# Steering Algorithms in Simulating Traffic

Hanna Yu

December 21, 2020

**Abstract**

Building on a previous agent-based simulation of traffic in Matlab, I add: 1) a directed graph representing our system of roads, and 2) a steering method based on the shortest path function of directed graphs. I examine the effect of steering method on traffic performance and find that for very long run times, steering based solely on driving in the directing closest towards the destination yields poor overall traffic performance. Completely random steering performs slightly better, followed by steering based on a combination of the first method and random steering, and finally steering based on the shortest path, which performs the best.

## 1 Introduction

Simulating traffic is a fun exercise and can be framed in many different ways. I build my simulation on Professor Charles Peskin's model of traffic which is agent-based, meaning that it tracks individual cars as opposed to treating traffic as a flow/fluid.

In this paper I will give an overview of a simple agent-based simulation of traffic, explain my additions to the code, and discuss the results of varying run time length and steering methods.

## 2 Agent-Based Model of Traffic

### 2.1 Overview

Our agent-based traffic model consists of a system of roads, cars which enter our system at random times and locations with random destination points, and traffic lights at each intersection. The system of roads can be arbitrary within the assumption that they are single-lane and one-way (although we can easily modify the model to account for two-way roads).

The original simulation is composed of eleven Matlab files and starts in a main file called traffic.m. First, the program initializes the system constants and creates the system of roads. Then for each time step until the total run time, the program updates the traffic lights, creates new cars that will appear on the road, and moves cars which were previously created. Figure 1 shows how each file is called in this process.

### 2.2 Steering Method

In the original code, cars choose which block they should turn into by finding the vector from their current position to their destination and picking the block whose direction yields the largest

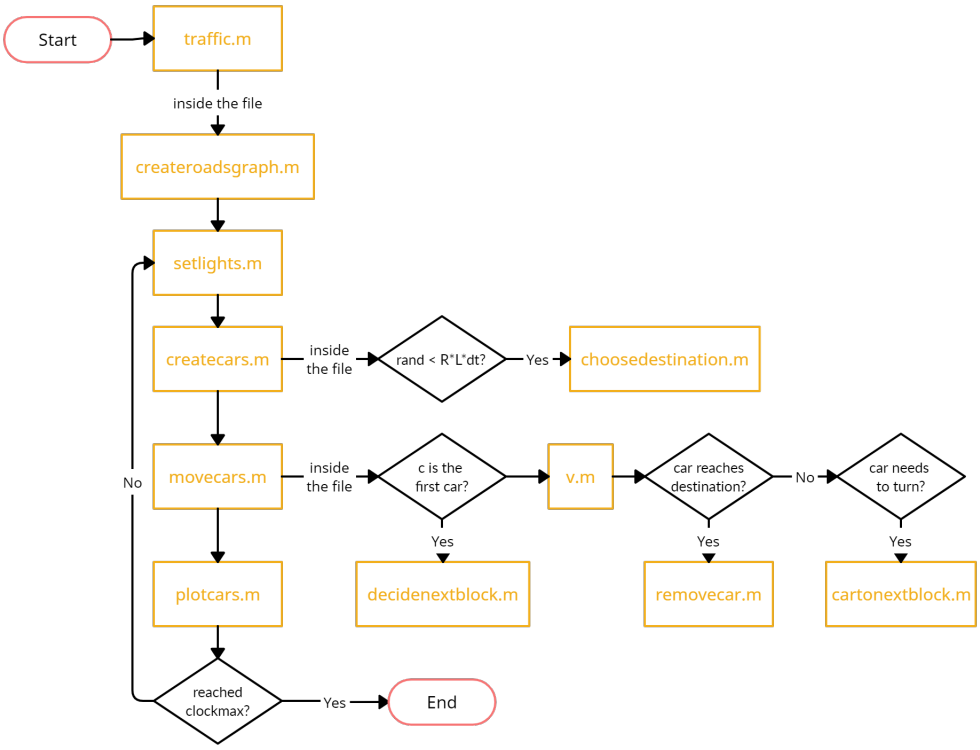


Figure 1: Flow chart with all program files

dot product with this vector. In addition to this steering method, cars may also choose to turn into any random outgoing blocks, which happens at a rate determined by the variable *prchoice*.

