Traffic Flow: Peachtree City Simulation

Dawar Ahmed, Robert Hill Pendley

Team 3

Code: https://github.gatech.edu/dahmed7/CX4230-Project-2

Cellular Automata Model

Problem Statement:

We are going to be investigating the average travel time for vehicles to traverse a portion of Peachtree Street, the corridor from 10th street to 14th street. We will focus on differing various conditions to find the distribution of travel times through the corridor. Using these various starting densities we will be able to see how the travel time is affected as a result. We will be adding small nuances to the model to make it more complex and more similar to the way a real work would function. It also adds to the stochasticity of model. We will changing the mean of the exponential distribution we are using to generate the interarrival times.

Conceptual Model:

This part of the project will be using cellular automata to model and simulate the project. This means our world consists of a world of open cells in a grid which can either be occupied (meaning a car is present) or unoccupied (meaning a car is not present).

Initially the grid is filled with 75% of cars where 75 is the starting density. These cars will later be eliminated from the results are their results are biased but they inputted so that the first added will have some resistance early on. Meaning the first cars model the actual road conditions

Our model is based in a simulated world which consists of a 2 x 128 Numpy array which will use as our backing for our cellular grid. The 2 represents the number of lanes on Peachtree Street and 128 represents the physical distance from 10th Street and Peachtree up to 14th. Using Google Maps we were able to determine the distance of Peachtree Street NE to be 1820ft. We also assumed that the average car length is 14.2ft. 128 is the actual physical distance divided by the average length of car, meaning there are 128 car sized cells from 10th street to 14th street. Traffic lights are also placed based on where they are placed along the road. Traffic Light 1 (11th Street) is placed 37 cells up to correspond to actual distance of 535ft. Traffic Light 2 (12th Street) is placed 35 cells up to correspond to the actual distance of 485ft. And Traffic Light 3 (14th Street) is placed a further 56 cells up to correspond to the actual distance of 800ft. Each Traffic Light is also using NGSim Signal Timings data to match up with the real life actual signal timings. For example Traffic Light (11th) remains green for 41 time intervals to match the 41.5s it remains green for in real life. And it remains red for 55 time intervals to 55.4s it remains red for in real life. Our model assumes one time interval or one iteration of the model is 1s.

Once we start the simulation, either two cars will be generated in a specified unit of time. The time between cars arriving (interarrival) will be based on the empirical data from the NGSIM data. This will be discussed later but it is based on the exponential distribution using the mean we calculated using the NGSIM.

Once the simulation begins the cars propagate forward with a given set of rules which we will discuss later. The cars also have the ability to change lanes and there is a random probability that a car breaks down which has initially been defined as 1/500. Cars may only the max velocity which we have defined as 3. Once we can run the simulation for a period of time we are able to determine data on how long each car stays in the system. Our model also adds in the probability that a car is considered a rash driver. Every car generated there is a 1/50 probability that a car will be a rash driver. A rash driver will

have a higher max speed (in this scenario 5) and a much higher chance of changing lanes when possible. This will be further discussed when we talk about our propagation rules.

Our model simplifies the changing lanes by only having the car move completely sideways as opposed to the more diagonal motion seen in real life. Also, our model assumes the area outside the visible world is empty. As in beyond the grid there are no cars and no speed restrictions which ultimately affects the simulation especially as you approach the end of the grid since the car may speed up even in high density traffic situation. We have also grossly oversimplified the two traffic lights. Currently the traffic lights only exist as either on or off (green or red) and each state is for 10 seconds. So, the simulation using a timer, switches between the green or red every ten seconds. In the real world you would also have amber in the traffic light and there is a set of completely different car movements which happen in that situation. Also, our traffic lights are arbitrarily offset by a couple of seconds just to make sure both weren't stopping and starting at the same time.

The values chosen for the stochasticity of the model (probability to change lanes, probability driver is a rash driver and probability of a car breaking down) all were chosen arbitrarily. While this may be seen negatively, there is no such literature available which would define these probabilities which we would be able to use. The numbers chosen were picked to seem reasonable and were used to make the model more realistic and stochastic.

To calculate the average time spent in the system, whenever a car is generate it is assigned the number referring to the count (current time). Once it leaves it calculates the total time spent in the system using the current time and adds to this to a list of times. Add the end the simulation calculates the average time spent in the system.

