Application of Monte Carlo Methods in Traffic Simulation using the Nagel-Schreckenberg Model

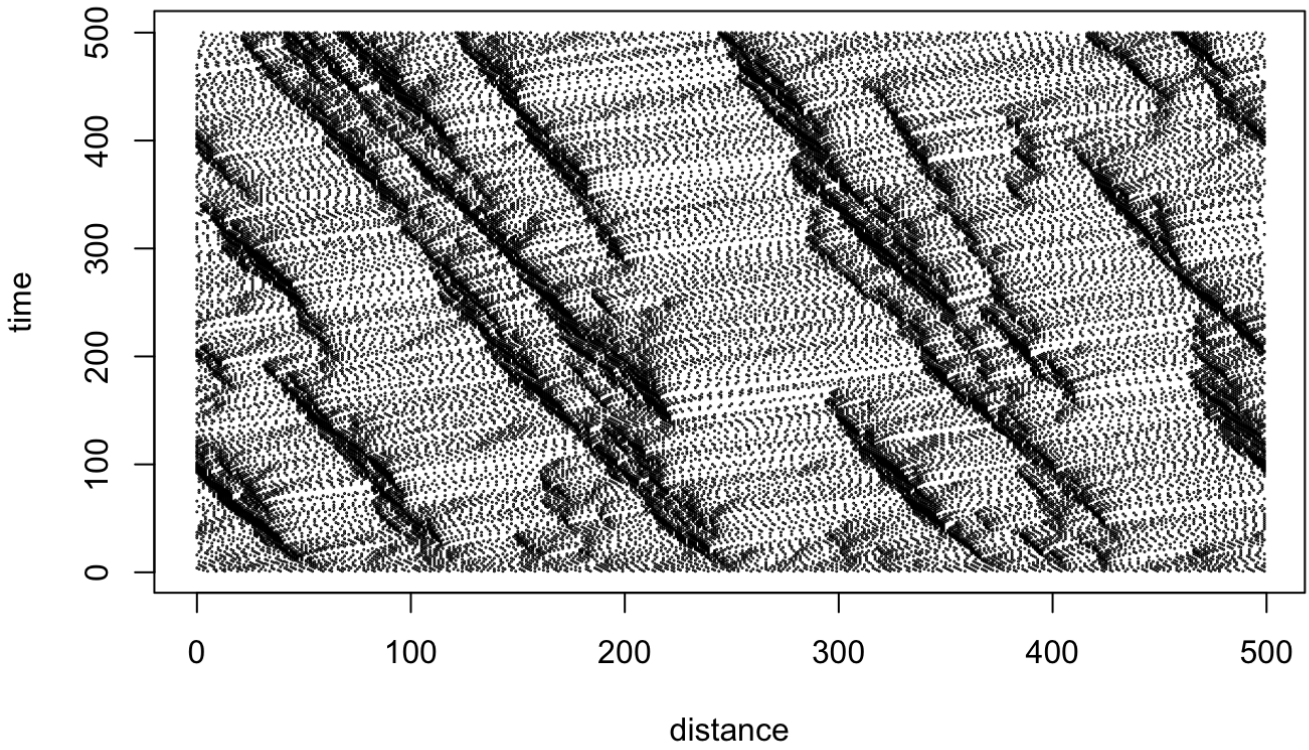
2022-06-04

Group number: 4

name: Jiayang Zhao net_id: jiayang_zhao [GROUP LEADER]

name: Christopher LaBorde net_id: claborde

name: Tianchen Dong net_id: tianchendong



Visualization of Traffic Flow by Monte Carlo Simulation

Abstract

In the modern era, the number of cars running on the road increases dramatically because of the mass production of vehicles. Consequently, more cars lead to more traffic congestion. Finding a proper prediction of the traffic becomes one of the priorities of the administration department. Our team uses the Nagel-Schreckenberg traffic model and Monte Carlo method to simulate the impact of an increasing number of cars on a road. We produce a visualization of the traffic flow so that traffic jams can be seen on the map. Moreover, we successfully find an optimal number of cars a road should have. Our results can be used for government planning of roads and traffic flow control.

Introduction

With ever-increasing global urbanization, traffic congestion affects more and more lives with each passing year. Traffic congestion results in losses of billions (yes, billions) of dollars per year. A study done by the American Transportation Research Institute estimates that in the US alone, roughly 74 billion dollars of value are lost annually due to traffic congestion (https://www.statista.com/chart/21085/annual-economic-losses-from-traffic-congestion/ (https://www.statista.com/chart/21085/annual-economic-losses-from-traffic-congestion/)). For this reason, the simulation of traffic is a problem that is worth discussing and investigating. Traffic Flow analysis has applications for city and architectural planning, road construction, and resource and safety management. Higher-fidelity traffic models lead to better planning, better foresight, better implementation, and reduced loss in value due to time and energy wasted in traffic jams. Thus, there is motivation to develop effective traffic congestion models in order to mitigate the economic losses experienced on crowded roads.

In the early 1990's, german physicists Kai Nagel and Michael Schreckenberg developed a model to simulate a simple traffic system. At its simplest, it is a simulation of a number of cars on a circular, one-lane track. The track is divided into a number of cells which are either occupied or unoccupied by a car. There are certain parameters and rules that constrain the simulation environment. There is a maximum velocity (analogous to a speed limit). There is a set amount of cars on the road, and there is a probability of random deceleration for each car. The simulation is run

over discrete time steps. The cars gradually increase their velocity, a car cannot pass the car in front of it, and cars reduce their velocity both at random and as they approach a car in front of it. Each car has several values associated with it, namely its velocity, its position, and its distance from the car behind it.

Because of the randomness of human behavior, it is normally hard to simulate traffic without massive amounts of computation. The Nagel-Schreckenberg model and Monte Carlo simulation enables us to simulate traffic in a relatively efficient way. The Nagel-Schreckenberg model is praised for its simplicity while simultaneously being able to model traffic flow relatively effectively. There are however shortcomings to the model. Because of its simplicity, the model does not take into account phenomena that are common in real-world road systems. These include multi-lane interactions, the changing of the number of cars on the road, the existence of pedestrians, and so on and so forth. The model can be modified to simulate these conditions, but is beyond the scope of this investigation.

