Traffic Simulation at a Signalized Four-Way Intersection

Sandesh Adhikary | Felice Chan | Scott Sims
(sadhikary6 | fchan9 | ssims36)

CSE 6010: Assignment 5

December 5, 2017

Traffic Simulation at a Signalized Four-Way Intersection

Sandesh Adhikary | Felice Chan | Scott Sims
(sadhikary6 | fchan9 | ssims36)

December 5, 2017

1. Introduction

The goal of our project was to simulate vehicle traffic at a four-way intersection. In order to implement this, we adopted the Nagel-Schreckenberg (NS) model discrete model for single lane freeway traffic. In the original model, time, space and state variables, are discrete, and the computation model is defined on a one-dimensional array of length *L*, where each cell in the array can be occupied by one vehicle or it can be empty. This implies that each cell on the grid is defined as the length of one car and the minimum space between one car and the car in front or behind it, and that it is a zero collision model. Each vehicle has an integer velocity value between 0 and v_{max}. There are four basic principles that govern car movement in the NS model, described below.

1. Acceleration: If the velocity of a vehicle is lower than v_{max}, and the distance to the car ahead is larger than v+1, then the speed is incremented by one.
2. Braking: If there is another vehicle in the way of the current vehicle at site *j*, then the car will reduce its speed to *j*-1. In other words, it will move directly behind the car in front of it.
3. Randomization: With some probability, *p*, the velocity of each vehicle, if greater than zero, is decremented by one.
4. Movement: Each vehicle is advanced *v* sites.

The original model assumes that when a car reaches the "end" of the road, or defined array, it will loop back to the beginning. However, for our implementation we will remove cars from the map when it reaches the end of the road, and generate new cars at the "beginning" of roads with the car spawner function, described in section 2.4. While this model is rather simple, it provides a realistic simulation of traffic flow. The third step, randomization, is essential for simulating realistic traffic conditions as it takes into consideration human behavior and other external conditions [1].

2. Implementation of Code

The code was split into a number of different parts. The main components are an environment map, which initializes a grid, a car calculations function, which calculates the new position and velocities of each car, a car spawner, which generates new cars at the beginnings of roads, and a simulation loop, which calls all the aforementioned functions, runs multiple iterations of the code. Apart from the grid, we also maintain lists of traffic lights and cars on the road. We iterate over this list of cars, and make kinematic updates based on the current state of the grid. At the end of every iteration, we also spawn a set of cars, based upon a tunable Poisson probability distribution. A graphical representation of the components of the code and the inputs and outputs are shown in figure 1.

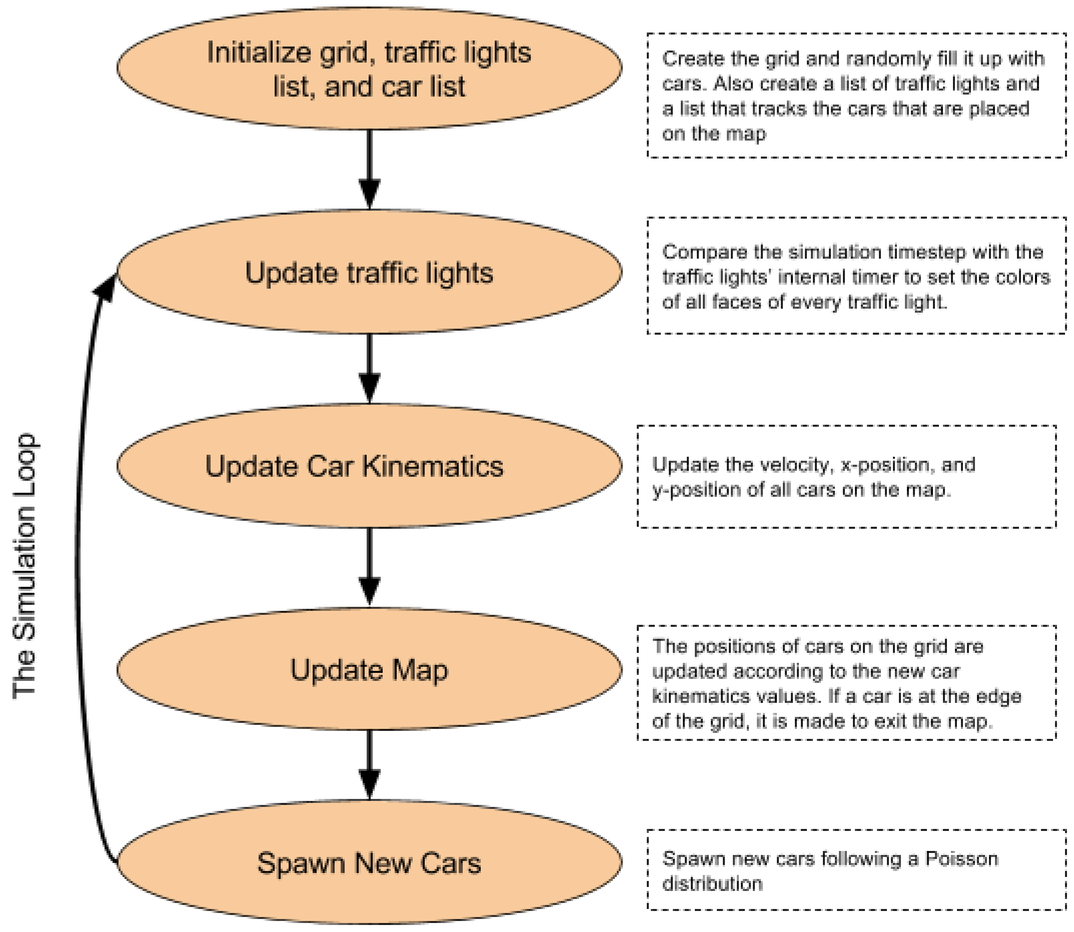

Initialize grid, traffic lights list, and car list
Create the grid and randomly fill it up with cars. Also create a list of traffic lights and a list that tracks the cars that are placed on the map
Update traffic lights
Compare the simulation timestep with the traffic lights' internal timer to set the colors of all faces of every traffic light.
Update Car Kinematics
Update the velocity, x-position, and y-position of all cars on the map.
Update Map
The positions of cars on the grid are updated according to the new car kinematics values. If a car is at the edge of the grid, it is made to exit the map.
Spawn New Cars
Spawn new cars following a Poisson distribution
The Simulation Loop

Figure 1: Graphical Representation of Code Interaction

2.1 Header File

2.1.1 Global Structures

A traffic.h file was created for the different components of the code to communicate. Several structures were defined and used in the simulation:

<u>Car Structure</u>: The car structure was defined as follows:

```
typedef struct car {
     char map_elem; //Car direction
     int id;        //Car ID
     int x_old;     //old x position
     int y_old;     //old y position
     int v_old;     //old velocity
     int x_new;     //new x position
     int y_new;     //new y position
     int v_new;     //new velocity
     int stop_time; //sum of timesteps when car is stopped
} car;
```

Each “car” stores the direction that the car should be moving in, the car ID, the old and new positions and velocities. The car structures are stored in a car list array, and the unique ID is also stored in the grid structure, which will be described in section 2.2.

The stop_time metric is required for analyses where we study the effect of various tuning parameters on the time a car spends stopping. In the course of our simulation, whenever we update the kinematics of a car, we check if its prior velocity was zero. If so, we increment the stop_time field by 1.

Cell Structure: The grid map, which is where the roads are defined also contains structures as defined below:

```
typedef struct cell{
      char map_elem;
      int car_id;
}cell;
```

Each cell on the grid will contain a map element and a car ID, which corresponds to the ID of the car in the car list. If there is no car in that cell on the road, the car ID is set to -1. The cell also contains a map element which indicates which direction the traffic on the road is heading. The car_id element serves as an index connecting the grid to the list of cars that we iterate over and update at every timestep.

Traffic Light Structure: The traffic light structure holds the following information.

```
typedef struct trafficLight{
      int id;
      int x;
      int y;
      int timer;
      int time_red;
      int time_green;
      int time_yellow;
      int northSouthLight; //the color of the NS facing lights
} trafficLight;
```

It is only necessary to hold the information for one of the directions (we picked North-South), because the East-West lights will always be opposite to the North-South lights.

2.1.2 Shared Functions and Global Variables

The header file also defines global variables like the cardinal directions and road elements, the spawn rates for each direction, maximum velocity, grid edges, and traffic light color definitions. The last part of the header file contains function prototypes for shared functions that are called in the simulate function.

2.2. Grid Map

The grid map is implemented as a matrix, with the map elements as shown below. Every cell in the grid is a *cell* struct, which consists of two fields: *map_elem* and *car_id*.

The *map_elem* field denotes what that particular section of the grid represents on the physical map. In the figure below, we present a representation of the map_elem fields for all cells in a single intersection grid. Here, the letters N,S,E, and W represent northbound, southbound, eastbound, and westbound road elements respectively. The elements J represent junction elements around the intersection. Finally, the elements G represent locations where ghost cars can be spawned. We use ghost cars, which are essentially imaginary cars, to simulate the effect of traffic lights being turned red. This ghost car strategy will be explained in detail later in the report. Note that the not all the ghost car locations will actually be used. Ghost cars are only generated on one side of the intersection, and past the junction, the cars on the other side of the road will be free to continue moving.