# 3 Additions to the Model

## 3.1 Directed Graph

The biggest change in the original code is an addition of "DGraph.m" which, given a number of avenues and streets, creates a directed graph representation of the roads. Since we no longer need to input the road geometry node by node and block by block, it allows us to easily change the size of our system of roads, although it also confines us to roads which are rectangular grids. In addition, the directed graph is constructed in such a way that the streets and avenues alternate in direction. Figure 2 shows the plot for a directed graph representing a system of four streets and six avenues. Note that if we would like the graph to be such that we can get to any destination point from any starting point on the graph, then the number of streets and avenues must both be even.

Because the directed graph is not fixed in physical geometry, it still remains for us in the file createroadsdgraph.m to assign coordinates to each node. I find the coordinates using the meshgrid() function and assuming that all blocks have length 1.

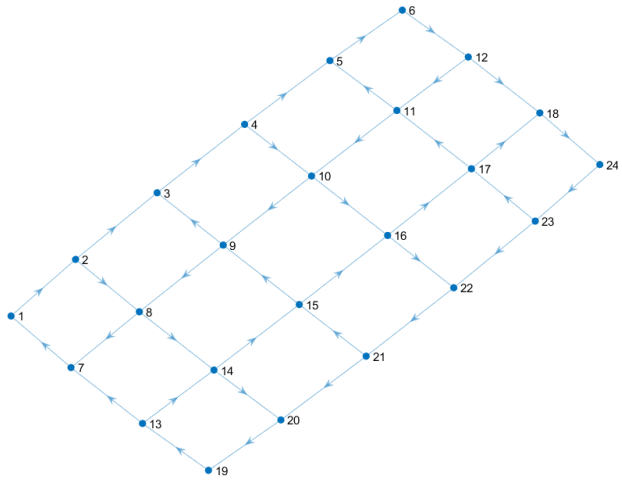


Figure 2: Directed graph for a system of four streets and six avenues

The most important benefit of adding a directed graph representation to the code is that we can now consider a steering method based on the shortest path between a car's entrance and its destination. Whenever a new car is created, we use the function shortestpath() and save the array of intersection indexes leading the car to its destination. We consult this shortest path when a car needs to decide the next block it should enter into.

Other smaller changes in the code include keeping track of the distance traveled by each car (in order to obtain data about each car's average speed during the run) (variable distancetraveled created in "choosedestination.m" and updated in "movecars.m"), recording a video of each run, and storing variables we are interested in an Excel file at the end ("recordData.m").

# 4 Results and Discussion

## 4.1 Steering Method with prchoice = 0

Figure 3 shows the Excel file output for a certain simulation run, which I call "pr0-s1-1280". For all the simulations I ran, *nstreets*, *navenues*, *tlcstep*, and $R$ were kept at the same values shown in the figure. Recall that *prchoice* is the frequency that cars using the original steering method choose the next block randomly. In the case in Figure 3, no random steering takes place, so we used only steering based on the dot-product calculation. *seed* is an integer set so that we are able to recreate the random movements in the simulation generated by the random number generator. This means that given the same system constants, one can obtain again the exact same simulation outcome.

Below the system constants are the statistics which give us an idea about the performance of traffic in our simulation. Some of the car data on the right have been color-coded to emphasize certain numbers. The rightmost column recording speed highlights large numbers in green, small numbers in red, and those in between in white. Notice that in the beginning of the simulation, most cars have a reasonable average speed. However as that as time passes, the average speed of the cars drops. Figure 4, which is that data for the last 38 cars, shows that most cars at the end have very low average speeds.