Given more time, we would develop the model further, in an attempt to implement more complex interactions as previously described. Our next target would be the implementation of multi-lane interactions within the simulation.

Methods

We use NaSch model and Monte Carlo simulation to simulate the cars running on the road. We assume that the number of cars is constant in the whole simulation. In addition, we assume that the cars are running in a circle so that we can see the traffic congestion more easily.

In this project, the Nagel-Schreckenberg model was implemented in order to visualize the spontaneous emergence of traffic phenomena in a closed-loop system. We specifically observed the compression of space between the “cars”, and the spontaneous emergence of a traffic jam due to random behavior by each of the cars.

A simulation was run in the RStudio environment, implementing the Nagel-Schreckenberg model. Three equal-length matrices were created, each containing specific information about a car. Each index within the matrices held information corresponding to one car. We had a position matrix, a velocity matrix, and a distance-from-the-last-car matrix. The matrices were updated with each iteration, to reflect the progression of the simulation.

To simulate road-like behavior, a set of rules were programmed into the simulation. In our primary simulation, the cars were spaced equidistantly on the track, and given an initial velocity equal to the maximum velocity allowed. The rules programmed were:

1. During each time step, a car increases its velocity by 1, unless it is at maximum velocity.
2. If a vehicle’s velocity is greater than the distance to the next car, decrease the velocity of the vehicle by 1
3. With probability p, each vehicle may slow down by one velocity unit.
4. Update the vehicle’s position

Expressed mathematically, the rules follow

1. $v_t \leftarrow \min(v_{t-1} + 1, v_{\max})$
2. $v_t \leftarrow \min(v_t, d - 1)$
3. $v_t \leftarrow (v_t - 1)$ with probability p
4. $x_{t+1} \leftarrow x_t + v_t$

Step 3 is the step in the process out of which the variations in spacing emerge. The randomness of deceleration across the cars is what causes the compression in spacing, and without this step, the process would be deterministic. It is also the step which gives this model its characteristic of a Monte Carlo simulation.

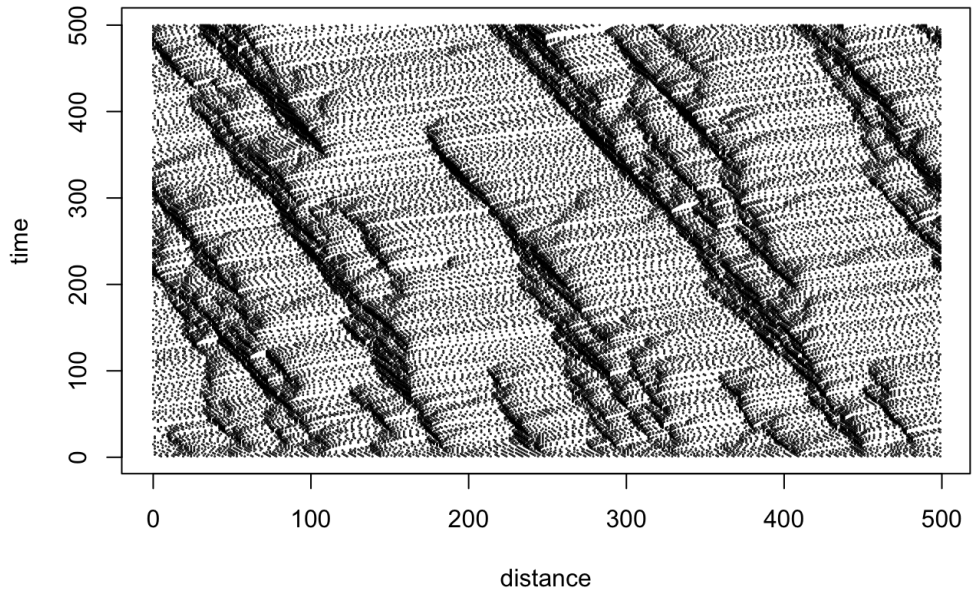
After running this simulation for t time steps, a graph is plotted with the cars’ positions for a range of time steps, by default 1 to t with the positions on the x-axis and the time steps counted on the y-axis. This graph shows the “flow” of the vehicles, and allows us to visualize the emergence of traffic phenomena over time.

Next, the total distance traveled by all cars within the model is summed up, and compared for different values for the number of cars in the system. The sums of distances traveled by all cars is plotted on a graph, with distance on the y-axis and number of cars in the system on the x-axis. The maximum of the curve shows the optimal density of cars on the road, where the most amount of vehicles travels in the most efficient manner.

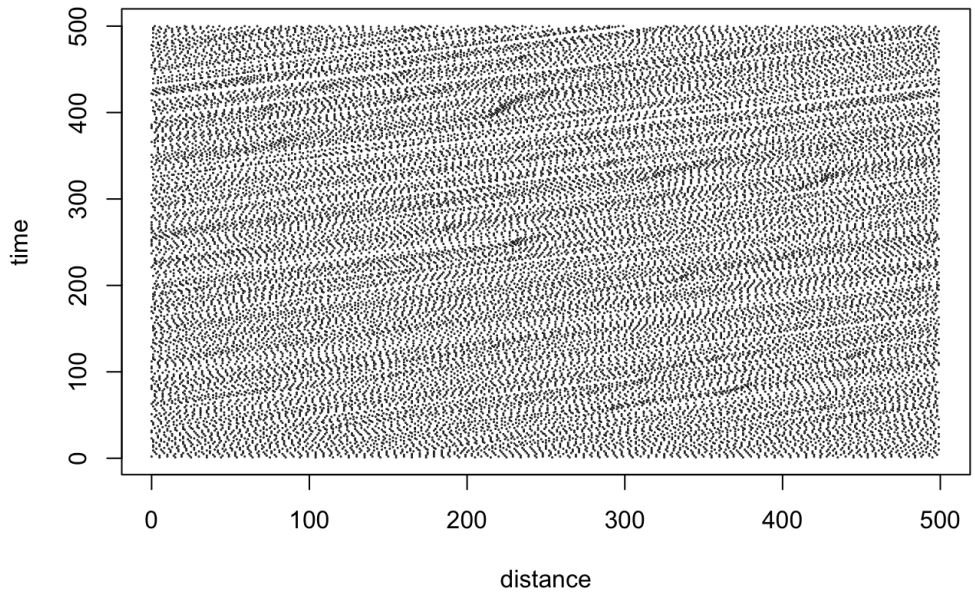
Varying the parameters such as the number of vehicles on the road, the probability of random deceleration and the maximum velocity allows us to examine the behavior of the aggregate system under different constraints, representing varying conditions in a real-world scenario.