The *car_id* field of the *cell* structure is the id of the car that occupies that cell in the map. If there are no cars on the map, we simply set the *car_id* field of that cell to be -1. Over the course of the simulation, the grid map will be updated to reflect any changes to car kinematics, as well as the spawning and removal of cars from the grid. When the kinematics of a car need to be updated, the kinematics functions will relate the *car.id* element of a car to the *grid.car_id* element on the grid to identify how many cells they can move without crashing into another car.

Apart from being used to identify the location of various elements over the course of the simulation, the grid also allows us to save and visualize snapshots of the grid at every time step.

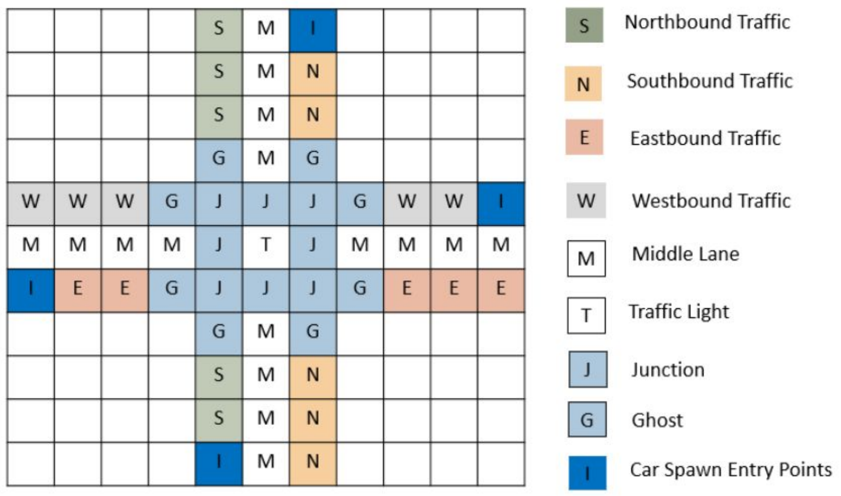
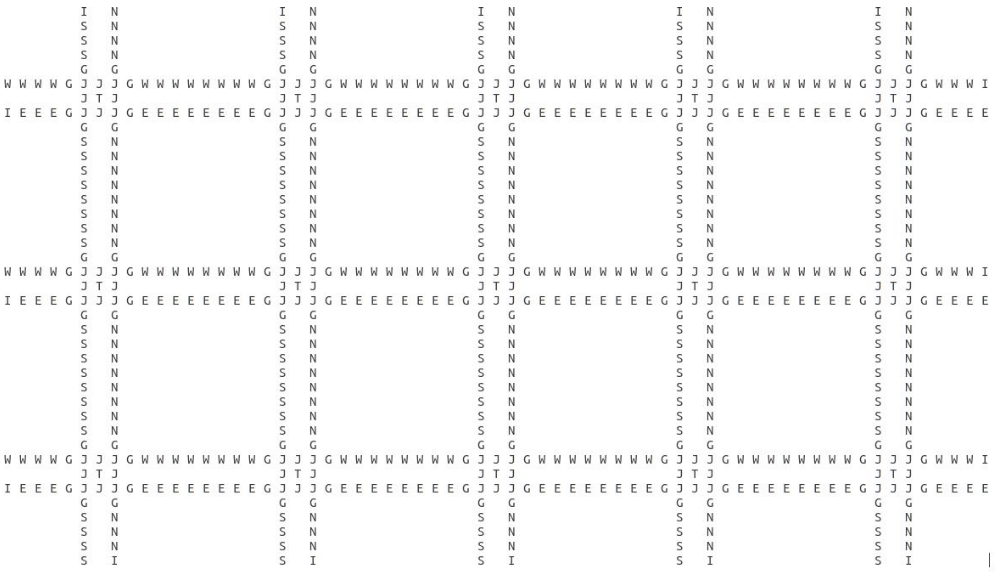


Figure 2: The Grid Map

For the purposes of this simulation, we only require a single intersection as pictured above. However, our grid generator has been designed to allow us to create an arbitrarily large grid with as many intersections as needed. The function takes as input the total length of road required, and the number of horizontal and vertical intersections required. Using this information, the function computes the appropriate locations for traffic lights and then builds the rest of the map around these lights. Below is an example of a grid created using 5 horizontal blocks and 3 vertical blocks.



```
          I   N                     I   N                     I   N                     I   N                     I   N
          S   N                     S   N                     S   N                     S   N                     S   N
          S   N                     S   N                     S   N                     S   N                     S   N
          S   N                     S   N                     S   N                     S   N                     S   N
          G   G                     G   G                     G   G                     G   G                     G   G
W W W W G J J J G W W W W W W W W G J J J G W W W W W W W W G J J J G W W W W W W W W G J J J G W W W W W W W W G J J J G W W W I
          J T J                     J T J                     J T J                     J T J                     J T J
I E E E G J J J G E E E E E E E E G J J J G E E E E E E E E G J J J G E E E E E E E E G J J J G E E E E E E E E G J J J G E E E E
          G   G                     G   G                     G   G                     G   G                     G   G
          S   N                     S   N                     S   N                     S   N                     S   N
          S   N                     S   N                     S   N                     S   N                     S   N
          S   N                     S   N                     S   N                     S   N                     S   N
          S   N                     S   N                     S   N                     S   N                     S   N
          S   N                     S   N                     S   N                     S   N                     S   N
          S   N                     S   N                     S   N                     S   N                     S   N
          S   N                     S   N                     S   N                     S   N                     S   N
          S   N                     S   N                     S   N                     S   N                     S   N
          G   G                     G   G                     G   G                     G   G                     G   G
W W W W G J J J G W W W W W W W W G J J J G W W W W W W W W G J J J G W W W W W W W W G J J J G W W W W W W W W G J J J G W W W I
          J T J                     J T J                     J T J                     J T J                     J T J
I E E E G J J J G E E E E E E E E G J J J G E E E E E E E E G J J J G E E E E E E E E G J J J G E E E E E E E E G J J J G E E E E
          G   G                     G   G                     G   G                     G   G                     G   G
          S   N                     S   N                     S   N                     S   N                     S   N
          S   N                     S   N                     S   N                     S   N                     S   N
          S   N                     S   N                     S   N                     S   N                     S   N
          S   N                     S   N                     S   N                     S   N                     S   N
          S   N                     S   N                     S   N                     S   N                     S   N
          S   N                     S   N                     S   N                     S   N                     S   N
          S   N                     S   N                     S   N                     S   N                     S   N
          S   N                     S   N                     S   N                     S   N                     S   N
          G   G                     G   G                     G   G                     G   G                     G   G
W W W W G J J J G W W W W W W W W G J J J G W W W W W W W W G J J J G W W W W W W W W G J J J G W W W W W W W W G J J J G W W W I
          J T J                     J T J                     J T J                     J T J                     J T J
I E E E G J J J G E E E E E E E E G J J J G E E E E E E E E G J J J G E E E E E E E E G J J J G E E E E E E E E G J J J G E E E E
          G   G                     G   G                     G   G                     G   G                     G   G
          S   N                     S   N                     S   N                     S   N                     S   N
          S   N                     S   N                     S   N                     S   N                     S   N
          S   N                     S   N                     S   N                     S   N                     S   N
          S   I                     S   I                     S   I                     S   I                     S   I
```

Figure 3: Sample Output of grid maps with multiple intersections

2.3. Traffic Lights

2.3.1 The Lights

The flow of traffic at an intersection will be directed using a traffic light structure. The program that controls this structure will essentially be a simple state machine that alternates between red, yellow, and green lights. The red and green traffic light colors have the usual meanings that inform cars to either stop or move. The yellow light color is used as an intermediary buffer between red and green. While in real life, the yellow light is meant to serve more as a signal for cars to prepare to stop or slow down, for the purposes of our simulation, we assume that the drivers in our simulation are perfect, law-abiding citizens and just come to a complete stop at yellow lights. We require this yellow light stopping to prevent collisions - if a car leaves an intersection immediately after the light stops being red, it might crash into cars moving in the orthogonal direction that are still at the junction. The yellow light provides cars in the orthogonal direction time to safely get to the other side.

On the physical grid, a traffic light will need to have four faces, each pointed towards the four directions North, South, East and West. The traffic light will be programmed such that it alternately allows the Northbound/Southbound and Eastbound/Westbound traffic to flow while stopping the other direction.

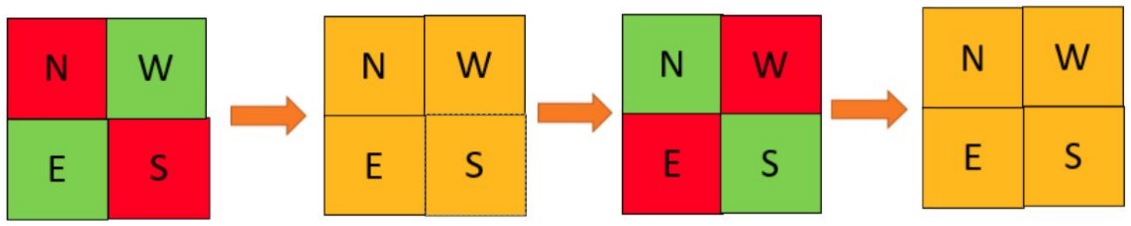


Figure 4: Traffic Lights. Note the yellow light works as a buffer between red and green lights.