Our model assumes that each car is travelling straight and is only travelling straight. So, each car should have no intention of switching lanes (in order to turn) since its sole object is to move forward as quickly as possible. We also assume that there will be no car accidents, which is certainly not true on real life roads. Once a car breaks down, it doesn't move off the road or onto the side, it is assumed that the car will remain on the road and be fixed in a matter of time (5 time units to be precise).

The major limitation we have faced with our model so far is solving the issue of making the cars move in a natural way. Because of the world being limited to just what can be seen, there is a sense of the cars always following a very predictable behavior. The cars will start slow, increase their speed and eventually start moving at max velocity as they approach the end and rarely change lanes. I believe this is because of the free world at the end. So, cars have the ability to increase their velocity at the end of the road, meaning cars before can also increase velocity and thus there is no need for a car to ever change lanes. Another issue is the appearance of a convoy. Cars will at times move in tight packs with almost no space in between, which I don't believe is reflective of a real road. Cars in the real life would slowly spread out over time.

Specific Code Implementations:

1. Generating Cars. Here the method generates new cars, where *new_cars* is the number of cars to add each iteration

```
def generateCars(self):

    new_lanes = list(range(0, self.lanes))

    for i in range(0, self.new_cars):

        #randomly generate a lane for the car to enter
        new_lane = random.randint(0, len(new_lanes) - 1)
        new_lane = new_lanes[new_lane]
        new_lanes.remove(new_lane)
        #create new car
        new_car = car.Car(1, self.FILLED, new_lane, 0, self.roadLength)
        #add car to world
        #

        if (self.world[new_lane][0].exists == 0):
            self.world[new_lane][0] = new_car
            self.cars.append(new_car)
```

2. Propagate Forward. Here is the method where the cars move forward, and other elements are updated in each unit time. The code for this method can be found in the propogateForward method in world.py.
    a. First Traffic Light times are updated
    b. The times of blocked cars are updated and are unblocked is the time has elapsed
    c. Next cars may break time (become blocked cars)
    d. Then the cars follow the propagation rules which are as following:
        i. If car velocity is below max and gap until next car allows speeding up, increase speed by 1
        ii. If car velocity cannot be maintained because of the gap until next car is too small, look to change lanes
        iii. If gap until next car is too small and you cannot change lanes, reduce velocity to match the gap
    e. Remove any cars which have moved off the world
    f. Time is incremented for the data plotting purposes
3. Changing Lanes. Before a car can change lanes, it must follow a specific set of checks. When a car wants to change lanes, it investigates all possibly lane changes (left and right). In our case it would only ever check left when right and right when left, but this method can be further applied to a large road with 3+ lanes. These are rules if checks for:
    a. Check if a car exists in the potential new lane. If so, it cannot change
    b. Find the distance to previous car in the new lane and the next car in the new lane
    c. If there is no car behind and the current velocity can be maintained in the new lane based on the distance to the next car, then swap lanes
    d. If the previous car can maintain its velocity and the car can maintain its velocity in the new lanes, then swap lanes

This method is in the changeLanes function in world.py

All other methods aren't really part of the model but can be viewed on the GitHub repository.

Input Analysis:

To find the empirical distribution we used the data given in the NGSim trajectories file to calculate the distribution of interarrival times. We would reduced the data to ID, EPOCH, Org_Zone and Dest_Zone. We then created a script which would find the first instance of each valid car (entering in zone 101, since that is going north on peachtree and leaving 214 since that is leaving north on peachtree). We then find the time each enters the system for the first time. We would then sort this list of valid cars based on ascending enter times and then calculate the difference between these arrival times. We then generated a histogram, the mean and standard deviation. The mean came out to 12438.24 and the standard deviation is 16594.51. Based on the shape of histogram, the mean and the standard deviation we decided to instead of an empirical distribution use a exponential distribution. Our reasoning for this is explaining in the next model's writeup.

To change the flow rate, we would change the mean of the exponential distribution. This would affect the interarrival time and thus either add more cars in the system or less in a certain period

Validation:

The model is full of a lot of moving parts and it is quite hard to really talk about each individual component and how each component was tested but we'll talk about the process we took to test each process. We'll then examine some examples just to make it clear what we did.