Results

The outcome of the simulation was promising. After running the simulation described above and adjusting values of the parameters, a graph of the positions of the vehicles for all time steps was plotted:



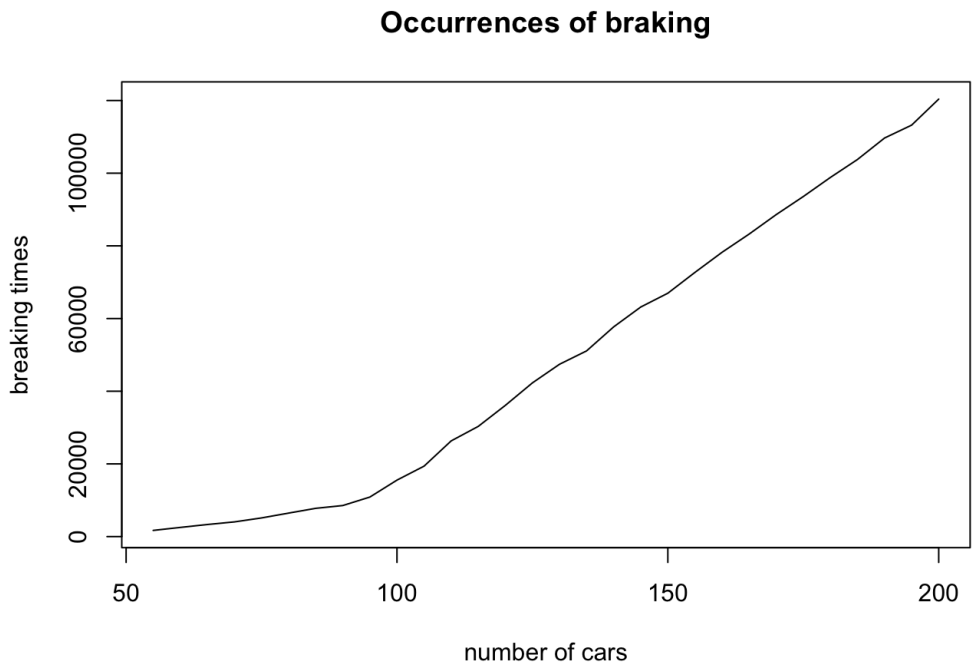
A simulation with the parameters:
car_num = 100
v_max = 5
distance = 500
p = 1/3
t = 500



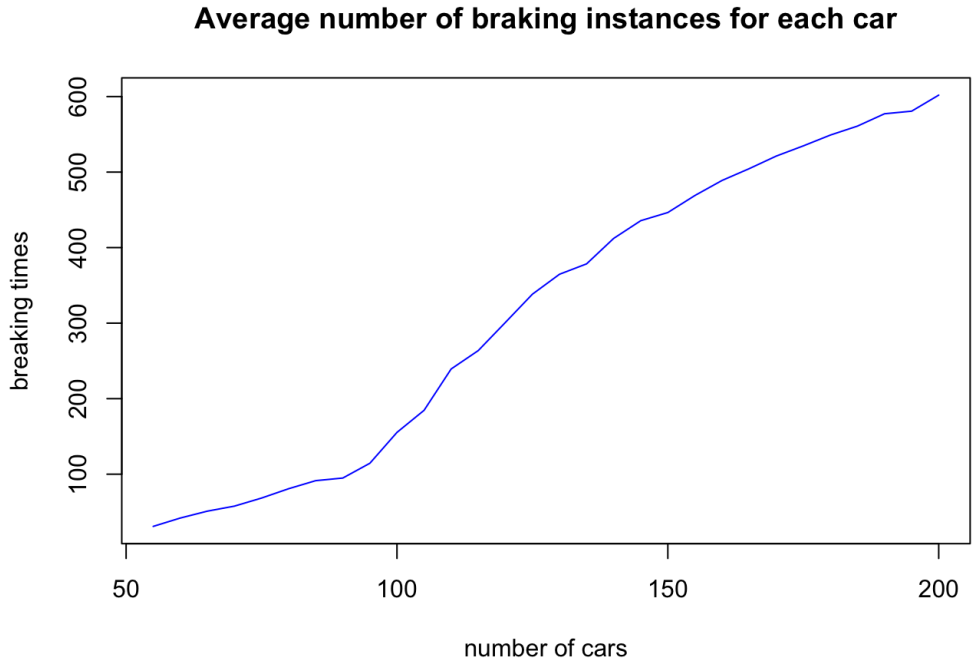
All parameter values held constant except for car_num, which is reduced to a value of 50.
car_num = 50
v_max = 5
distance = 500
p = 1/3
t = 500

The graph shows some remarkable features. Namely, compression waves (“traffic jams”) are observed emerging randomly throughout the track, persisting, then either growing or spontaneously disappearing. This is interesting, since this very simple model helps us to visualize emergent phenomena with a very limited set of constraints. We then continued with an analysis of the simulations.