Given this interpretation of the lights, we note that each traffic light only needs to store color information for a single direction. If we define green to be 1, red to be -1, and yellow to be 0 , we can derive lights for all direction by only using the state of the north facing light.

North facing Light = South facing Light = - (East Facing Light) = - (West Facing Light)

Note that, using the above definition, when the north-south light is yellow, so is the east-west light. As mentioned earlier, this additional buffer time between light changes prevents collisions at intersections.

2.3.2 Light Control

We are able to control the behavior of a traffic light by manipulating its *time_color* fields for the three possible light states. These *time_color* fields indicate the number of timesteps that a certain color will remain active. For example, by increasing the value of *time_red*, we can therefore simulate traffic behavior where cars spend a significant amount of time waiting at the light. The sum of these time_color fields defines the period of our traffic light, which is the time required for the traffic light to complete a cycle:

Traffic light period = time_red + time_yellow + time_green + time_yellow

The internal timer of a traffic light is incremented at every time step, and is reset to zero when it is equal to the traffic light period.

In the figure below, we plot the required behavior of one of our traffic lights (a) and the behavior obtained in our simulation (b). As we can see in (b), the traffic lights alternate between red, yellow, and green in the period fashion required by our simulation.

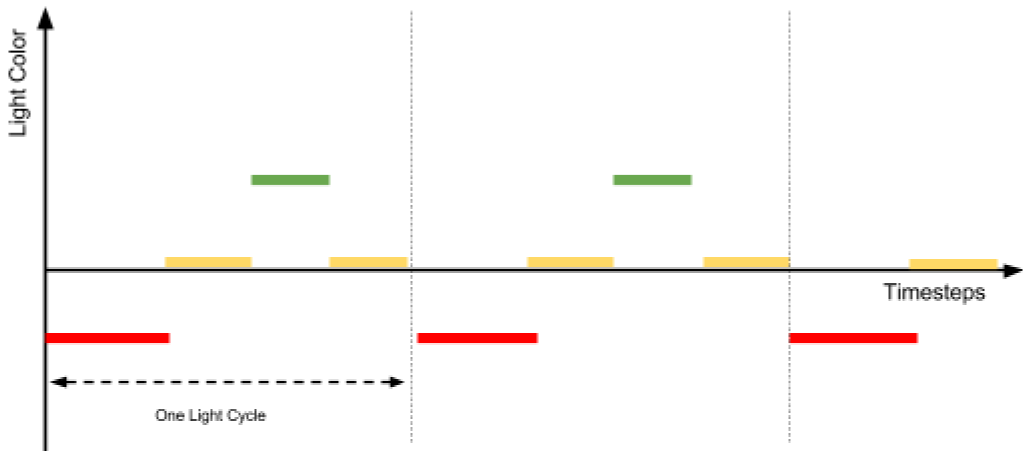


Figure 5a: Traffic Light Cycle Design

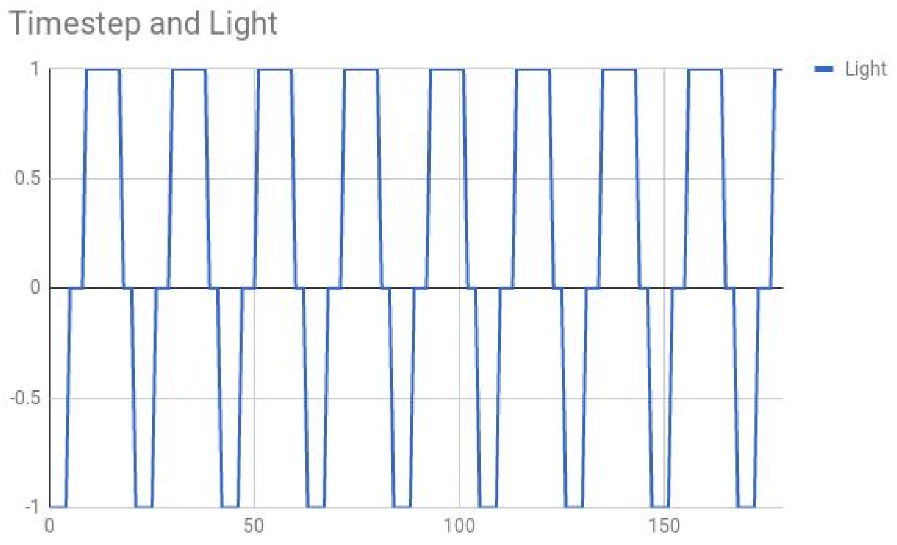


Figure 5b: Traffic Light Cycle Implementation: Obtained in simulations for short durations of yellow and red light, and a long duration for green light.

2.4. Spawning and Removing Cars from the Map

The function update_spawners() generates cars along the edges of the map by referencing the locations marked by "I" for *into* the map. The physical spawn locations are actually a distance v_{max} from the edge of the map matrix where "I" is marked. When a car is closer than v_{max} from the edge of the matrix, checked by the function get_bounds(), it is removed from the map and deactivated. The thickness of this edge boundary was chosen to be the maximum distance a car could travel in a single time-step (1 sec). This also means that the physical map is smaller than the map matrix. The physical map's width is $2v_{max}$ smaller than the matrix width and $2v_{max}$ smaller than the matrix height.

The spawn rate for each direction of traffic is adjustable by changing the mean time between spawn events. By lowering the mean spawn time for cars moving south, cars will enter the north side of the map with a higher frequency. Alternatively, increasing the mean spawn time will lower the spawn frequency. These parameters are useful for simulating scenarios where the flow of traffic in one direction differs from the other directions.

2.5. Car Kinematics

To generate the car kinematics, several functions were created. The emptycellcount() function returns the number of spaces from a car to the car in front of it. It does this by using the the direction and location provided in the car structure by the active car list, and looping in the appropriate location and direction on the grid. It will continue to loop and increment a counter until it either it sees a car in front of it (Car ID is not equal to -1) or it reaches the end of the grid.

The integer returned by this function was used in the update_velocity() function, which determines if and how the velocity should change based on the current velocity and the maximum velocity. If the car is below the maximum velocity, it is incremented. However, if the number of empty spaces in front of the car (the number returned by emptycellcount()) is less than the updated velocity, then the velocity becomes the number of spaces it can move, which is less than *v*. Third, the velocity function determines whether there should be any randomization for this particular car by generating a random number and comparing it to a set probability. If there is, then the velocity is decremented. The output of the update_velocity() function is used by the car_update() function, which then calculates the new x,y positions on the grid.

This accounts for car acceleration and stalling, but what about cars stopping? There are two conditions for cars to stop. If there is a car in front of it, or if there is a red light. Therefore, in order to simulate a red light, the code will generate a "ghost" car at the positions labeled as "G" on the grid. This only updates the grid, and does not generate a new car structure, and therefore the active car list stays intact. The ghost car will stay in that position for both the yellow and red light signals, and will be removed when the light turns green. The behavior is opposite for the North-South bound traffic and for the East-West bound traffic.

2.6. Simulation Loop

The simulation loop calls all the aforementioned functions and updates the grid map. The first step is the initialization. It will first initialize traffic lights, initialize the grid map, initialize cars on the map, initialize ghost cars, and then spawner cars. Then, it will update for the next iteration. The simulation loop will continue to run until a set simulation time.

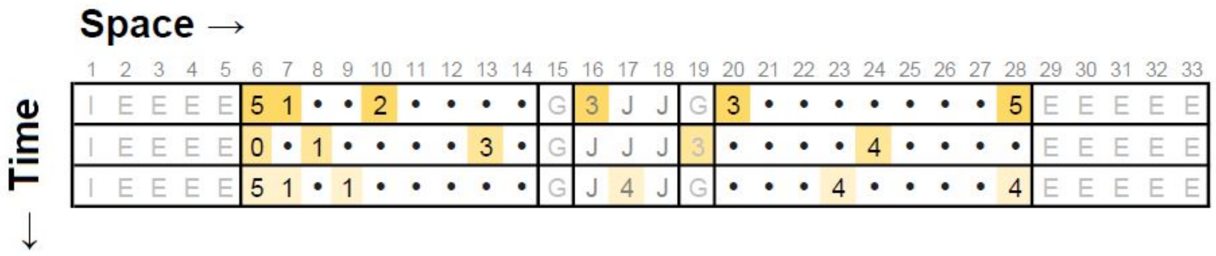
3. Testing and Verification of Code

3.1 Verification of Car Movement

We tested the code to make sure cars were moving at the appropriate speed and in the appropriate direction. A sample three timesteps were output for eastbound traffic (figure 6a). The letters in grey on the edges of the map, are the outer bounds of the map, where cars will be removed, and therefore, are void of cars. The dots on the inside of the map represent areas on the road where there are no cars and the numbers represent the velocities of the cars. G represents locations where ghost cars could spawn, and J represents the junction, and the top row of consecutive numbers are indexes. Because cars cannot be spawned in the outer bounds of the map, it is spawned at the sixth location on the map (if v_{max} is 5), in this map. Figures 6a through 6d are not printed from the same time step, to demonstrate different scenarios in traffic flow (the change of a light turning from red to green, or cars piling up behind a red light). The purpose of these figures is only to demonstrate that the velocity and movement behavior of these vehicles is correct.