| | A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | **nstreets** | 4 | | **tenter** | **texit** | **x enter** | **y enter** | **x dest** | **y dest** | **time on** | **distance** | **speed** |
| 2 | **navenues** | 4 | | 0.01 | 7.15 | 2 | 1.30233 | 3.09234 | 1 | 7.14 | 4.79 | 0.67 |
| 3 | **tlcstep** | 1 | | 0.11 | 10.24 | 1.02324 | 2 | 2.02622 | 2 | 10.13 | 7.00 | 0.69 |
| 4 | **R** | 0.05 | | 1.50 | 6.66 | 1 | 2.99043 | 1.49432 | 2 | 5.16 | 4.50 | 0.87 |
| 5 | **run time** | 1280 | | 1.52 | 6.83 | 2 | 2.23132 | 3.3438 | 4 | 5.31 | 4.42 | 0.83 |
| 6 | **shortest-path** | 0 | | 2.06 | 10.90 | 3 | 1.75419 | 3 | 2.14883 | 8.84 | 7.61 | 0.86 |
| 7 | **seed** | 1 | | 2.07 | 9.59 | 3.37724 | 1 | 2 | 1.93281 | 7.52 | 7.56 | 1.00 |
| 8 | **prchoice** | 0 | | 4.10 | 5.87 | 2.41738 | 2 | 1.18699 | 2 | 1.77 | 1.23 | 0.70 |
| 9 | | | | 4.60 | 12.17 | 2 | 2.66994 | 2 | 1.0264 | 7.57 | 6.36 | 0.84 |
| 10 | **avg time to destination** | 8.49156 | | 4.73 | 10.17 | 1.83977 | 3 | 3 | 3.175 | 5.44 | 4.99 | 0.92 |
| 11 | **avg speed** | 0.56452 | | 4.92 | 0.00 | 2 | 3.32057 | 2.89472 | 4 | 1275.08 | 527.91 | 0.41 |
| 12 | **% reach desination** | 0.90575 | | 7.54 | 13.62 | 2.34589 | 1 | 3.41073 | 4 | 6.08 | 5.24 | 0.86 |
| 13 | **final num on road** | 146 | | 7.54 | 11.94 | 4 | 1.25967 | 1.42995 | 2 | 4.40 | 3.31 | 0.75 |
| 14 | | | | 8.26 | 0.00 | 3.6063 | 2 | 1 | 2.71553 | 1271.74 | 660.54 | 0.52 |
| 15 | | | | 10.20 | 0.00 | 1.02489 | 3 | 2.7913 | 4 | 1269.80 | 523.83 | 0.41 |
| 16 | | | | 10.59 | 13.46 | 1 | 2.83856 | 2 | 1.03942 | 2.87 | 2.88 | 1.00 |
| 17 | | | | 10.89 | 11.62 | 2.04513 | 2 | 1.42995 | 2 | 0.73 | 0.62 | 0.84 |
| 18 | | | | 11.32 | 20.40 | 3 | 3.46628 | 1.35699 | 3 | 9.08 | 6.82 | 0.75 |
| 19 | | | | 13.04 | 18.07 | 1.90256 | 3 | 4 | 1.02063 | 5.03 | 4.12 | 0.82 |
| 20 | | | | 14.59 | 16.77 | 1 | 3.09066 | 1.09695 | 1 | 2.18 | 2.19 | 1.00 |
| 21 | | | | 14.67 | 20.61 | 4 | 1.03684 | 3 | 2.44184 | 5.94 | 5.52 | 0.93 |
| 22 | | | | 15.35 | 23.24 | 2.36537 | 2 | 3.79805 | 2 | 7.89 | 6.57 | 0.83 |
| 23 | | | | 17.11 | 19.81 | 1.64498 | 3 | 4 | 3.29366 | 2.70 | 2.65 | 0.98 |
| 24 | | | | 18.96 | 20.93 | 1.27416 | 4 | 1.4578 | 3 | 1.97 | 1.73 | 0.88 |
| 25 | | | | 19.12 | 20.68 | 4 | 2.59253 | 4 | 3.63085 | 1.56 | 1.04 | 0.67 |
| 26 | | | | 19.63 | 25.20 | 3.98956 | 1 | 1 | 1.01454 | 5.57 | 5.00 | 0.90 |
| 27 | | | | 20.00 | 28.93 | 2.579 | 2 | 4 | 1.88378 | 8.93 | 6.46 | 0.72 |
| 28 | | | | 20.05 | 23.88 | 1 | 3.64239 | 2 | 1.10448 | 3.83 | 3.75 | 0.98 |
| 29 | | | | 20.88 | 23.51 | 3 | 3.05749 | 4 | 3.46561 | 2.63 | 1.52 | 0.58 |
| 30 | | | | 22.38 | 31.51 | 1 | 1.42745 | 4 | 3.1677 | 9.13 | 5.60 | 0.61 |
| 31 | | | | 22.47 | 26.89 | 2 | 1.36154 | 2 | 3.67801 | 4.42 | 2.32 | 0.52 |
| 32 | | | | 22.70 | 0.00 | 3.67144 | 1 | 1 | 2.03483 | 1257.30 | 651.14 | 0.52 |