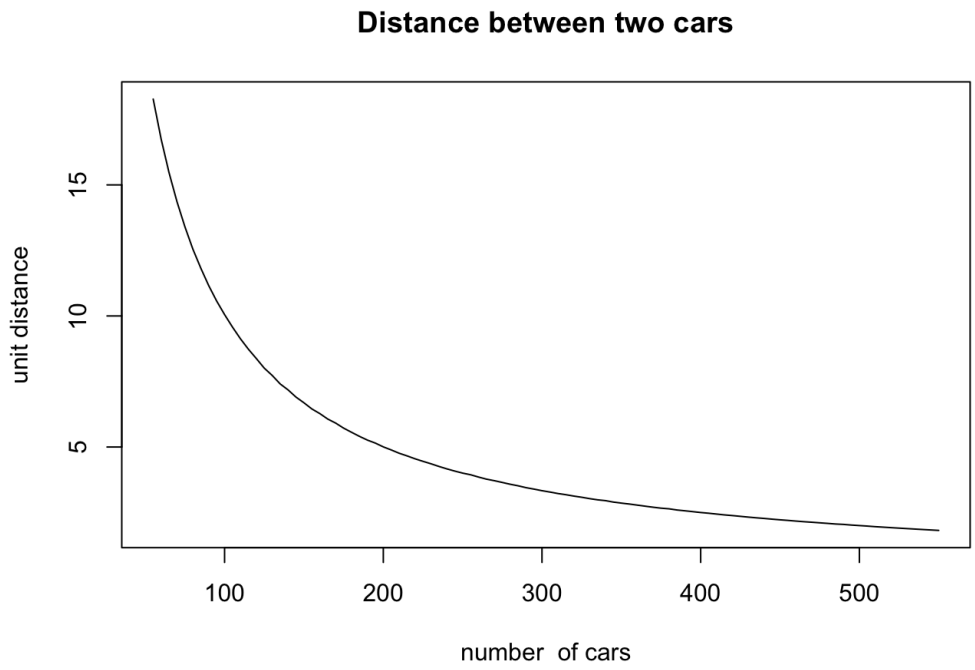
In the analysis of the simulations, we examined the number of cars on the road vs. the number of total braking instances, the average distances between cars, and the distances traveled per car. Below are the graphs produced in the analysis:



This graph is the number of cars running on the road versus the total number of break times. As we can see, it shows that the braking time increases as the number of cars increases. We can also see from the plot that when the car number is less than 100, the curve has a relatively flat slope.

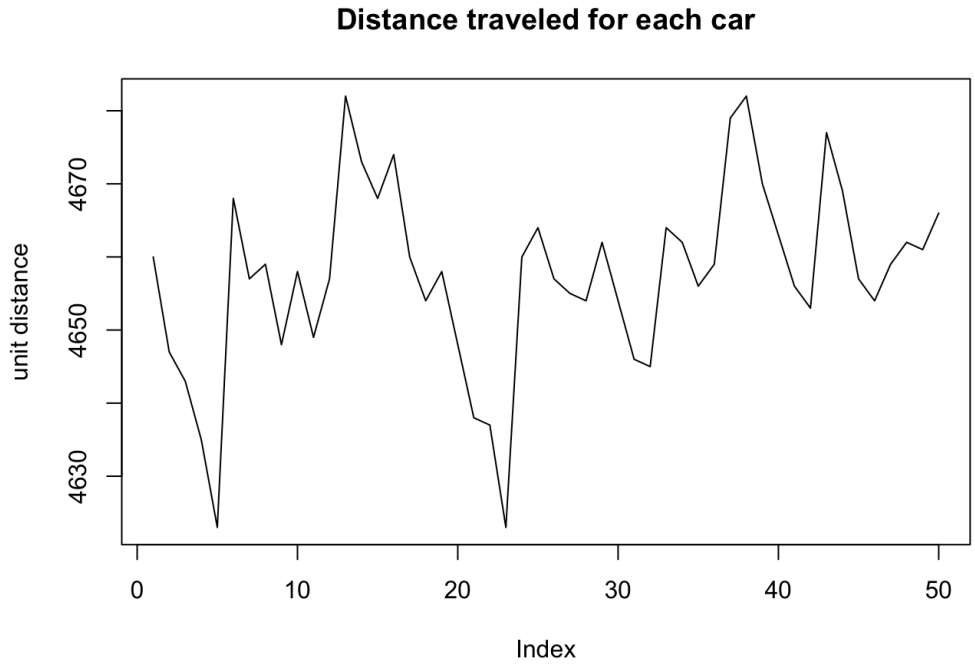


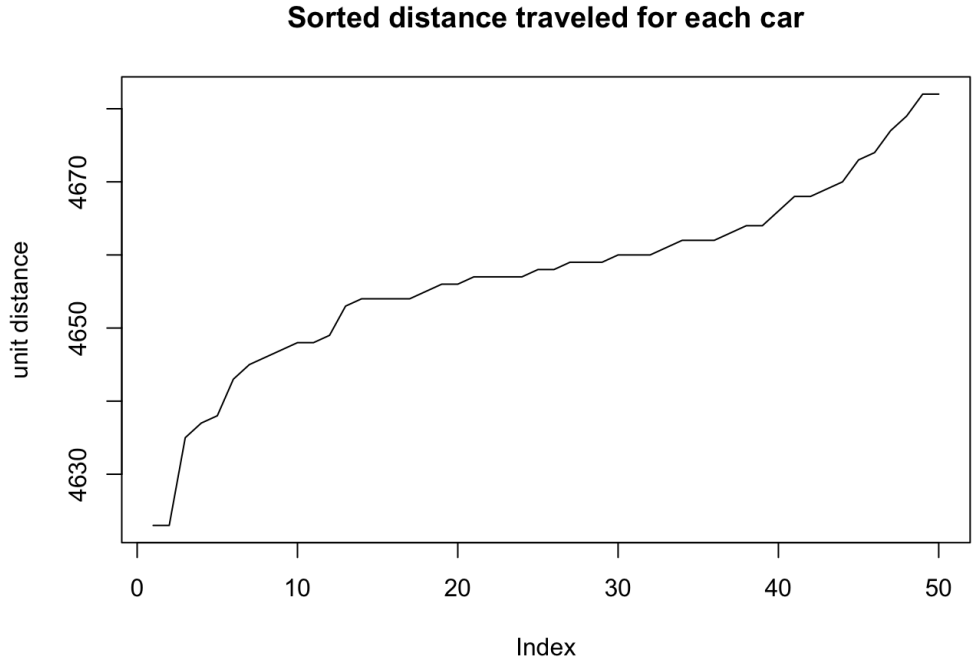
We also produce the graph of the number of cars running on the road versus the average number of break times. The curve shown in the graph is similar to the last one. But still, we can see that the curve has a low slope when the total number of cars is low.