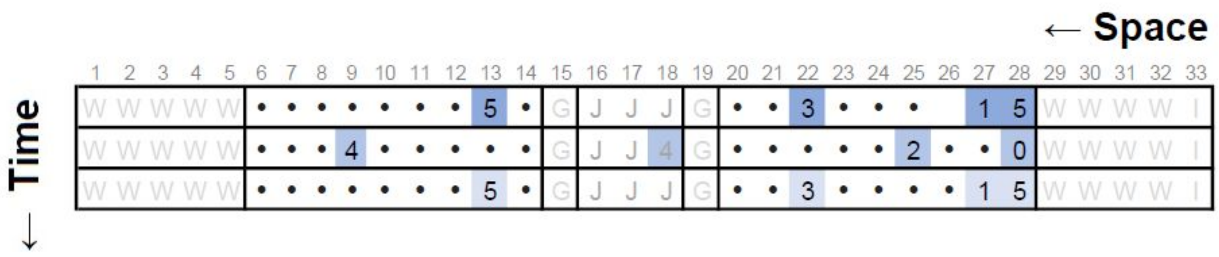
Because figure 6a shows eastbound traffic, the traffic is moving from "left" to "right" on the page. Starting from the top row, there are currently 6 cars on the road. The first car is spawned with an initial velocity of 5, but sees that there is a car in front of it so it cannot move forward, and therefore has a velocity of 0 in the next timestep. The car in position 7 in the first row has a velocity of 1, but randomization kicks in, so its velocity in the next time step is also 1. In the first row, the car at position 10 has a current velocity of 2, but speeds up to 3 in the next time step. This logic continues for all of the elements on the grid. The car at position 28 in the first row is removed from the map in the next step. In the third row, a new car is spawned at position 6 again with a velocity of 5, which is randomly set between 0 and v_{max} in the car spawner function

Figure 6b shows westbound traffic, so in this case traffic is moving from right to left. Looking at the car that is in position 13 in the first row, the car experiences randomization, so it decreases its velocity to 4 in the next step. The cars enter the map at position 28, on the grid. In the first row, the car is spawned, but there is a car directly in front of it, and therefore it has a velocity of zero in the next step. Note that in both the eastbound and westbound traffic, the light is green in these sample outputs.



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
I	E	E	E	E	5	1	•	•	2	•	•	•	•	G	3	J	J	G	3	•	•	•	•	•	•	•	5	E	E	E	E	E
I	E	E	E	E	0	•	1	•	•	•	•	3	•	G	J	J	J	3	•	•	•	•	4	•	•	•	•	E	E	E	E	E
I	E	E	E	E	5	1	•	1	•	•	•	•	•	G	J	4	J	G	•	•	•	4	•	•	•	•	4	E	E	E	E	E

Figure 6a: Eastbound Velocity Outputs

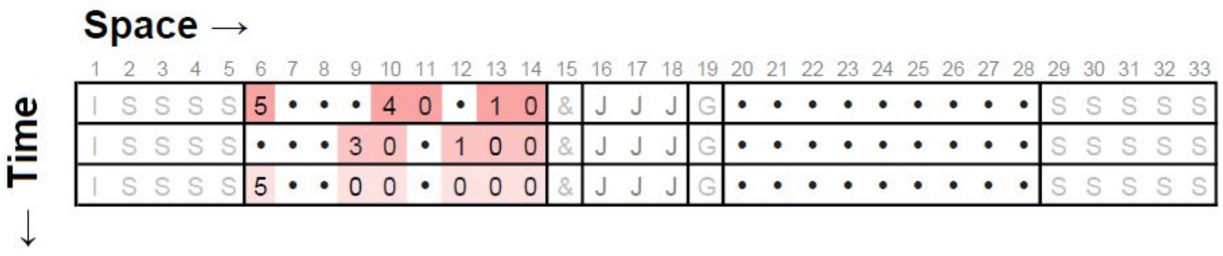


1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
W	W	W	W	W	•	•	•	•	•	•	•	5	•	G	J	J	J	G	•	•	3	•	•	•		1	5	W	W	W	W	I
W	W	W	W	W	•	•	•	4	•	•	•	•	•	G	J	J	4	G	•	•	•	•	•	2	•	•	0	W	W	W	W	I
W	W	W	W	W	•	•	•	•	•	•	•	5	•	G	J	J	J	G	•	•	3	•	•	•	•	1	5	W	W	W	W	I

Figure 6b: Westbound Velocity Outputs

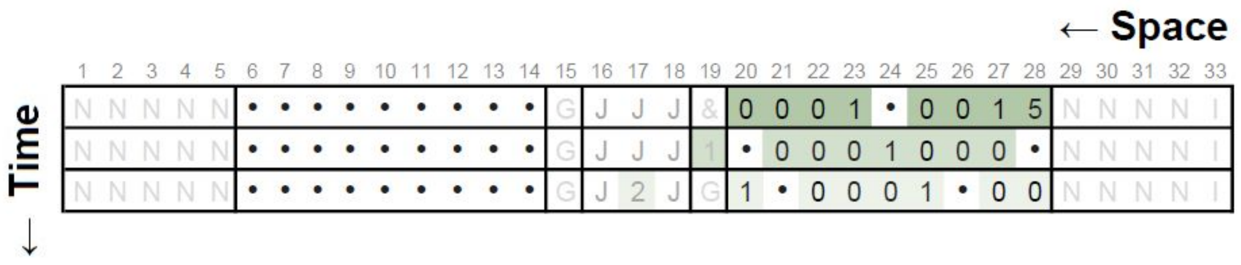
Figures 6c and 6d show Southbound and Northbound traffic, respectively and are represented horizontally. At this particular time, the light is red for N-S traffic, and they are stopped in front of the intersection, where the ampersand (&) represents a ghost car. Figure 6c demonstrates that the cars far away from the traffic lights (the car with a velocity of 5 in position 6) will continue to move towards the intersection until it reaches the "jam" and is forced to stop. This, however, is also one of the weaknesses of the Nagel-Schreckenberg model, which is further discussed in the conclusions section of this report.

The northbound traffic in figure 5d shows the case where the traffic light has just switched back to green (after the first time step) and the traffic has begun to move through the intersection. This is a slow process, as the leading car needs at least five time steps to reach v_{max}.



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
I	S	S	S	S	5	•	•	•	4	0	•	1	0	&	J	J	J	G	•	•	•	•	•	•	•	•	•	S	S	S	S	S
I	S	S	S	S	•	•	•	3	0	•	1	0	0	&	J	J	J	G	•	•	•	•	•	•	•	•	•	S	S	S	S	S
I	S	S	S	S	5	•	•	0	0	•	0	0	0	&	J	J	J	G	•	•	•	•	•	•	•	•	•	S	S	S	S	S

Figure 6c: Southbound Velocity Outputs



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
N	N	N	N	N	•	•	•	•	•	•	•	•	•	G	J	J	J	&	0	0	0	1	•	0	0	1	5	N	N	N	N	I
N	N	N	N	N	•	•	•	•	•	•	•	•	•	G	J	J	J	1	•	0	0	0	1	0	0	0	•	N	N	N	N	I
N	N	N	N	N	•	•	•	•	•	•	•	•	•	G	J	2	J	G	1	•	0	0	0	1	•	0	0	N	N	N	N	I

Figure 6d: Northbound Velocity Outputs

4. Results and Analysis

4.1 Flow and Density

The plot of flow vs. density is considered the "fundamental diagram" of traffic flow and has the following characteristics. When the density is zero, flow will also be zero because there are no cars on the road. When the number of vehicles increases, both density and flow increase. As density reaches a maximum, flow is also zero because the vehicles are in a gridlock and cannot move. Theoretically, there should be some density at which the flow is maximized. Figure 7 show plots of the simulation data in comparison to real traffic data from the original Nagel and Schreckenberg paper, which shows the model is relatively valid. The density in this case is defined by equation (1), and the flow is the number of cars that cross a certain location per time step, which in this case, was the number of cars that crossed the intersection from all four directions.

$$\rho = \frac{N}{L} = \frac{Number\ of\ Cars}{Number\ of\ Sites\ of\ the\ Road} \qquad (1)$$

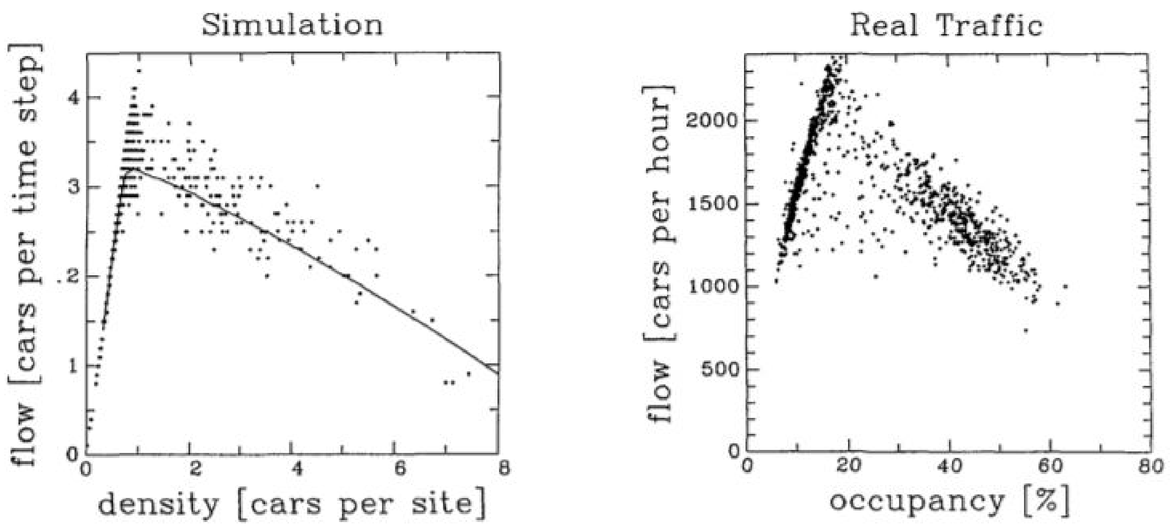


Figure 7: Original Density vs. Flow Plot from Nagel & Schreckenberg and comparison with real traffic data [1]