Figure 3: Data for first 31 cars in pr0-s1-1280

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1530 | | | | 1259.38 | 0.00 | 3 | 1.52421 | 2.67945 | 1 | 20.62 | 4.14 | 0.20 |
| 1531 | | | | 1260.03 | 1274.34 | 3 | 1.52949 | 3.55062 | 2 | 14.31 | 2.98 | 0.21 |
| 1532 | | | | 1261.91 | 1274.27 | 3 | 1.75258 | 4 | 2.52998 | 12.36 | 3.28 | 0.27 |
| 1533 | | | | 1262.04 | 1270.77 | 3 | 3.54432 | 4 | 3.67443 | 8.73 | 2.22 | 0.25 |
| 1534 | | | | 1262.68 | 1278.34 | 2.87253 | 2 | 2 | 1.83962 | 15.66 | 4.71 | 0.30 |
| 1535 | | | | 1264.01 | 0.00 | 4 | 1.58204 | 1 | 3.21383 | 15.99 | 4.93 | 0.31 |
| 1536 | | | | 1264.19 | 0.00 | 1 | 1.60751 | 1 | 1.74579 | 15.81 | 3.56 | 0.22 |
| 1537 | | | | 1264.81 | 0.00 | 2.58414 | 2 | 1.13887 | 3 | 15.19 | 4.60 | 0.30 |
| 1538 | | | | 1265.06 | 1275.97 | 1.14573 | 3 | 2 | 2.91782 | 10.91 | 4.77 | 0.44 |
| 1539 | | | | 1268.77 | 0.00 | 1 | 3.03816 | 3.26154 | 2 | 11.23 | 3.58 | 0.32 |
| 1540 | | | | 1268.85 | 1279.53 | 2.48046 | 3 | 4 | 1.33355 | 10.68 | 3.85 | 0.36 |
| 1541 | | | | 1268.87 | 1272.81 | 2.02752 | 2 | 1 | 1.80045 | 3.94 | 1.23 | 0.31 |
| 1542 | | | | 1270.13 | 1276.76 | 2 | 1.68922 | 1 | 1.25056 | 6.63 | 2.06 | 0.31 |
| 1543 | | | | 1270.19 | 0.00 | 3 | 3.19186 | 3.97456 | 2 | 9.81 | 3.03 | 0.31 |
| 1544 | | | | 1270.92 | 1273.98 | 2 | 3.7096 | 1.33156 | 4 | 3.06 | 0.96 | 0.31 |
| 1545 | | | | 1272.91 | 0.00 | 1.44122 | 3 | 2 | 2.33564 | 7.09 | 3.36 | 0.47 |
| 1546 | | | | 1273.11 | 0.00 | 4 | 2.54967 | 1.8078 | 3 | 6.89 | 4.44 | 0.65 |
| 1547 | | | | 1273.25 | 0.00 | 2.33909 | 1 | 4 | 2.46178 | 6.75 | 2.34 | 0.35 |
| 1548 | | | | 1273.25 | 0.00 | 1.91693 | 2 | 2 | 2.44093 | 6.75 | 2.43 | 0.36 |
| 1549 | | | | 1276.96 | 0.00 | 3.33037 | 3 | 1.8006 | 4 | 3.04 | 0.60 | 0.20 |
| 1550 | | | | 1279.78 | 0.00 | 3.39046 | 1 | 3.8378 | 1 | 0.22 | 0.18 | 0.80 |

