in zipped:
        position = previous_site[1]

        if position >= SIZE:
            position = position % SIZE

        velocity = next_site[0]

        sites_to_display[position] = str(velocity)

    print("".join(sites_to_display))


ITERATIONS = 22
SIZE = 100
CARS = 20
PROBABILITY = 20
VMAX = 5


starting = populate_sites_randomly()
previous_iteration = starting

for _ in range(ITERATIONS):
    this_iteration = sort_on_position(list(map(lambda car: rules(car,
previous_iteration), previous_iteration)))
    display(previous_iteration, this_iteration)
    previous_iteration = this_iteration
```