Figure 8a shows the fundamental flow diagram for our simulation. The general shape is consistent with those from from the original Nagel and Schreckenberg paper, and from real traffic data. Because our model differed from the original Nagel Schreckenberg model where the number of cars could vary at any given time (whereas the original model assumed a circular road), the density and flow would depend on the spawn rate of the cars, more so than it would depend on the defined density in the random fill initialization. Therefore, in order to actually get the desired average density, the spawn rates were adjusted on a range between one second mean time, to 35 seconds mean time. For the lower densities, the spawn rate was increased and for higher densities, the spawn rate was set to 1 second mean time, or the maximum. For this simulation, traffic light times were set to 60 seconds green, three seconds yellow and 60 seconds red, the probability of slowdown is 0.4, and the road length (one side) was 30. The simulation time was 200 seconds.

Even with the maximum spawn rate, the density of the graph could not make it above a density of approximately 0.5. This is most likely because multiple cars can be removed from a graph at a time, whereas only one car can be spawned at one location at a time, and if there is a buildup of traffic, the newly generated car cannot move far, thus artificially slowing down the spawn rate.

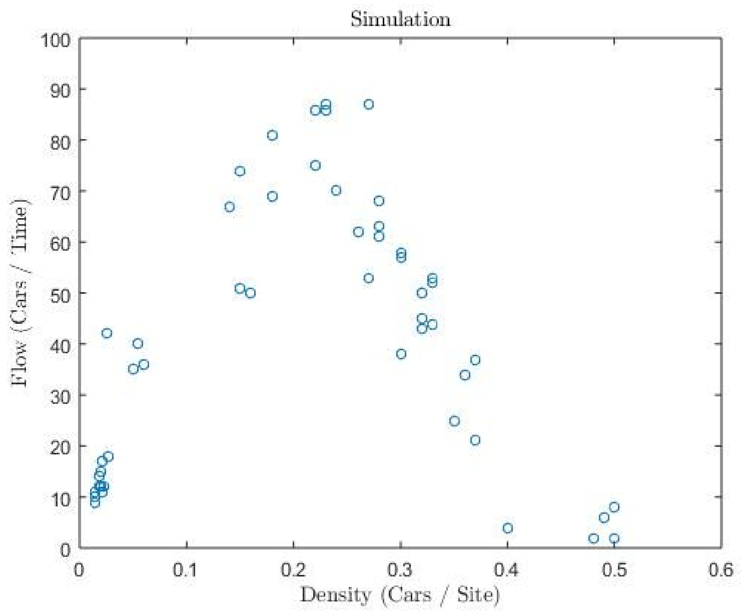


Figure 8a: Flow vs. Density - Fundamental Traffic Flow Diagram

Figure 8b shows the same plot, except with the originally intended density overlaid, meaning the density of the random fill. The general shape of the graph is the same, except density has a larger range. Above a set density of 0.3 the spawn rate was set to 1 second, but the actual density still begins to separate itself from the original random fill density.

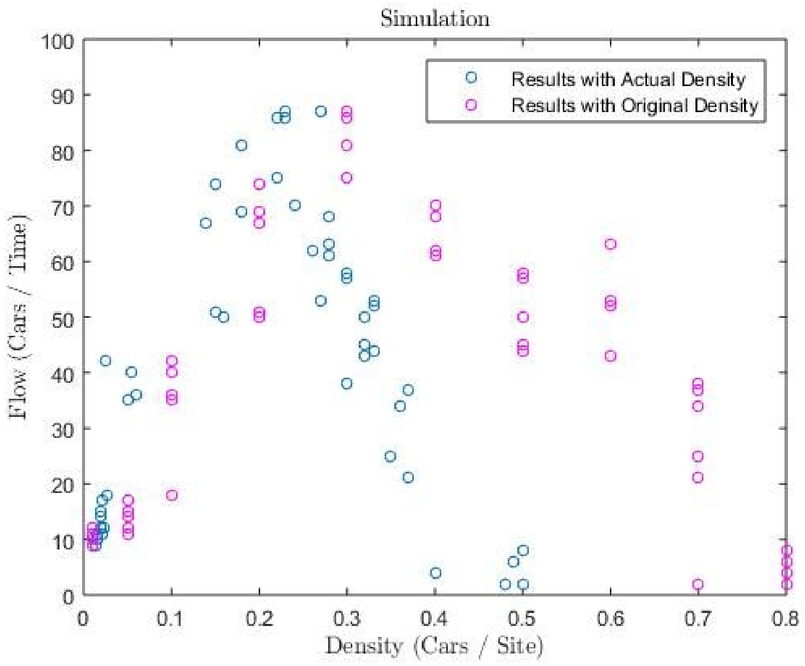


Figure 8b: Flow vs. Density with originally defined density overlaid

The maximum value of the fundamental diagram represent the optimal conditions for traffic flow at this simulated intersection. In this case, the maximum number of cars that can get through the intersection in 200 seconds is approximately 90 cars at a density of 0.25.

4.2 Number of Cars over Time

The number of cars on the map at any given time N(t) is influenced primarily by the rate at which cars enter the map. On the other hand, the rate at which cars exit is limited by the intersection. Figure 9 displays N(t) over a simulation time of 1000 seconds for various mean-times $\langle T_S \rangle$ between spawn events (from 2 sec to 8 sec). During the first 100 second, N(t) appears to be mostly decreasing because the initial density of cars is too high. After this initial period, N(t) begins to oscillate around an average. This average appears to stabilize as $t \to \infty$.

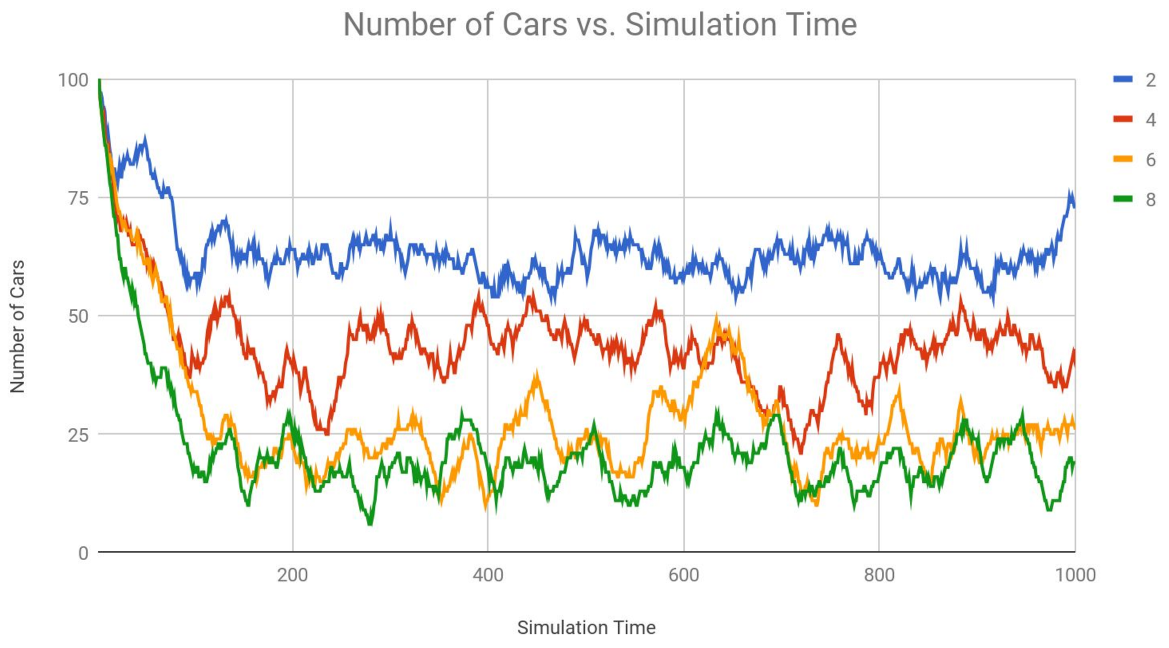


Figure 9: Number of Cars vs. Time

If the average number of cars per time ⟨N⟩ was evaluated during the first 100 seconds, it would also be strictly decreasing. To avoid this instability, ⟨N⟩ was evaluated after 200 seconds equation 2.

$$\langle N \rangle = \frac{Avg\ Cars}{Time} = \frac{\int N(t)\,dt}{\Delta t} \qquad (2)$$

Because of the discrete time-steps, N(t) was integrated using the midpoint rule with an interval length of one second. Figure-10 displays ⟨N⟩ vs twelve average spawn rates $\langle R_S \rangle$, the reciprocal of the average time per spawn $\langle T_S \rangle$. As expected, the average number of cars increases with the average spawn rate, though the rate diminishes as the maximum spawn rate is approached.