Figure 4: Data for last 38 cars in pr0-s1-1280

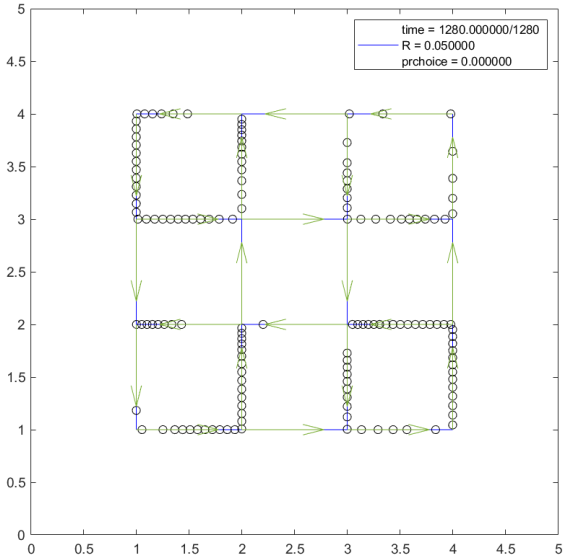


Figure 5

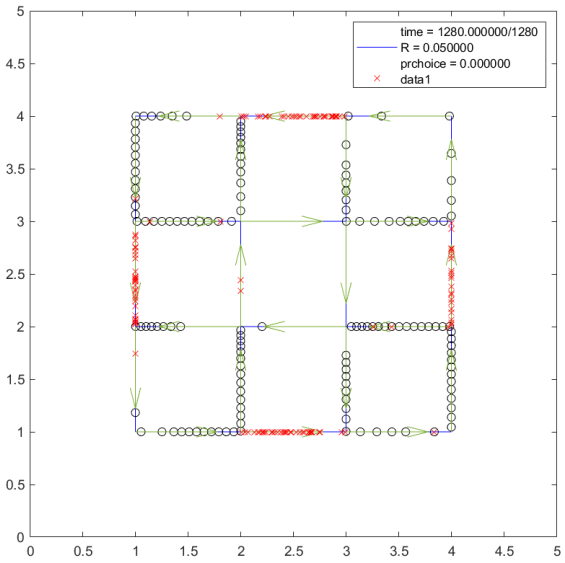


Figure 6

Figure 5, which shows the simulation plot at the end of the run, tells us why this happens. Because of our choice of dot-product steering, certain cars are never able to reach their destination. This happens if a car enters the road at one of the circuits at the corners of the grid and has a destination on a block turning turning into the corner circuit.

Figure 6, which plots the destinations of all the remaining cars, confirms this. At each intersec-

tion, the car either chooses to turn into the block pointing closest to its destination or is forced to turn into the only available block. These cars end up going round and round at the corners and effectively become stuck. Looking at *texit* (the time of exit for each car), Figure 3 again shows that some cars even created at the beginning are never able to reach their destination. As the simulation goes one, such cars accumulate at the corners and cause the cars there to steadily slow down in speed.

The statistics at the left in Figure 3 show that the average time cars takes to reach their destination is 8.49. This is one of the better numbers compared those for other steering methods (see Figure 7). However, only 91% of cars do reach their destination, and the number of cars left on the road at the end of the simulation is 148, which is very large.

## 4.2 Random Steering and Shortest Path Steering

Random steering and steering based on the shortest path do not face the same problem as dot-product steering. Figures 7, 8, 9, and 10 graph the performance statistics for various steering methods at different lengths of run time.