The general process we took to examine each component was to first do a static code review together as a team. We made sure go through each individual component and make sure both members were comfortable with the code's syntax and its semantics. We would then conduct just a general run through of each components, printing out outputs and making sure each part was right. We would run through edge cases and every possible scenario we could think of. We also make sure to try and test each component in the most modular fashion possible. We would try to stub all previous values which were used in the coding process so we could know which aspect of the code specifically we were testing and didn't need to worry about other parts.

We'll now go through some specific examples

Output

Final Results:

Mean	5	10	12.438	15	20
Attempt 1	147.9	144.6	139.6	137.9	134.4
2	134.4	145.9	157.2	135.5	132.2

3	141.3	149.9	125.2	145.5	140.4
4	153.3	145.5	135.5	136.9	145.5
5	151.5	142.0	140.2	137.7	137.3
6	142.4	144.3	141.9	132.2	132.2
7	160.2	140.3	164.2	143.4	139.0
8	145.2	144.4	155.2	140.4	138.8
9	148.2	147.4	145.2	145.5	123.3
10	140.2	149.3	145.6	130.3	140.4
Average	146.2	145.36	144.98	138.53	136.35

Confidence Intervals:

Mean 5: 146.2 ± 6.82

Mean 10: 145.36 ± 6.82

Mean 12.438: 144.98 ± 6.82

Mean 15: 138.53 ± 6.82

Mean 20: 136.35 ± 6.82

Why we choose 10 repeats?

When we were investigating the results, 10 results seem to encompass all the different variations of the results and the average time plateaued. Also this was also we could do with time permitting.

Warm Up:

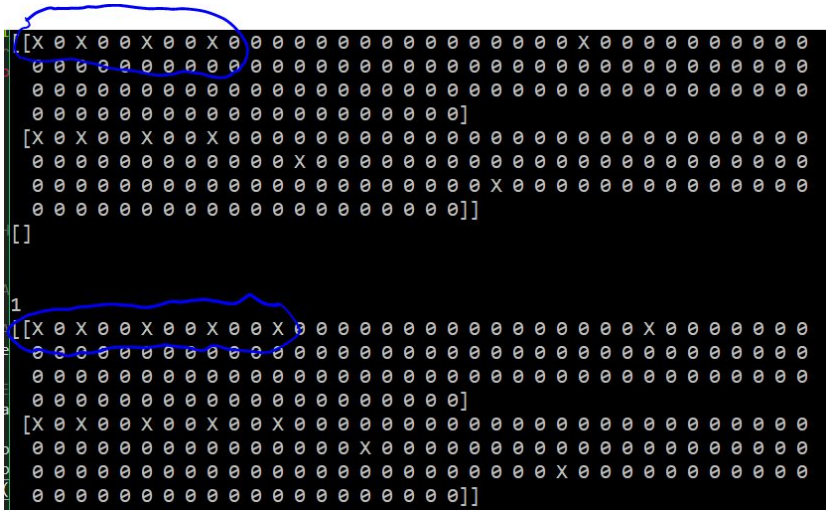
I choose a warm of about 75 cars, meaning the first 75 cars were removed from the model. This is because the first 75 cars were populated with when the system just started so there was nothing in front of them and thus the results were skewed for them. I choose this number because the system starts with approximately 75% cars so of those 75%, about 75 cars had a biased run (nothing ahead).

Comparison with Other Model:

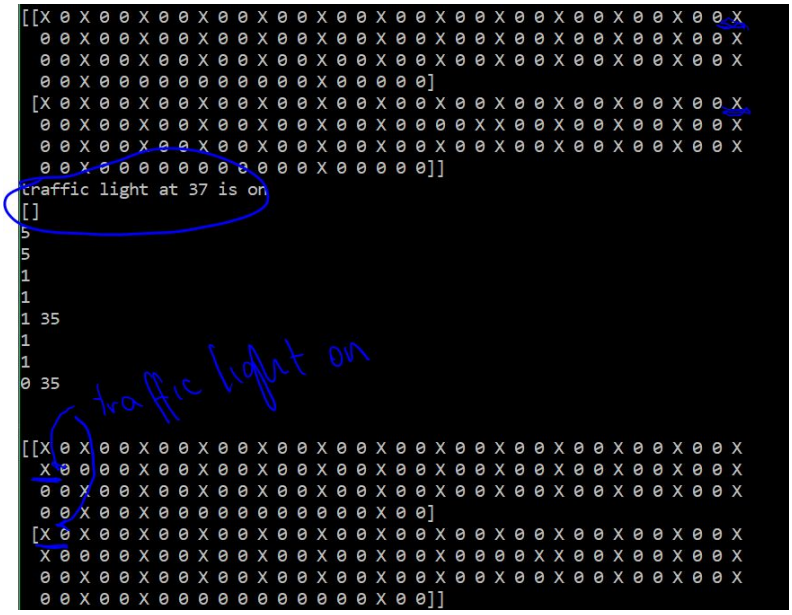
Just looking at the results we got pretty similar results. A few of the results were a little off, for example the average time with mean 10 however the differences were quite small which indicated that both models were accurate and valid.