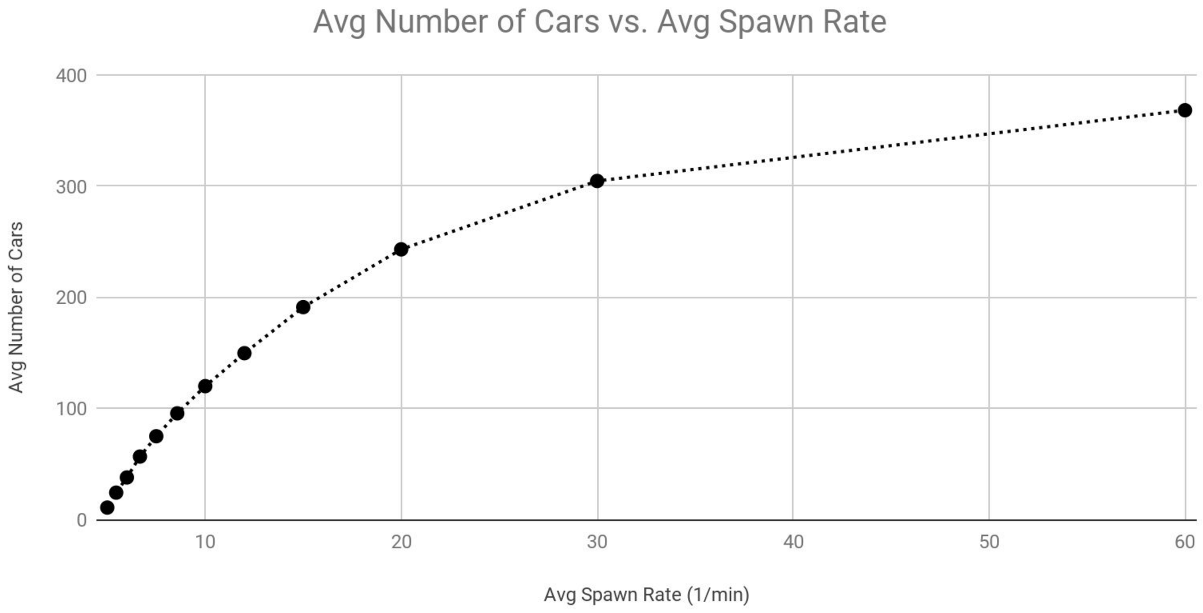


Figure 10: Avg Number of Cars vs. Avg Spawn Rate

The maximum number of cars on the map was limited by the road length between an intersection and the map's edge. Both Figures 9 and 10 were simulated on road lengths of 30 cells between an intersection and edge. Because the traffic lights were green for 60 seconds in both directions, an empty road leading to a red-light would fill before turning green with an average spawn rate of 30 cars/min. If the road lengths were increased to 60 cells, then more cars could wait at red lights when spawn rates are highest. Regardless, the overall shape of the plot would be expected to remain the same.

To keep the impact of random slowing to a minimum, its probability was kept to a conservative value of 0.2. The speed limit was set to 5 cells/sec (83 mph). Though this is not realistic for traffic near an intersection, this should have little effect on traffic approaching a red-light. The speed limit would most affect the density of cars passing through an uncongested green-light. For cars traveling at 5 cells/sec, the minimum spacing between each car must be 5 cells, leading to the smallest density. This was not considered for the Figures 9 and 10 but was studied more thoroughly when evaluating the average time spent traveling across the map for each car.

4.3 Stop Time Analysis

Another metric of interest in analyzing the flow of traffic is the amount of time cars on the grid spen waiting as opposed to moving. In our simulation, there are two factors that can lead a car to be stop - the presence of another car ahead of it or the traffic light in front of it being red. Therefore, the parameters that should affect the stop time of a car are the frequency of traffic light changes, the spawn rate of vehicles, and the probability that a car will randomly decrement its velocity. In reality, the third parameter represents human behavior and is not adjustable when designing traffic systems.

We define the Time Stopped Ratio as the ratio of time spent stopped versus the time spent moving:

$$Time\ Stop\ Ratio = \frac{Total\ Stop\ Time\ for\ All\ Cars}{Total\ Movement\ Time\ for\ All\ Cars}$$

4.3.1 Varying the Probability of Random Velocity Decrement

Below, we present the total stop ratios for a simulation run for 5000 timesteps and the mean time between car spawns set to a constant value of 5. Here, the blue curve represents. In the blue and orange curves presented below, the probability of random velocity decrement set to a 0.2 and 0.8 respectively. We see that overall the time stop ratio is higher for the case when cars are more likely to randomly decrease their velocity, causing cars behind them to stop. More interestingly, we see that for both regimes, there seems to exist an ideal number of time steps for which a green light is active which minimizes the time stop ratio for cars. For both cases, this value seems to be somewhere around 50-60 timesteps.

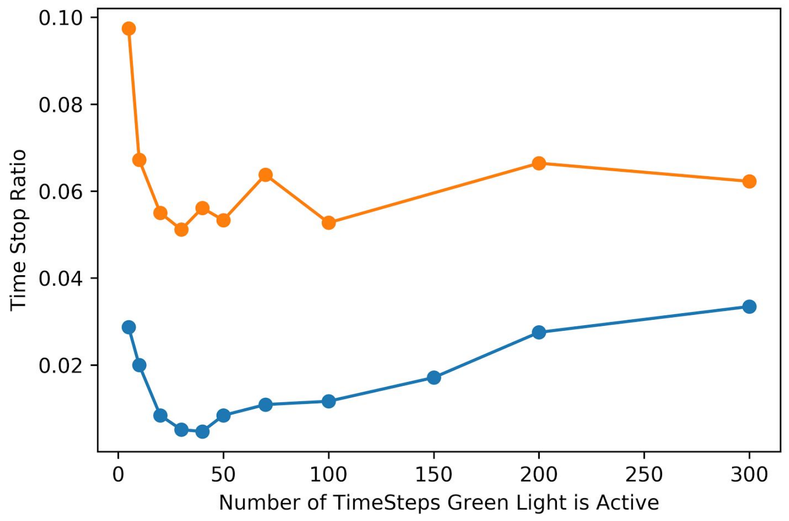


Figure 11 : Time Stop Ratio as a function of the number of green light active timesteps. The orange and blue curves represent simulations where the probability of random velocity decrement is set to 0.8 and 0.2 respectively.

In the above curve, we note that the number of time steps for which green lights and red lights are active are dependent: time_red = time_green - time_yellow. This explains why we do not observe monotonically decreasing curves in the above plots. The optimal values of time_green observed here represent the optimal ratio of red/green lights within our simulation.

For both cases, we observe that there is an optimal value of the number of timesteps for which the light is green and red. This information can be useful to design traffic light systems that adapt to the behavioral patterns of the drivers on the road.

4.3.1 Varying the Spawn Rate

Below, we present the time stop ratios for various green light active time step choices, for the cases of high (red) and low spawn rates (green). In these low and high spawn rate regimes, new cars are being spawned onto the map at mean times of 10 and 1 second respectively, following a Poisson distribution.

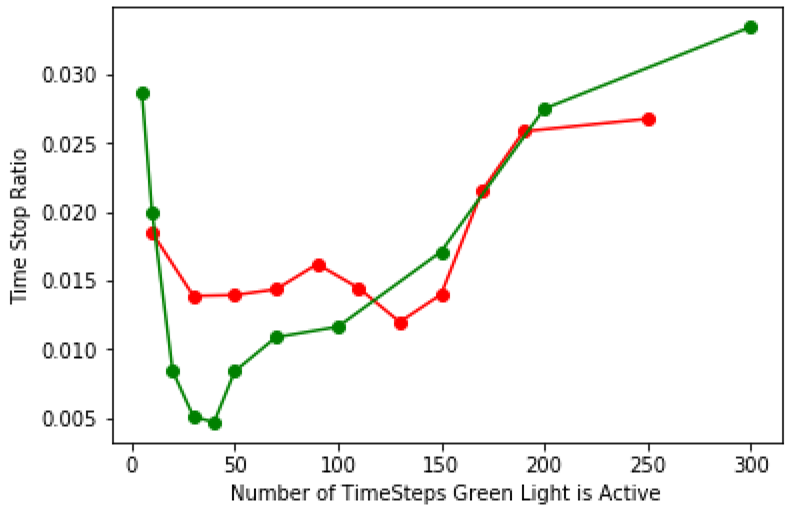


Figure 12 : Time Stop Ratio as a function of the number of green light active timesteps. The red and green curves represent simulations where the mean time between car spawns was set to 1 and 10 seconds respectively.

Here, for lower values of green light active time, we find that the time stop ratio is lower for the low spawn rate regime (green). However, for higher values of green light active time, the low spawn rate regime can even have higher values of time stop ratio than the high spawn rate regime. While counterintuitive, this is likely due to the fact with higher values of green light active time, we also obtain high values of red light active time. Therefore, beyond the optimal value of green light active time, we expect the cars to be stopped for a larger proportion of the time when the spawn rate is relatively low. However, for a higher spawn rate, the density of cars on the road add to the stopping of cars. Therefore, higher green/red light time durations could be serving as some form of regulation over the traffic flow and reducing the time stop ratio further.

For both cases, we observe that there is an optimal value of the number of timesteps for which the light is green and red. This information can be useful to design traffic light systems that adapt to the rate of of incoming cars onto an intersection.