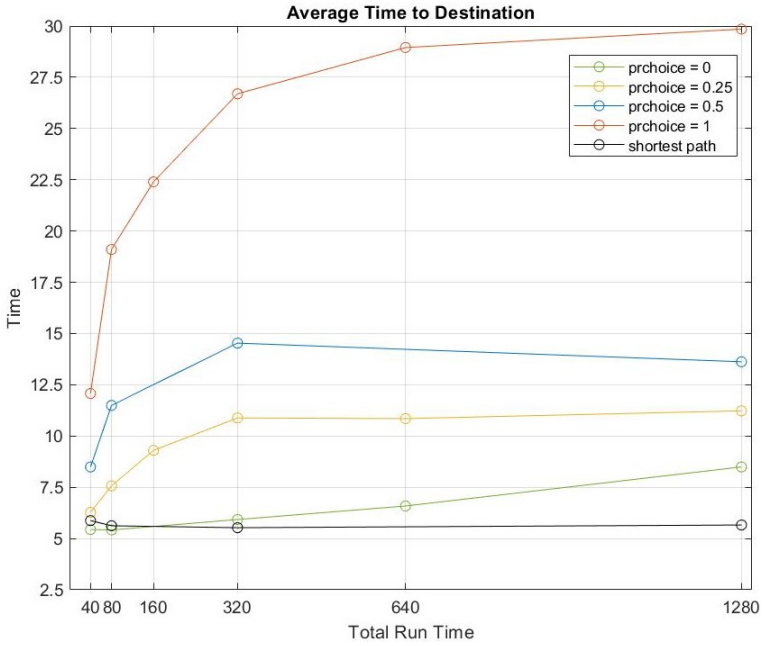


Figure 7

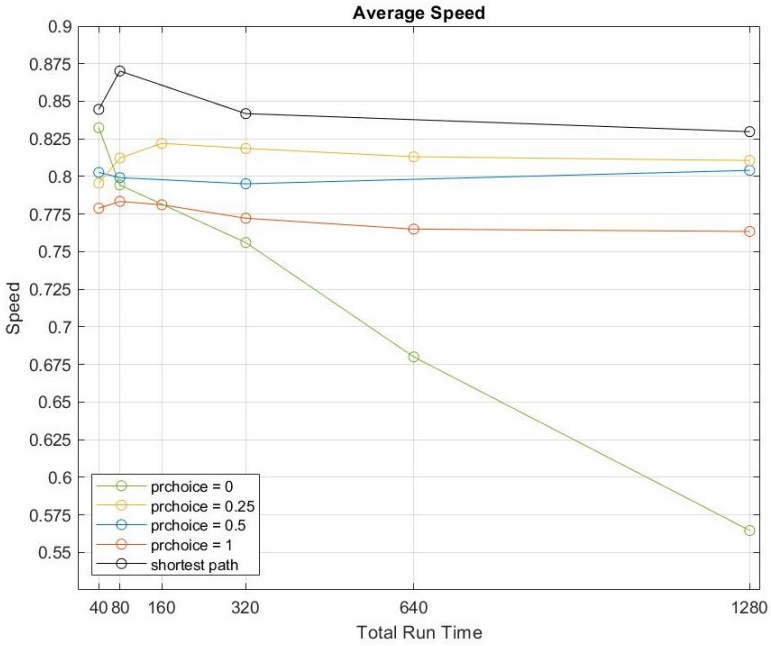


Figure 8

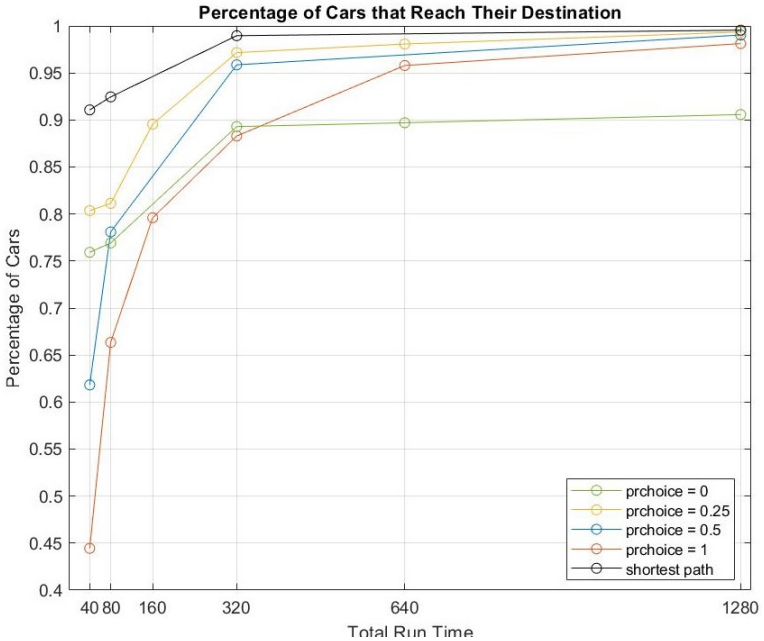


Figure 9

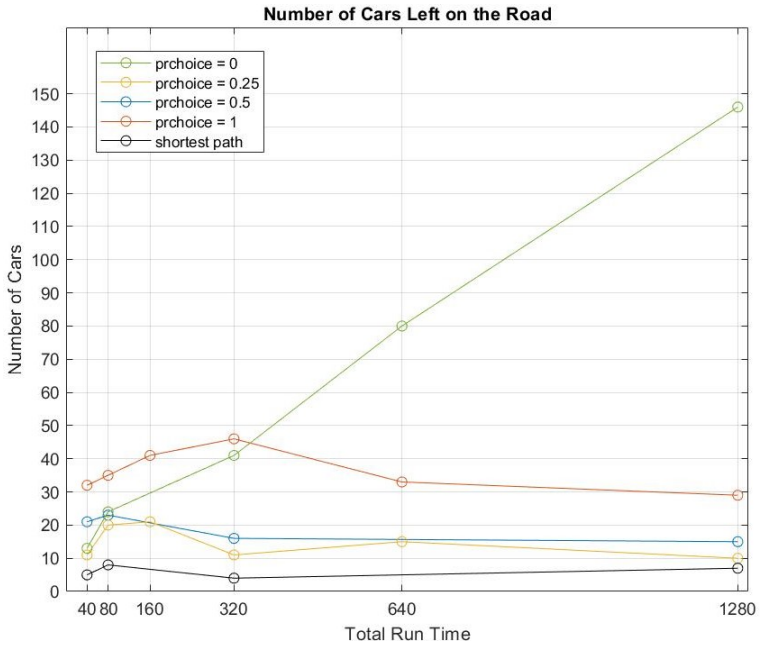


Figure 10

Notice that for run time 1280, random steering yields a better percentage of destination reach, higher average speeds, and a lower final number of cars on the road. Through completely random driving, all the cars at the beginning of the simulation run eventually hit their destination. However, the average time it takes for cars to arrive there is much longer.