Basic Propagation Forward

In the picture below we see two views of the world, back to back iterations. We here are visually testing the propagation of the cars. If we look at the cars highlighted in the blue circles we can see the cars moving in the correct fashion. In the above world, in the highlighted there are four cars. Moving from left to right, the first car just entered the system so it should have a velocity of 1, the second car will have a velocity of 2 and the third and fourth velocity will have a velocity of 3. Therefore the first car should move 2 spots (it can increase its velocity by one since the gap is 2, and thus will move 2 spots). The second, third and fourth will move 3 spots (since they have all achieved max velocity and will thus move three spots). We can see that each car is moving the correct amount in the second iteration of the world.



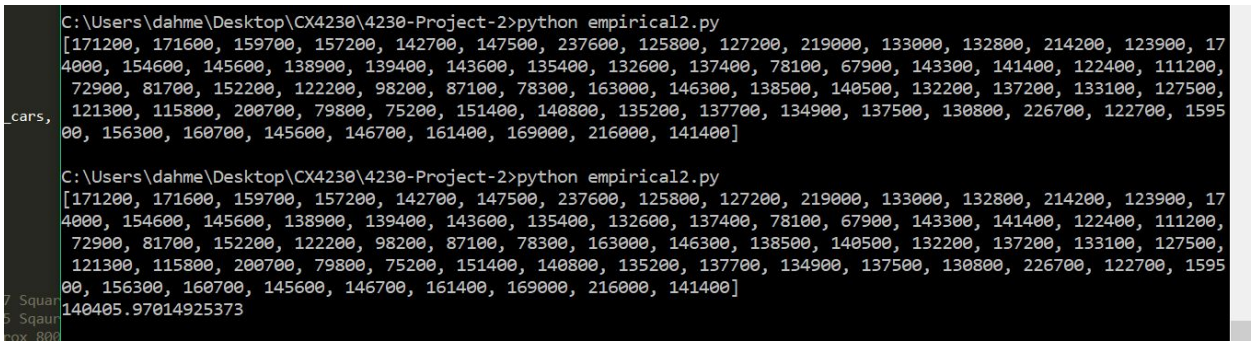
Traffic Lights Switching On



In this second example we can see, the traffic light at spot 37 has switched on after the top iteration. We can then see the next iteration below. The two blue arrows are indicating where the traffic light is. If we look at the highlighted cars in the first iteration, you can see that they have a gap of 3. This means that they can potentially travel up to 3 spaces, but instead only travel 1 space as seen below. This shows that they stopped at the traffic light and will wait for it to turn green again.

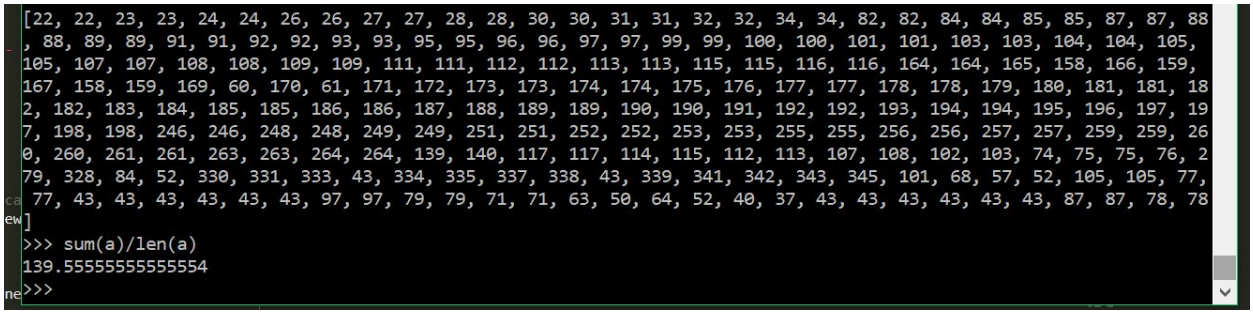
Verification:

To verify my model we needed to make sure our model matched the data given through the NGSIM data. To do this, we first reduced the data to the ID, EPOCH time and Org_Zone and Dest_Zone. We first weeded out any data which wasn't entering in zone 101 (facing north entering peachtree) and leaving 214 (leaving north). We then find the first time (entering) and last (leaving) to calculate total time spent in the system. We calculated this for each time and generated a average for the all the valid. This came out to about 140s. This was in the file **Empirical2.py**



It is the final number at the bottom of the above image.

To verify this model we calculated the average time spent in the system. We would calculate the average time spent in the system from all the data given to us. When accessing all the times of cars in the system we would delete the first 75 results, since these results were mostly from cars which were in the model to begin with and thus their motion was not in line with the model. We were able to get an average mean 144.98 which we felt was within the bounds for a valid model. Below is the times for one run.



Event-Oriented Queueing Model

Problem Statement:

The goal for the project is to simulate traffic in the stretch of Peachtree Street in between 10th and 14th street. The data to be analyzed is the average travel time for all vehicles depending on certain parameters. These parameters include stoplight greenlight length, stoplight greenlight frequency, and frequency of cars entering the system.

Conceptual Model:

The event-oriented queueing model contains separate software modules for the queueing network model (engine1.c) and the simulation logic(roadway.c). The queueing model contains the software needed to schedule events, run the simulation, and handle the list of future events. We used the necessary software from the example airport simulation engine that was provided to us by Dr. Fugimoto. Using this engine, we developed methods to implement a queueing model for the road to be simulated.

1) Assumptions made
   a. All cars are of the same speed and size
   b. Two cars cannot enter the simulation roadway at the same time.
   c. The car "enters" the simulation as soon as it crosses 10th street or turns onto Peachtree from 10th street.
   d. The car "exits" the simulation as soon as it reaches 14th street
   e. Assume that cars will not be affected by swerves in the road
   f. A car cannot exit the simulation at any time
   g. The stoplight timings of red, yellow, and green lights don't change
   h. Cars will travel through yellow lights
   i. The travel times between intersections are the same (10 seconds)

2) Details Concerning the model
   a. The number of cars to be simulated is the constant NARRIVALS
   b. The rate at which cars enter the simulation is determined by an exponential distribution with average ARRIVAL
   c. Each intersection has different timings for their lights
      i Stoplight 1 is green for 41 sec, yellow for 3.2 sec, and red for 55.4 sec
      ii Stoplight 2 is green for 61 sec, yellow for 3.2 sec, and red for 36 sec
      iii Stoplight 3 is green for 35 sec, yellow for 3.2 sec, and red for 44 sec

3) Simplifications made

a. Cars do not physically "move" distances. They simply queue at fixed points until they can move to the next point. The goal of the simulation is to develop methods and algorithms to determine when the cars are to "move" so that the simulation imitates real life.

b. We only modeled cars that were travelling north

c. We only modelled cars travelling on Peachtree Street from 10th street to 14th street

d. We only modelled one lane of cars (cars cannot pass each other)

Code Implementations:

1) Important Numbers

a. ARRIVAL is the mean time between each car being generated

b. TRAVELBETWEENLIGHTS is the time each car takes to get from one light to another

c. numEnters, numFirstMoves, numSecondMoves, and numExits is the number of times each event is processed

d. totalMinusTime is the time to be subtracted from the total time to get an accurate measure

e. totalTime is incremented by the current time whenever a car leaves the simulation, because of this, we must subtract the total time cars are waiting to enter the simulation

f. rand1, rand2, and rand3 are random numbers that determine the starting schedule for the stoplight

```
#define ARRIVAL 12.34
#define TRAVELBETWEENLIGHTS  10.0

// number of arrivals to be simulated (used to determine length of simulation run)
#define NARRIVALS   20

// State Variables of Simulation
int numEnters = 0;
int numFirstMoves = 0;
int numSecondMoves = 0;
int numExits = 0;
double minusTotalTime=0;  //used to subtract the time cars are not on the road
double totalTime = 0;       //used to find total time all the cars take to finish

int numReds1 = 0;       //used to make sure cars dont leave red lights at same time
int numReds2 = 0;
int numReds3 = 0;
```

```
#define rand1 (RandNum(99.0))       //used to start stoplight schedule at random numbers
#define rand2 (RandNum(100.0))
#define rand3 (RandNum(84.0))
```