4.4 The Effect of Randomization

When designing our simulation, we made the decision of introducing an element of randomization by ensuring that the velocities of a random number of cars on the grid are decremented by 1 at each time step. This design was motivated by the comments from Nagel and Schreckenberg indicating that randomization was necessary for the simulation to model "real" traffic.

To evaluate the effect of this randomization, we plot the velocity map of cars in the grid over 300 timesteps, with and without randomization. For simplicity, we pick an arbitrary direction (East-West) and follow the movement of cars along that road. Here, we have temporarily set the traffic lights to always be green, so that we are able to isolate the effects of the randomization. For both cases, we initialize the grid with a density of 0.7.

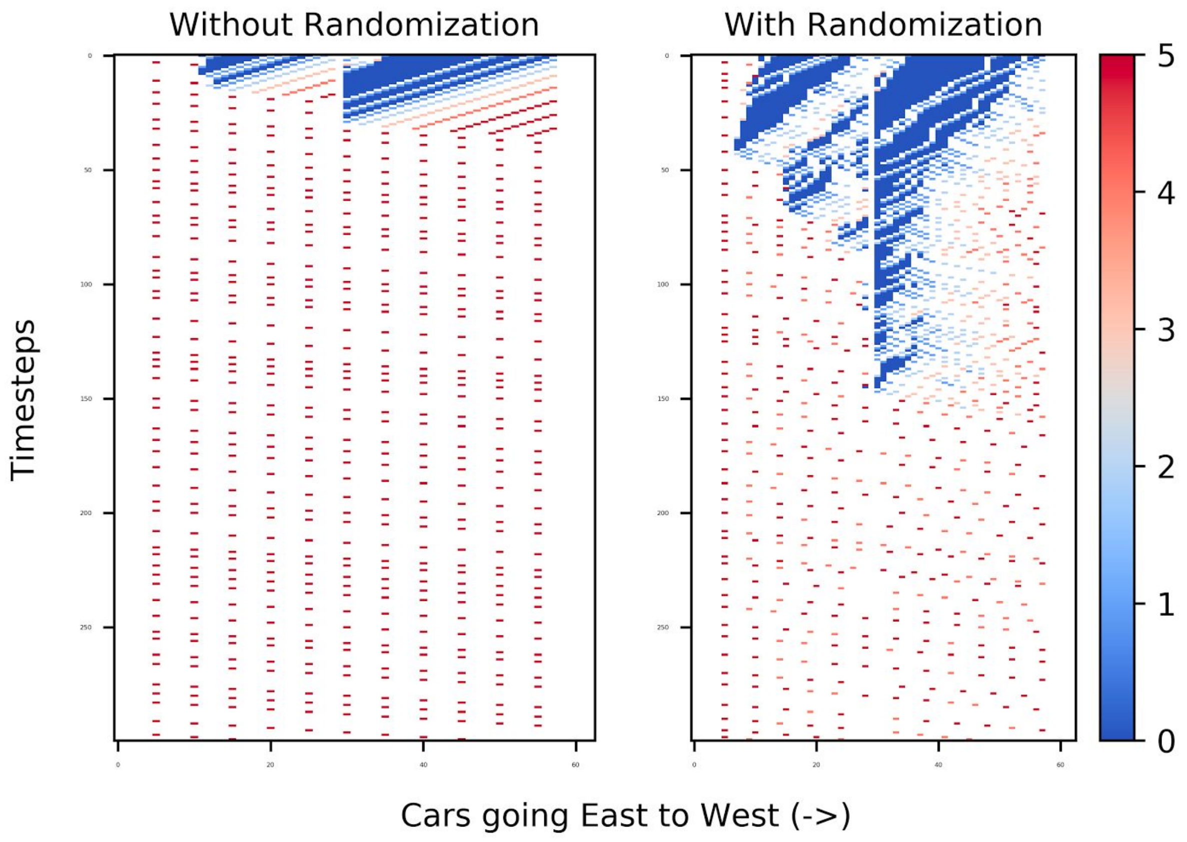


Figure 13: Velocity map of cars on the East-West road showing the effect of adding randomization to velocity updates. The color scale represents velocity.

As seen above , without the random velocity decrement, the traffic moves in an extremely mechanical manner. There is some traffic jam that is introduced early on in the simulation, which rapidly diminishes and disappears in a somewhat linear fashion. When we introduce the random velocity decrement, the traffic takes a much longer time to break free from the initial traffic jam. Furthermore, we see that the distribution of cars is no longer as mechanical and predictable. Therefore, the random velocity decrement has, in fact, made our simulation more aligned with "real" traffic. Here, the random velocity decrement serves to model the randomness within driver behavior.

4.5 Traffic Jams

One of the key motivations for this simulation is to analyze the flow of traffic upon varying different parameters. We are, therefore, interested in seeing the various patterns of traffic congestion that may arise over the course of the simulation. We begin by varying the initial density of cars placed on the map. Since the grid is symmetric in all four directions, we arbitrarily pick the East-West direction.

Since we want to fully isolate the movement in this direction, we have temporarily disabled car movements in the orthogonal direction (North-South). Furthermore, we also set the probability that a car's velocity will be set to zero at a given time step to be zero. Finally, the last remaining component of the simulation that can affect the car's movement is the traffic light at the intersection. We initialize the traffic lights such that the East-West traffic light is green for the first 64 seconds, red for the next 60 timesteps, and yellow for another 4 seconds. Note that since the yellow functions as a buffer light where cars cannot move, it essentially serves as an additional 4 seconds of red light at the end. With these adjustments, we ensure that the motion of the westbound traffic is only affected by other westbound cars, and the traffic light at the intersection.

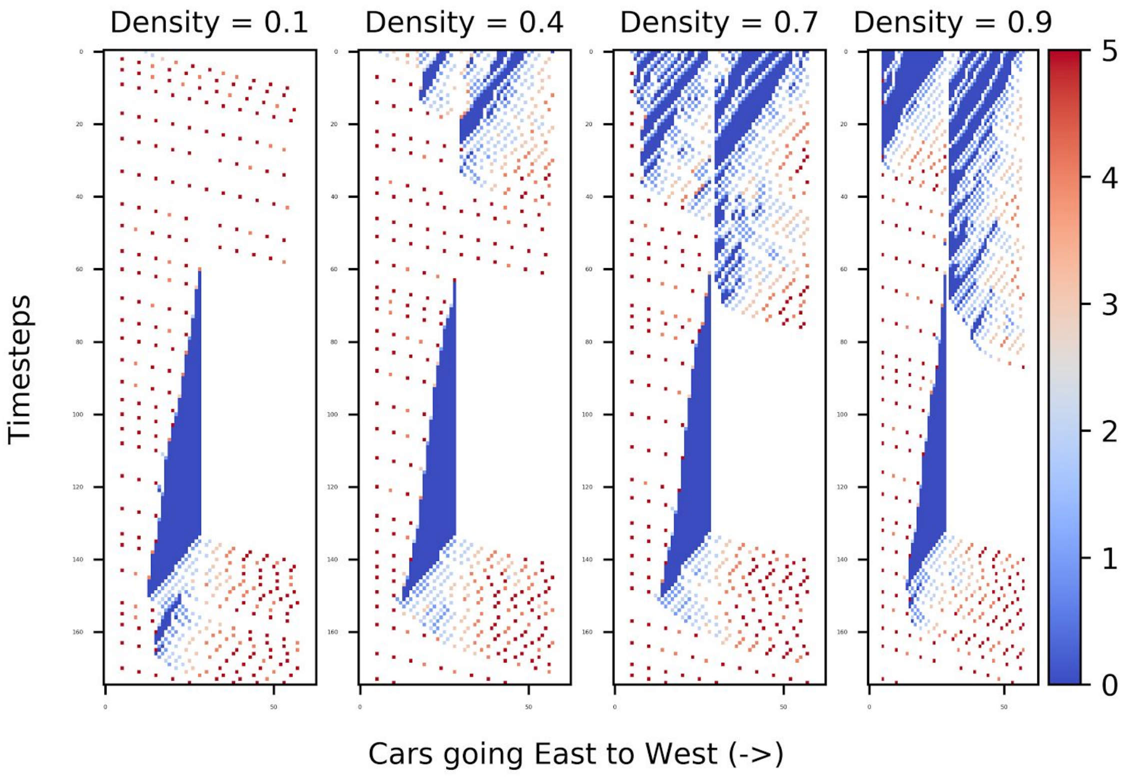


Figure 14: Velocity maps of cars on the East-West road showing the effect of altering the initial density of cars on the grid. The color scale represents velocity.

As shown in the figure above, the intensity of traffic jams increases as a function of car density. The sections of blue dots in the plot represent carst with 0 velocity. Therefore, a block of consecutive blue dots at a given time step represents a traffic jam along that section of the road. For the first 64 seconds, the East-West traffic light is green, so there are traffic queues visible on both sides of the intersection. However, after the first red light, the flow of traffic moves in a much more smooth manner, with queues building only on the left side of traffic light, and cars being released one after another.
At the low car density of 0.4, we see a few queues developing, which seem to resolve themselves out within the first 40 timesteps. As we increase the car density to around 0.7, we see a larger number of traffic jams dispersed along the road, which take longer to resolve and yield steady traffic flow.

In the figures above, we see that the blocks of stalled cars appear almost as tails emerging from the cars that were stopped at the first timestep. This result is consistent with that originally demonstrated by Nagel and Schreckenberg as shown below.