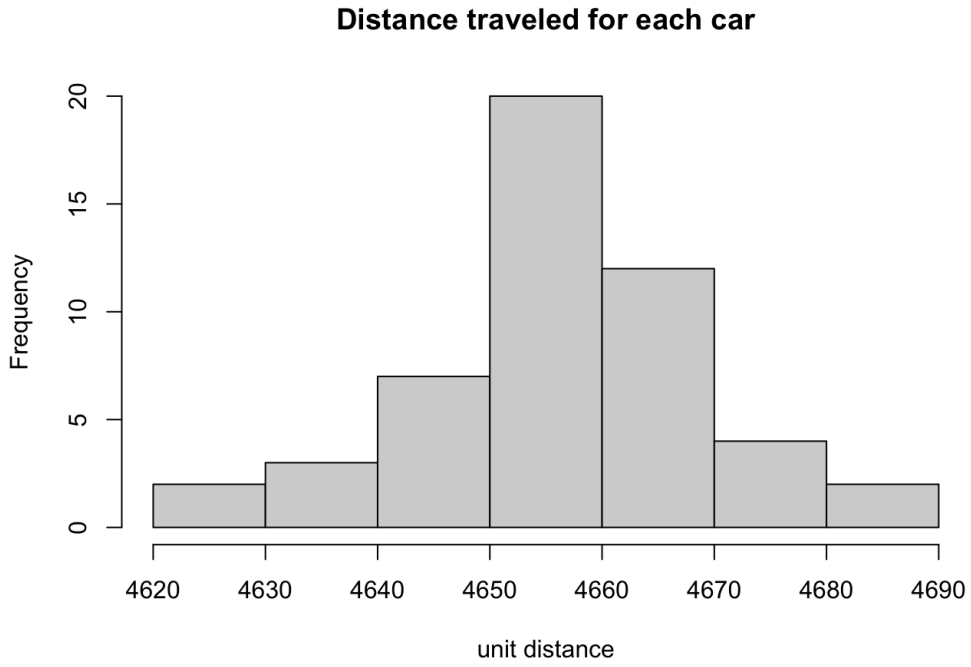
This is the plot of the average distance between cars. In the graph, we can see an exponential-like curve. The ratio between the number of cars and the average distance between cars is not fixed. This means that we can probably find an optimal point where we have the most number of cars possible and the average distance between cars is as big as possible.

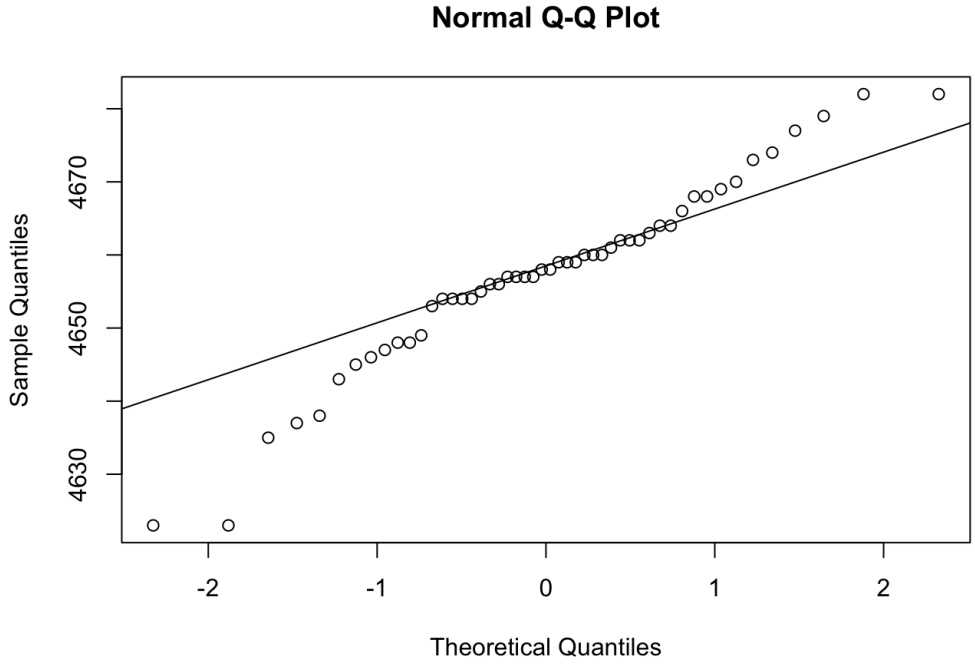
Lastly, we investigated the relationship between the number of cars and the distance traveled.





This is a plot showing the distance traveled by different cars. The distribution is pretty random. We can't conclude that certain cars can travel further than others. However, after we sort it, we can see the graph become something like this. We then start to analyze the possible distribution of it. By using histogram and ggplot (shown below), we find that the distribution of the distance traveled by the cars is normal distribution.