2) Random number calculation

a. The second line of the method generates a random number, then divides it by the max random number + 1 to obtain a number between 0 and 1. Then log generates an exponential distribution with mean -1, and we multiply by -M to obtain a number with mean M. This code was obtained from Dr. Fujimoto's example project.

```
// Compute exponenitally distributed random number with mean M
double RandExp(double M)
{
    double urand;   // uniformly distributed random number [0,1)
    urand = ((double) rand ()) / (((double) RAND_MAX)+1.0); // avoid value 1.0
    return (-M * log(1.0-urand));
}
```

3) Entering the system

a. First the number of enters is increased

b. Then if there are more cars to be generated, another arrival is scheduled.

c. Then we schedule the arrived car at the second light

i We use rand1, and the current time to determine if the light is green/yellow or red

ii If it is green, the car will pass through the intersection and arrive at the next intersection in 10 seconds.

iii If the light is red, the car will wait till the light turns green plus three seconds for each car in front of it, to account for the time it takes a car to speed up

d. Then we update the time and free the memory pointed to by the parameter

```
// event handler for cars entering the system
void Enter (struct EventData *e)
{
    numEnters++;
    struct EventData *d;
    double ts;

    printf ("Car %d arrived at intersection 1: time=%f\n",numEnters, CurrentTime());

    // schedule next arrival event if this is not the last arrival
    if (numEnters < NARRIVALS) {
        if((d=malloc(sizeof(struct EventData)))==NULL) {fprintf(stderr, "malloc error\n"); exit(1);}
        d->EventType = ENTER;
        ts = CurrentTime() + RandExp(ARRIVAL);
        minusTotalTime += ts;
        Schedule (ts, d, (void *) Enter);
    }
```

```
    //schedule arrival at second light
    if ((d=malloc (sizeof(struct EventData))) == NULL) {fprintf(stderr, "malloc error\n"); exit(1);}
    d->EventType = MOVELIGHTS1;
    double t = fmod(CurrentTime() + rand1, 99.0);  //current time % 99
    if (t <= 44) {      //light is green
        ts = CurrentTime() + TRAVELBETWEENLIGHTS;   //car moves through light
    } else {            //light is red
        ts = CurrentTime() + (99-t) + TRAVELBETWEENLIGHTS + numReds1;
        numReds1+=3;  //cars leave red lights 3 seconds apart
    }
    Schedule (ts, d, (void *) MoveLights1);

    LastEventTime = CurrentTime();
    free (e);               // free storage for event parameters
}
```

4) Moving Intersections #1

a. This function moves the car from intersection 1 to intersection 2

b. First increase the number of moves

c. Then schedule the next event. Arrival at intersection 2. The logic for red vs green lights is similar to the logic in the Enter function, which can be viewed in part C of “Entering The System”. The only difference is the timing of the traffic light

d. Then update the time and free the memory pointed to by the parameter

```
// event handler for cars leaving intersection 1 and moving to intersection 2
void MoveLights1 (struct EventData *e)
{
    numFirstMoves++;
    struct EventData *d;
    double ts;

    printf ("Car %d moved to intersection 2: time=%f\n",numFirstMoves, CurrentTime());

    // schedule arrival at third light
    if ((d=malloc (sizeof(struct EventData))) == NULL) {fprintf(stderr, "malloc error\n"); exit(1);}
    d->EventType = MOVELIGHTS2;

    double urand;   // uniformly distributed random number [0,1)
    urand = ((double) rand ()) / (((double) RAND_MAX)+1.0); // avoid value 1.0

    double t = fmod(CurrentTime() + rand2, 100.0);  //current time % 100
    if (t <= 64) {      //light is green
        ts = CurrentTime() + TRAVELBETWEENLIGHTS;   //car moves through light
    } else {            //light is red
        ts = CurrentTime() + (100-t) + TRAVELBETWEENLIGHTS + numReds2;
        numReds2 +=3;
    }
    Schedule (ts, d, (void *) MoveLights2);

    LastEventTime = CurrentTime();      // time of last event processed
    free (e);               // event parameters
}
```