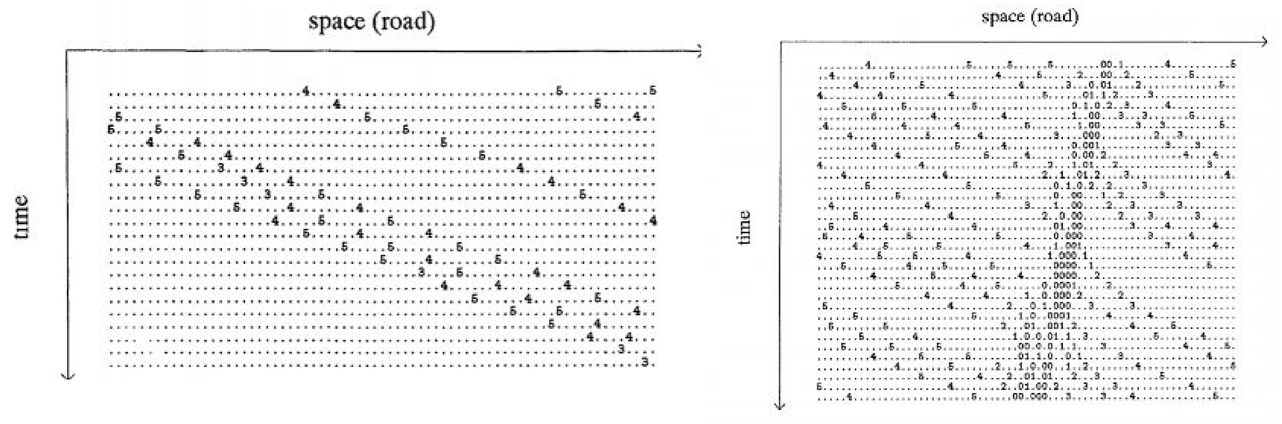


Figure 15: Original Space vs. Time Plots from Nagel & Schreckenberg [1]

This phenomena is consistent with the concept of shockwaves, which is an emergent behavior arising from traffic congestion and queuing. A shockwave illustrates the boundary between two traffic states characterized by different flow rates. When a stream of traffic flowing under certain conditions encounters traffic under different conditions, a shockwave is started that separates the two different traffic flow regimes. Generally shockwaves are of interest as they move across the traffic flow at their distinct velocity determined by the specifics of the interacting traffic flows. In our analysis, we observe a very simplistic mixture of traffic flows. When a stream of traffic flowing at some average velocity encounters a stopped car, it causes the formation of a traffic jam. This traffic jam essentially moves in the opposite direction of the traffic as the number of cars continues to pile up [3].

4.6 Traffic Regulation Through Lights

While the initial traffic flow is purely a function of the initial car density and car kinematics, we anticipate that with the introduction of traffic lights, we will be able to force a much more restricted and predictable traffic flow. Namely, we should able able to observe that the traffic lights regulate the flow traffic and release a steady stream of cars at regular intervals. In the figure below, we plot the car velocities for 1000 timesteps, accounting for many traffic light cycles while keeping the initial density to be relatively high at 0.7. Here we see can see the car movements correspond to the presence of the traffic light as expected. Initially, when the light is green, we see traffic jams on both sides of the intersection. Once the traffic light begins its cycle, the traffic jams on the right side of the road begin to reduce in intensity. At the beginning of the second red light, we see that there is barely any traffic jam on the right side.

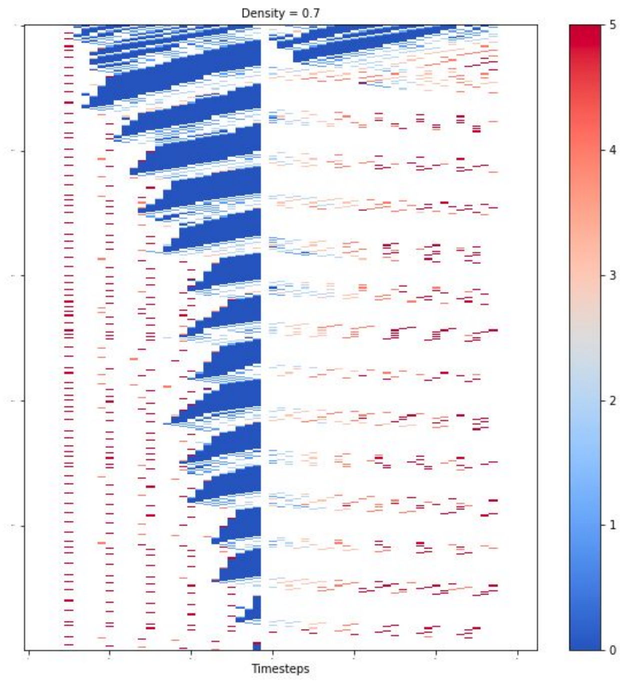


Figure 16: Velocity map of cars on the East-West road showing the regulating effect of traffic lights. The color scale represents velocity.

In the figure below, we plot the car velocities without (traffic light always green) and with traffic lights in place at the intersection. To see car dynamics at longer time steps, we have increased the number of cars spawned at every round of car spawns.

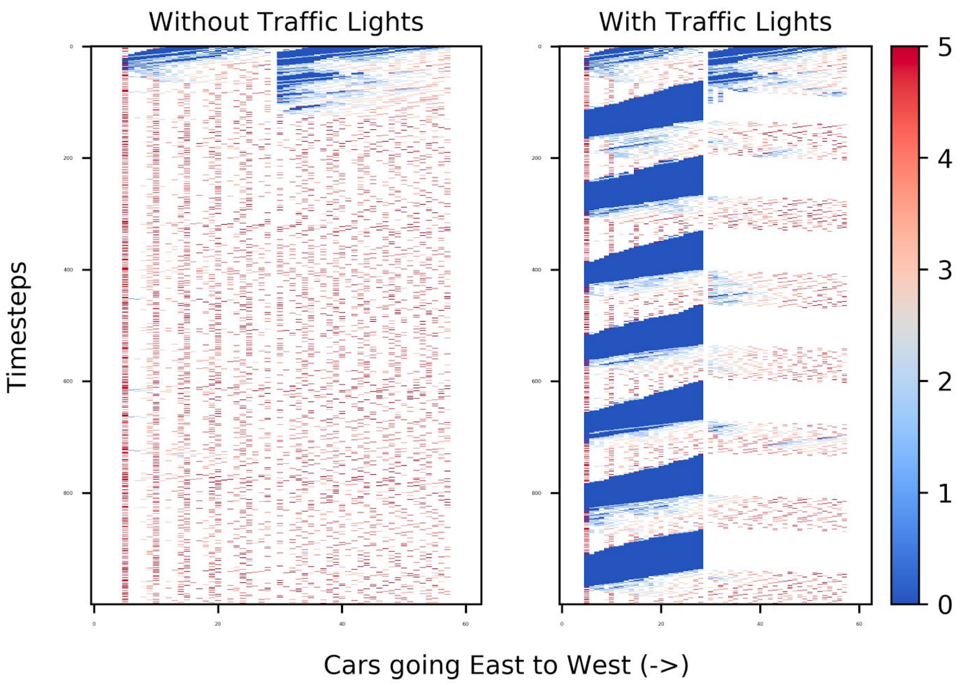


Figure 17: Velocity map of cars on the East-West road with and without traffic lights. The color scale represents velocity.

5. Discussion and Conclusions

Traffic dynamics describes the interaction of vehicles, drivers and the environment. Traffic modeling can be used in a number of ways including determining optimal routes in navigation systems, planning and optimizing the operation of traffic lights, and even simulating the environmental effects of traffic operations with fuel consumption and carbon emissions.

The Nagel Schreckenberg model is one of the simplest models that are still used for traffic simulations. There are numerous other more complex methods of modeling traffic that may yield more realistic results, or consider other parameters. One of the major weaknesses of this model is that cars slow down in an unrealistic way. Oftentimes cars will drive up to a traffic light or traffic jam at full speed and then break immediately in one time step, whereas real drivers will likely slow down if they see a car in front of it. Modifications to the NS model could be implemented where a car knows the velocity of the car in front of it, and will adjust its own velocity accordingly. Another major weakness is that this model is for single lane traffic, and therefore does not allow overtaking of cars or lane changes. Also, this model does not allow collisions and does not allow any speed limit violations [2].

Our model was also representative of a simple situation. Turning was not implemented, and therefore the cars can only travel in a straight path, which of course, is unrealistic. In addition, only one intersection was considered, so only one traffic light cycle had to be coordinated, and there was never a buildup of cars on the other side of the traffic light since they were always free to leave the map. While our analyses focused on a single intersection, the majority of our simulation architecture is adaptable to multiple intersections. Therefore, by making some further adjustments, our simulation could be used to perform additional testing that expand the model to simulate traffic at multiple intersections.

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. <10.1051/jp1:1992277>.
2. Clarridge, A., & Salomaa, K. (2010). Analysis of a cellular automaton model for car traffic with a slow-to-stop rule. Theoretical Computer Science, 411(38-39), 3507-3515. Retrieved December 1, 2017, from https://doi.org/10.1016/j.tcs.2010.05.027.
3. Hoogendoorn, S., & Knoop, V. (n.d.). Traffic Flow Theory and Modeling. Retrieved December 3, 2017, from http://victorknoop.eS