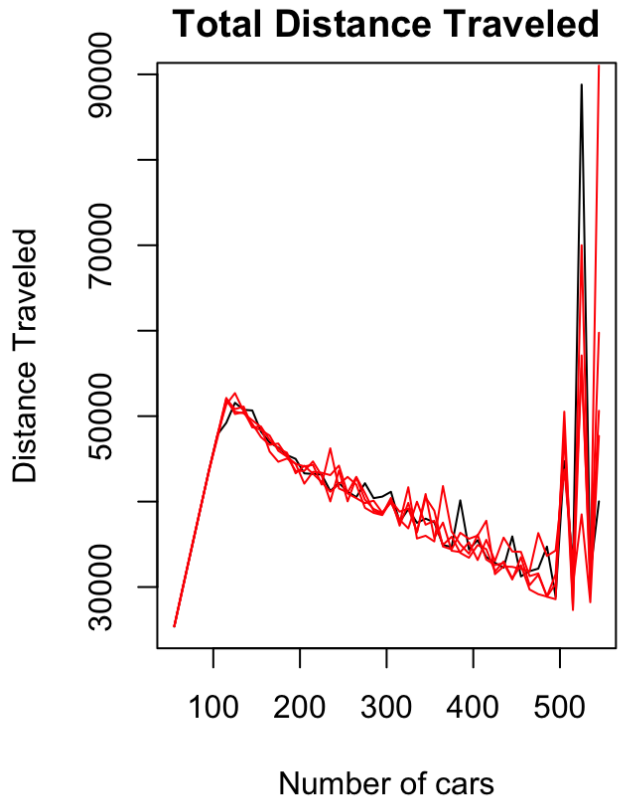


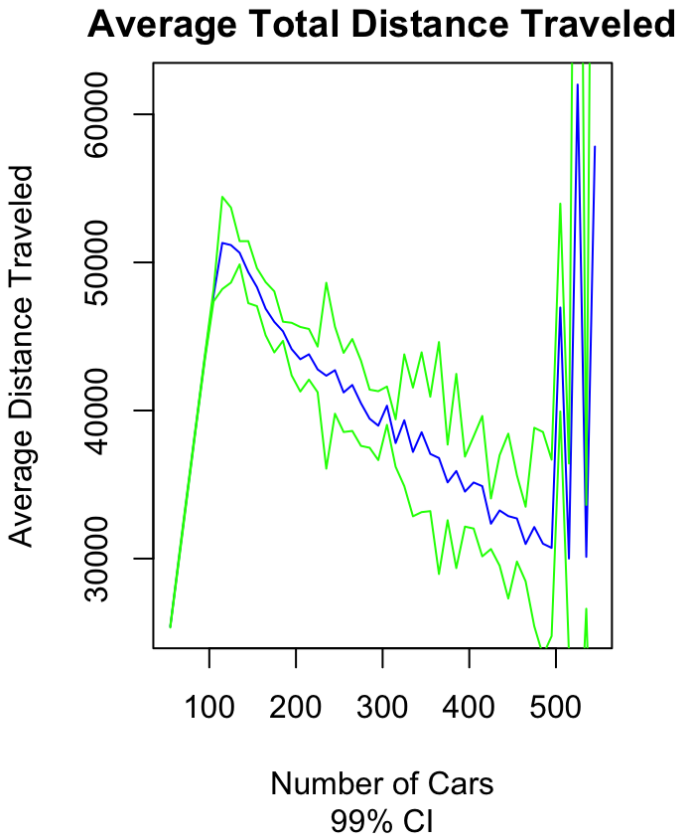

```
shapiro.test(sim_sort)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  sim_sort
## W = 0.95917, p-value = 0.08205
```

```
# ks.test(sim_sort, 'pnorm')
```

As another part of the analysis, we plotted the fundamental diagrams for multiple iterations of the simulation



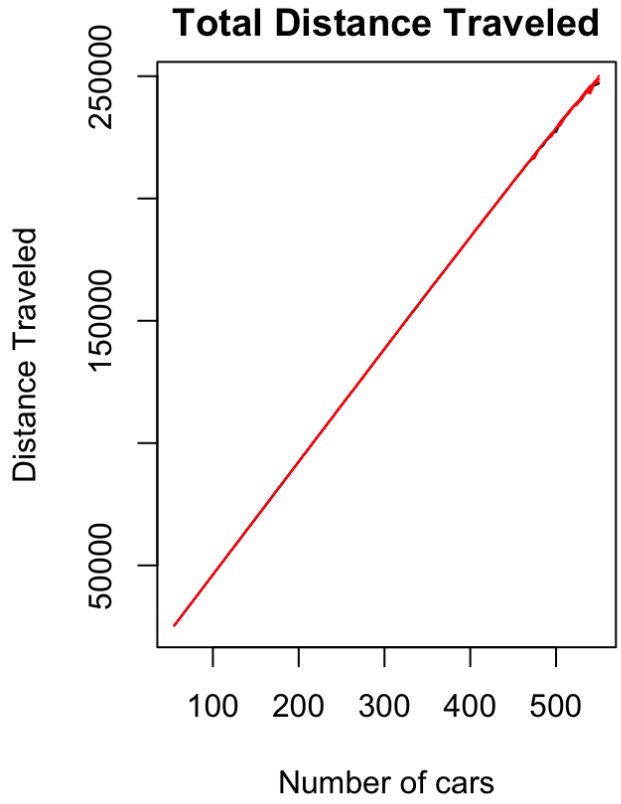


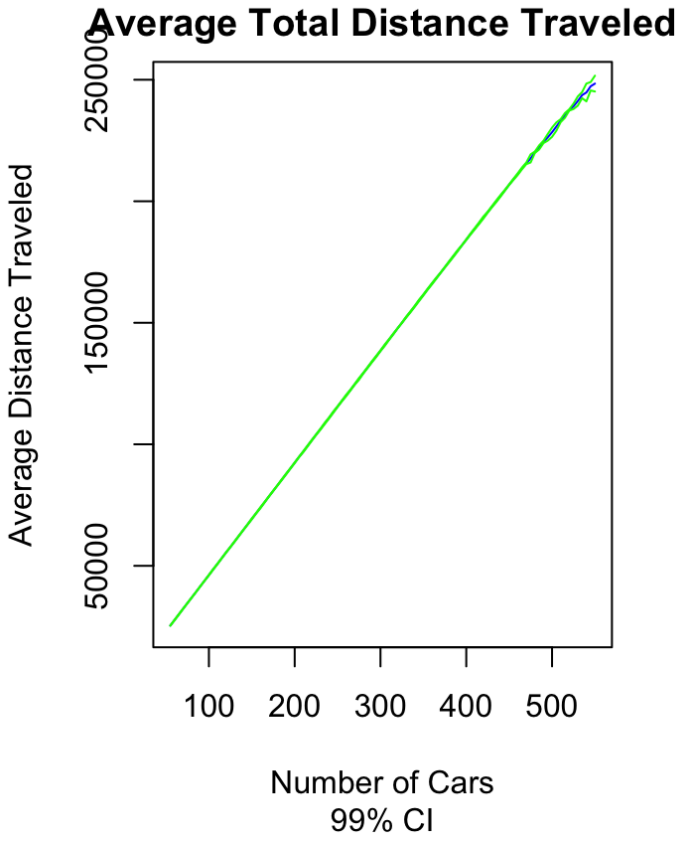
These are the fundamental diagrams for 5 iterations of the simulation, with the parameter values as shown above. The mean is plotted in blue on the right hand graph, with 99% upper and lower confidence bounds plotted in green. These fundamental diagrams show that the optimum number of cars in the system to maximize efficiency is somewhere around car_num = 120.