For steering methods involving a combination of dot-product steering and random steering, a *prchoice* of 0.25 seems to yield the best results for all performance statistics at all lengths of run time. Trumping over all other steering methods is the shortest path steering method. At run time 1280, it gives the lowest average time to destination, the highest average speed, the higher percentage of cars that reach their destination, and the lowest final number of cars on the road. Its level of traffic performance for shorter run times is also the most consistent among other steering methods.

## 4.3 Variations due to Random Processes

In the previous simulation runs, all the seeds were set to 1. Now for $prchoice = 1$, and run time 1280, I examine how randomness produces variation in our performance statistics by testing $seed = 2$ and $seed = 3$. The next five figures show that the variations are small compared to differences exhibited between different steering methods. This makes us confident that the results we compared before are valid.

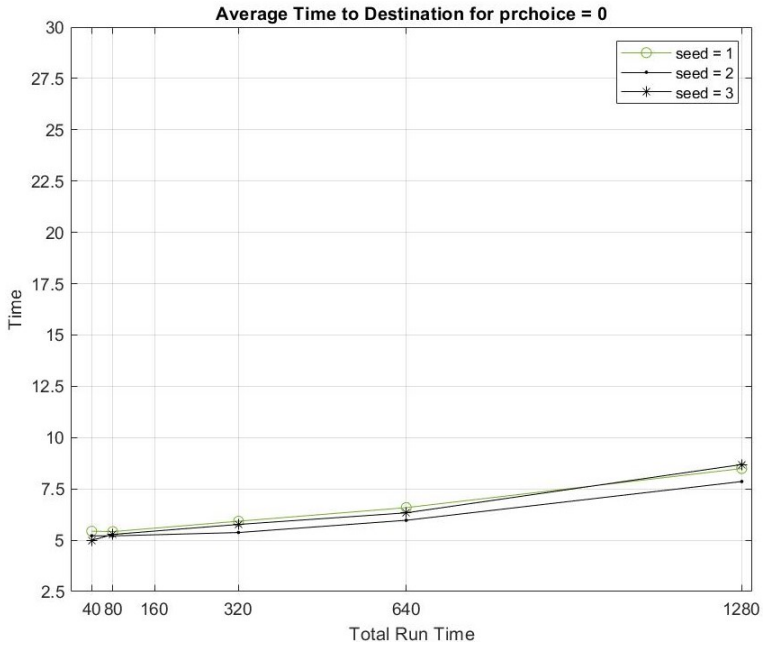


Figure 11

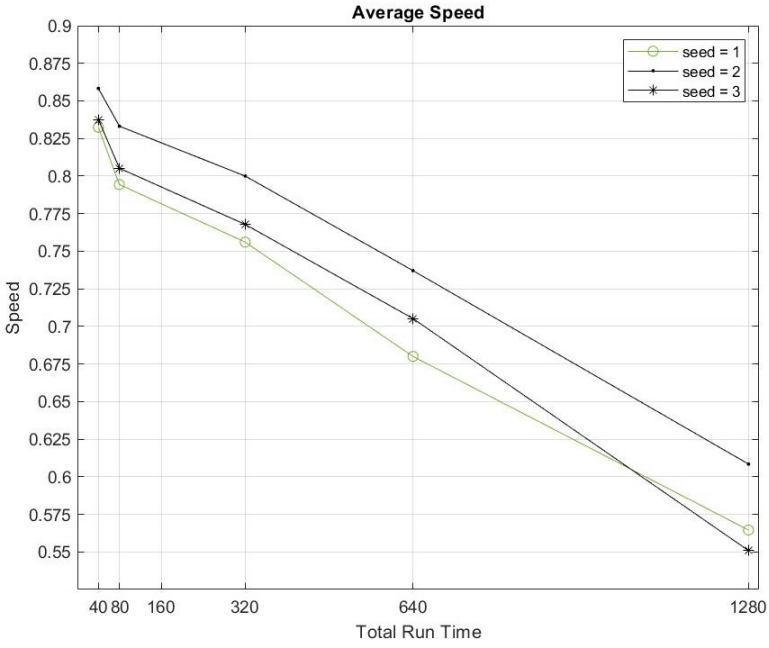


Figure 12

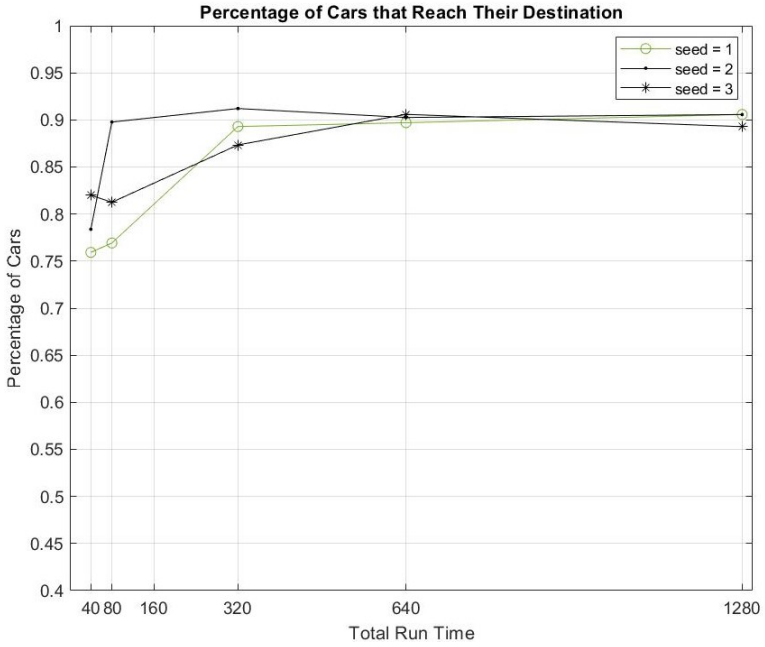


Figure 13

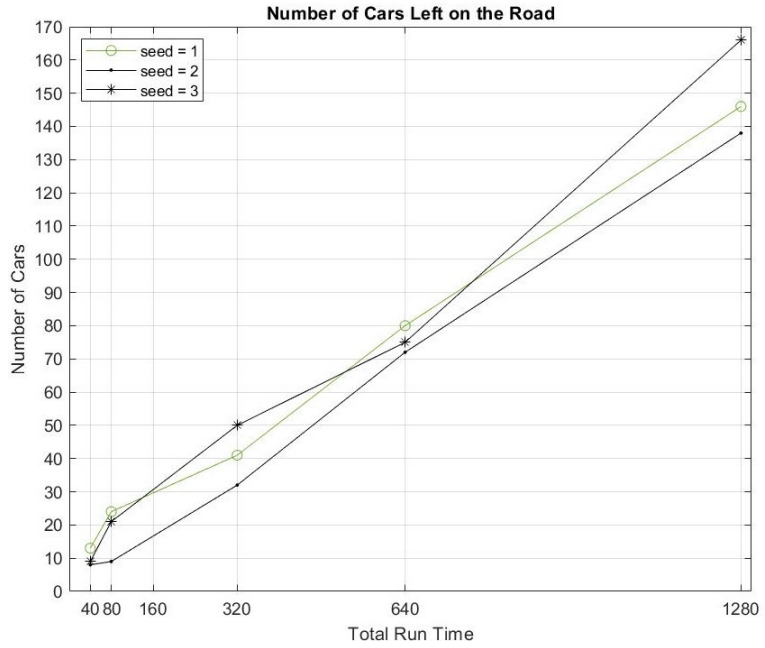


Figure 14

# 5 Conclusion

Building on Professor Charles Peskin's simulation of traffic in Matlab, I added a directed graph representation of our system of roads, and a steering method based on the shortest path function of directed graphs. I then examined the effect different of steering methods on traffic performance. I found that for very long run times, dot-product steering performs the worst, followed by completely random steering, then steering based on a combination of the first method and random steering, and finally steering based on the shortest path, which performs the best. I also found that for a very long run time and $prchoice = 0$, the random processes in the simulation produce little variation in the overall performance statistics.

# 6 Acknowledgements

I'd like to thank Professor Charles Peskin and Scott Weady from the Courant Institute of Mathematical Sciences at New York University for their assistance with this project.

# 7 References

Charles Peskin lecture on "Traffic Simulation on an Arbitrary Network of Roads with Lights at Intersections and Cars that Steer to Destinations" (Fall 2020)
https://www.math.nyu.edu/faculty/peskin/modsim_lecture_notes/traffic_simulation_notes.pdf

# 8 Appendix

All the Matlab code can be found on GitHub: https://github.com/hannayu/Traffic-Simulation