5) Moving Intersections #2

a. This function is very similar to the first “moving Intersections” function but instead of scheduling another move, it schedules an exit

```
// event handler for cars leaving intersection 2 and moving to intersection 3
void MoveLights2 (struct EventData *e)
{
    numSecondMoves++;
    struct EventData *d;
    double ts;

    printf("Car %d moved to intersection 3: time=%f\n",numSecondMoves, CurrentTime());

    // schedule leaving event
    if ((d=malloc (sizeof(struct EventData))) == NULL) {fprintf(stderr, "malloc error\n"); exit(1);}
    d->EventType = LEAVE;
    double t = fmod(CurrentTime() + rand3, 84.0);  //current time % 84
    if (t <= 38) {      //light is green
        ts = CurrentTime() + TRAVELBETWEENLIGHTS;   //car moves through light
        numReds3 = 0;
    } else {            //light is red
        ts = CurrentTime() + (84-t) + TRAVELBETWEENLIGHTS + numReds3;
        numReds3+=3;
    }
    Schedule (ts, d, (void *) Leave);

    LastEventTime = CurrentTime();      // time of last event processed
    free (e);               // event parameters
}
```

6) Leaving the Simulation

a. First increase the number of exit events

b. Then update the time and free the memory pointed to by the method parameters

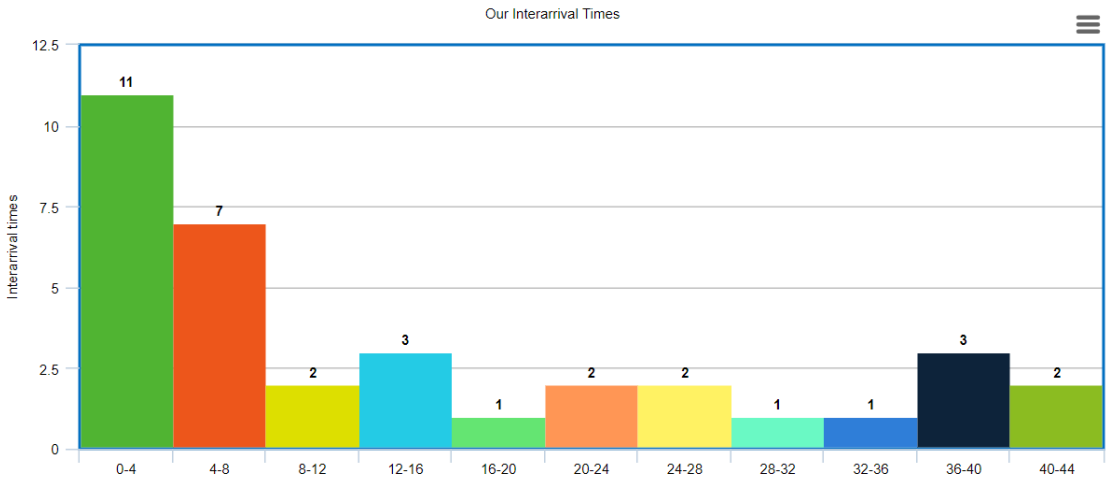
```
// event handler for cars leaving the system
void Leave (struct EventData *e)
{
    numExits++;
    printf ("Car %d exited the last intersection: time=%f\n", numExits, CurrentTime());

    LastEventTime = CurrentTime();      // time of last event processed
    free (e);               // event parameters
}
```

Input Analysis:

We modelled our simulation after the NGSim data set that was provided for us. After running a script to find useful data points like arrival times and exit times, we created graphs to plot these points. The graph of interarrival times was very obviously a quadratic distribution. Firstly, because it fit the shape of a logarithmic curve, and second because it had very similar mean and standard deviation. This is an important finding because quadratic distributions have the same mean and standard deviation when sampled many times.

From this information, we decided to use a function that computes an exponentially distributed random number to model the interarrival times between cars. This function takes one input which is the mean of the numbers generated. Using this input, we can change the flow rate of the incoming traffic to alter the simulation results. To validate this method of generating traffic flow, we decided to create our own histogram with a sample of size 35 from our random number generating function and compare it with the Data set.