As of this writing, it is unknown why the distance traveled jumps significantly when the number of cars > 500. This is likely an artifact of the code.

We find that because of the randomness of human action, it is inevitable for traffic congestion to occur. However, we also find that with good control of the number of cars, we can reach an optimal situation where we have the most number of cars and they travel the distance as far as possible.

#





There is no downturn of total distance traveled because the track of length 5000 is not crowded enough for cars to hinder each others' movement with a maximum of 550 cars.

Discussion

The result of the traffic simulation model we used can be interpreted as the distribution of cars on a single-lane infinitely long highway in real life, and showed why there will be randomly produced traffic jams produced by randomly slowing down of the cars even though there might be enough space between them.

From the model, we get two useful pieces of data with certain parameters given. The first one is the total distance traveled of all cars against the number of cars. This gives us the optimal number of cars on the road. As in the example, the theoretical optimal number of cars is d = 1000 / v_max = 5 = 200 if there is no randomly slow down at all. But we got the optimal number which is around 120. This confirms the results of the model. So, from the model, someone who has access to the data of total number of cars on a certain road can forecast if there will be a traffic jam in the future and decide whether to widen the road. Or instead, one can use the existing data to estimate the slow down probability parameter on certain roads, and give analysis on that. Another piece of data is the time of breaks of all cars against the number of cars, of which the breaks do not include those random ones. Since usually a break will lead to extra cost of gas for the car. One can produce an estimation of gas cost on a certain road with this information.

The biggest limitation of this model is that only one-road is considered. So, to improve this, we first need to create a model that allows the realization of lane-changing. Also, we can add more parameters rather than only the randomly slowing down parameter, such as an accident that takes more time to be removed from the road, or simply a traffic light. And for the slowing down parameter itself, maybe we can try to randomize it since not every driver is the same. Since for now we observe the distance traveled of each car is possibly following a normal distribution, but it may change if the parameters are more complex and randomized.

References

- Art Owen, Monte Carlo theory, methods and examples, in progress (https://artowen.su.domains/mc/Ch-intro.pdf)
- Prof. Joseph Kasprzyk, Statistical Approaches for Simulation Modeling, CVEN 5393: Water Resources Systems and Management (https://www.colorado.edu/lab/krg/sites/default/files/attached-files/5393_06_statistical-approaches-for-sim-model.pdf)
- Ibna Rahman, Faysal. (2018). SHORT TERM TRAFFIC FLOW PREDICTION BY MONTE CARLO SIMULATION. 18. 26-33. (https://www.researchgate.net/publication/326958648_SHORT_TERM_TRAFFIC_FLOW_PREDICTION_BY_MONTE_CARLO_SIMULATION)

Code Appendix

```
# Traffic simulation goes through t time steps updating a vehicles speed, position and distance traveled

traffic_simulation<- function(car_num,v_max,distance,p, t){
  v_vector = rep(v_max,car_num) ### 0 and v_max
  d_vector = rep(distance/car_num,car_num)
  p_vector = seq(distance/car_num, distance, by = distance/car_num)

  result = data.frame(matrix(NA,nrow = t, ncol = car_num))
  time = 1
  while(time <= t){
    for(i in 1:car_num){
      #algorithm start
      v_vector[i] = min(v_vector[i]+1,v_max) # increase speed by 1 until max velocity
      if(i == car_num){
        v_vector[i] = min(v_vector[i],d_vector[1]-1) # car in front reduce speed to loop around to beginning (so does not over
take first car)
      }
      else{
        v_vector[i] = min(v_vector[i],d_vector[i+1]-1) # slow down vehicle so it does not overtake car in front
      }

      u = runif(1,0,1)
      if(u<p){
        v_vector[i] = max(0,v_vector[i]-1) # reduce vehicle velocity randomly with probability p
      }

      p_vector[i] = p_vector[i]+v_vector[i] # update position

      if(p_vector[i] >= distance){
        p_vector[i] = p_vector[i] - distance # if vehicle reaches end, loop around to beginning
      }

      #update distance between car i and car i-1
      if(i != 1){
        if(p_vector[i] - p_vector[i-1] > 0){
          d_vector[i] = p_vector[i] - p_vector[i-1]
        }else{
          d_vector[i] = p_vector[i] + distance - p_vector[i-1]
        }

      }else{
        #for the first car
        if(p_vector[i] - p_vector[car_num] > 0){
          d_vector[i] = p_vector[i] - p_vector[car_num]
        }else{
          d_vector[i] = p_vector[i] + distance - p_vector[car_num]
        }
      }
    }
    #print(p_vector)
    result[time,] = p_vector
    time = time + 1
  }
  return(result)
}
```

```
plot_car_flow <- function(sim_data){  # plot function
  t <- nrow(sim_data)
  car_num <- ncol(sim_data)

  plot(sim_data[,1], 1:t, type = "p", pch = 20, cex = 0.1,
       xlab = "distance", ylab = "time")

  for(i in 2:car_num){
    points(sim_data[,i], 1:t, pch = 20, cex = 0.1)
  }
}
```

```
car_num = 70
v_max = 5
distance = 500
p = 1/3
t = 500

sim_1 <- traffic_simulation(car_num,v_max,distance,p, t)

plot_car_flow(sim_1)
```

```
# calculate total distance traveled of one car
calculate_distance <- function(car_data, d){
  r <- 0
  for( i in 2:length(car_data)){
    r <- r + as.numeric(car_data[i] < car_data[i-1])
  }
  final_pos <- car_data[length(car_data)] + r * d
  dist_travel <- final_pos - car_data[1]
  return(dist_travel)
}
```

```
car_num = seq(55,550,5)
v_max = 5
distance = 1000
p = 1/3
t = 100


# BEGIN ITERATION 1

sims <- list()
total_dist_list <- list()


for(i in 1:length(car_num)){
  sims[[i]] <- traffic_simulation(car_num = car_num[i], v_max,
                                  distance, p, t)
}

for(i in 1:length(sims)){
  total_dist_list[[i]] <- apply(sims[[i]], MARGIN = 2, calculate_distance, d = distance)
}

ySim <- sapply(total_dist_list, sum)


# ITERATION 2
sims1 <- list()
total_dist_list1 <- list()


for(i in 1:length(car_num)){
  sims1[[i]] <- traffic_simulation(car_num = car_num[i], v_max,
                                  distance, p, t)
}

for(i in 1:length(sims1)){
  total_dist_list1[[i]] <- apply(sims1[[i]], MARGIN = 2, calculate_distance, d = distance)
}


ySim1 <- sapply(total_dist_list1, sum)



# ITERATION 3
sims2 <- list()
total_dist_list2 <- list()


for(i in 1:length(car_num)){
  sims2[[i]] <- traffic_simulation(car_num = car_num[i], v_max,
                                  distance, p, t)
}

for(i in 1:length(sims2)){
  total_dist_list2[[i]] <- apply(sims2[[i]], MARGIN = 2, calculate_distance, d = distance)
}

ySim2 <- sapply(total_dist_list2, sum)


# ITERATION 4
sims3 <- list()
total_dist_list3 <- list()


for(i in 1:length(car_num)){
  sims3[[i]] <- traffic_simulation(car_num = car_num[i], v_max,
                                  distance, p, t)
}

for(i in 1:length(sims1)){
```

```
  total_dist_list3[[i]] <- apply(sims3[[i]], MARGIN = 2, calculate_distance, d = distance)
}

ySim3 <- sapply(total_dist_list3, sum)


# ITERATION 5
sims4 <- list()
total_dist_list4 <- list()


for(i in 1:length(car_num)){
  sims4[[i]] <- traffic_simulation(car_num = car_num[i], v_max,
                                   distance, p, t)
}

for(i in 1:length(sims1)){
  total_dist_list4[[i]] <- apply(sims4[[i]], MARGIN = 2, calculate_distance, d = distance)
}

ySim4 <- sapply(total_dist_list4, sum)

# Repeat the simulations in the part b) four more times. Add the four new
# results to the previous plot. Make a second plot with the mean flow at each
# value of k along with the upper and lower approximate 99% confidence
# limits for the mean flow at that value of k. Roughly what range of k values
# has the maximum flow rate?

par(mfrow = c(1,2))

plot(car_num, ySim, type = "l", main = "Total Distance Traveled", xlab = "Number of cars", ylab = "Distance Traveled")
lines(car_num, ySim1, type = "l", col = "red")
lines(car_num, ySim2, type = "l", col = "red")
lines(car_num, ySim3, type = "l", col = "red")
lines(car_num, ySim4, type = "l", col = "red")


# average of the distances
avg_distance_traveled <- (ySim + ySim1 + ySim2 + ySim3 + ySim4)/5
distance_traveled_matrix <- rbind(ySim, ySim1, ySim2, ySim3, ySim4) # create matrix to facilitate calculations of standard dev
iations for distance per car_num across all trials

sd_vector <- list()

for (i in 1:dim(distance_traveled_matrix)[2]){
  sd_vector <- append(sd_vector, sd(distance_traveled_matrix[,i])) # Calculate the standard deviations of the distances for ea
ch number of cars and save them to a vector
}

sd_vector <- unlist(sd_vector) # unnest the nested lists in sd_vector

ciUpper <- avg_distance_traveled + qnorm(0.995) * sd_vector # calculate upper bound for 99% conf interval
ciLower <- avg_distance_traveled - qnorm(0.995) * sd_vector



plot(car_num, avg_distance_traveled, col="blue", type = "l", main = "Average Distance Traveled For varying number of Cars", su
b = "99% CI", xlab = "Number of Cars", ylab = "Average Distance Traveled") # plot the average distance traveled
# Plot confidence intervals
lines(car_num, ciUpper, col = "green")
lines(car_num, ciLower, col = "green")


print("The system has the maximum flow rate at a range of 100 - 120 cars on the track.")
```

```
# breaking time:
traffic_simulation<- function(car_num,v_max,distance,p, t){
  v_vector = rep(v_max,car_num)
  d_vector = rep(distance/car_num,car_num)
  p_vector = seq(distance/car_num, distance, by = distance/car_num)

  result = data.frame(matrix(NA,nrow = t, ncol = car_num))
  time = 1
  break_time = 0
  while(time <= t){
    for(i in 1:car_num){
      #algorithm start
      v_vector[i] = min(v_vector[i]+1,v_max)
      if(i == car_num){
        v_vector[i] = min(v_vector[i],d_vector[1]-1)
        if(v_vector[i] == d_vector[1]-1){
          break_time = break_time + 1
        }
      }
      else{
        v_vector[i] = min(v_vector[i],d_vector[i+1]-1)
        if(v_vector[i] == d_vector[i+1]-1){
          break_time = break_time + 1
        }
      }

      u = runif(1,0,1)
      if(u<p){
        v_vector[i] = max(0,v_vector[i]-1)
      }

      p_vector[i] = p_vector[i]+v_vector[i]

      if(p_vector[i] >= distance){
        p_vector[i] = p_vector[i] - distance
      }

      #update distance between car i and car i-1
      if(i != 1){
        if(p_vector[i] - p_vector[i-1] > 0){
          d_vector[i] = p_vector[i] - p_vector[i-1]
        }else{
          d_vector[i] = p_vector[i] + distance - p_vector[i-1]
        }

      }else{
        #for the first car
        if(p_vector[i] - p_vector[car_num] > 0){
          d_vector[i] = p_vector[i] - p_vector[car_num]
        }else{
          d_vector[i] = p_vector[i] + distance - p_vector[car_num]
        }
      }
    }
    #print(p_vector)
    result[time,] = p_vector
    time = time + 1
  }
  return(break_time)
}

car_num = seq(55,200,5)
v_max = 5
distance = 1000
p = 1/3
t = 1000

sims <- list()
sims_avg <- list()
x = 1:length(car_num)
for(i in x){
  a <- traffic_simulation(car_num = car_num[i], v_max,
```

```
                                distance, p, t)
  sims[i] <- a
  sims_avg[i] <- a / car_num[i]
}

#total break time
plot(car_num,sims,"l",main = "Occurrences of braking",xlab="number of cars",ylab="breaking times")
```