Validation:

To verify our simulation, we want to check that the features of the simulation match the features in real life. The first thing that we validated were the interarrival times. We looked at the interarrival times of the NGSim data set and determined them to match an exponential distribution. Then we found the mean of this data set (12.44 s). To maximize the validity of the simulation, one should generate the interarrival times with mean of 12.44.

We also validated our system by making sure that the cycles of each traffic light exactly matched the data that we were given. Assuming a car will drive through yellow lights, this means our simulation exactly matches the section of Peachtree street. Also we made sure that the time it takes cars to travel between lights accurately reflects the data gathered from the actual length of road, about 25 seconds.

Verification and Output Analysis:

Here is a sample of what the simulation outputs when only 5 cars are generated. As you can see, evert time an event is processes, the car, the intersection, and the current time is outputted. The "random" numbers are the initial seeds for each stoplight when we start the simulation. At the end of the simulation, the total time that all cars spend in the simulation is printed. The average travel time is found by dividing this number by the number of cars generated. The reason for all of these printed outputs is to verify that the simulation is working along every step of the way.

```
Peachtree Street Simulation
random1 =44.787728
Car 1 arrived at intersection 1: time=13.544761
Car 2 arrived at intersection 1: time=31.885774
Car 3 arrived at intersection 1: time=51.371456
Car 4 arrived at intersection 1: time=59.480239
random2 =75.349340
Car 1 moved to intersection 2: time=64.212272
Car 2 moved to intersection 2: time=67.212272
Car 3 moved to intersection 2: time=69.480239
Car 4 moved to intersection 2: time=70.212272
random 3 = 75.780897
Car 1 moved to intersection 3: time=74.212272
Car 5 arrived at intersection 1: time=75.206705
Car 2 moved to intersection 3: time=77.212272
Car 3 moved to intersection 3: time=79.480239
Car 4 moved to intersection 3: time=80.212272
Car 5 moved to intersection 2: time=85.206705
Car 5 moved to intersection 3: time=95.206705
Car 1 exited the last intersection: time=102.219103
Car 2 exited the last intersection: time=105.206705
Car 3 exited the last intersection: time=105.219103
Car 4 exited the last intersection: time=108.219103
Car 5 exited the last intersection: time=111.219103
Number of cars = 5
20 events executed in 0.000104 seconds (192307.692308 events per second)
Total time =314.138942
Average travel time for each car = 62.827788
```

To find some statistics for our simulation, we decided to run the simulation multiple times and plot data points to create a distribution. We decided to see how the average travel time is affected when the flow rate of cars entering the system is changed. For each flow rate, we will generate 40 cars on the system and view the average travel time for all of the cars. We will view how the travel time is affected when we have mean flow rates of 5, 10, 15, and 20 seconds. From the chart below, it seems that as the mean interarrival time increases, the average travel time decreases at. This is an expected result, since a denser flow rate should result in more cars becoming stuck behind each other at red

lights. We chose 10 because this sample size will give us enough data points to balance out any outliers that might have appeared in the data.

Mean arrival times	5 sec	10 sec	12.44	15 sec	20 sec
1st	111.2	139.1	134.3	117.3	150.2
2nd	142.2	122.5	119.8	113.1	108.7
3rd	163.1	126.4	117.6	121.6	118.9
4th	149.5	163	128.5	137.2	123.7
5th	146.9	127.1	157.1	142.6	139.4
6th	103.5	130.6	133.9	126.6	115.1
7th	158	132.4	122.8	183.8	155.8
8th	191.7	152.1	147	128.7	107.4
9th	149.4	124	131.8	123	120.8
10th	164.1	125.9	124.4	135	128.3
Mean travel Time:	147.96	134.31	131.72	132.57	126.82

Confidence intervals:

Using the mean and standard distribution of the sample above, we were able to create three confidence intervals

95% -> 129 to 143

90% -> 130 to 142

80% -> 132 to 140

The simulation does multiple things to avoid inaccurate results due to a warmup period. The first way in which we combat the warmup period is by seeding the cycles of the traffic lights so that the traffic lights start at random places in their cycles. This is helpful because it eliminates the need to wait for the traffic lights to get out of sync with each other. The second way that we can deal with the warmup period is by only taking statistics for the last half of the cars that run through the system. This will eliminate any possibility of skewed data. The cars that enter the system first will travel through the simulation faster because they will not need to wait on as many cars in front of them after a light turns green.