Four-Lane Cellular Automata with 3 different passing rules

Team No. 31664

February 10, 2014

Abstract/Summary

In this paper, a multi-lane cellular automata model of freeway traffic is used as a metric for the evaluation of traffic flow and safety for various driving rules, including the right hand rule for driving. In order give validity to our granular model for traffic flow we begin with an explanation of how our cellular automata model incorporates concepts from car following and kinematic wave models for traffic flow. Next we use our model to evaluate the right hand rule against other passing rules. Finally we discuss the efficacy of the right hand rule for traffic flow and safety.

Problem

The keep-right-except-to-pass rule, or right hand rule is a ubiquitous heuristic for organizing traffic flow around the world, followed only by its sister rule, keep-left-except-to-pass in selected countries where driving on the left is the norm. However, a rigorous analysis of the efficacy of these rules has yet to be performed. We propose cellular automata model to act as a metric of the traffic flow and safety induced by the right hand rule, and two additional rules which propose as alternatives. The first of these is the fast rule, where vehicles attempt to maximize their velocities through passing using local spatial information. We refer to the other rule as the no rule, as vehicles under this rule will pass randomly between lanes. The cellular automata approach allows for simple statistics gathering by implementing the microscopic behaviors of each vehicle according to each rule, and observing the macroscopic behavior of the system as a whole. We also consider the effects of these rules in countries where driving on the left is the norm.

Background

Ever wonder why we drive on the right lane in the U.S. and sit in the left side of the car? Back in the 1700s when wagons were used to haul farm products, driver's preferred sitting left of the horse so that his right arm was free to last the team (citation). Today in the U.S. it is regulation that we drive on the right side of the road; the U.S. and 65% of the world employ this rule, and it is called the Right Hand (RH) Rule. The Left Hand (LH) Rule, where drivers drive on the left side of the road, is used by 35% of the world including the U.K.

World-wide, these two rules are the most popular if not the only ones used, but what rule should be embraced? That is the question that we hope to assess in our report, and analyze in parallel with safety and traffic flow. There is not much public traffic data available, so we found it more appropriate to model the road by a modified version of the Nagel-Schreckenberg Model and interpret those results.

Tools and Assumptions

- Our model was written using C++ code and was executed on an Intel Core i7 four core processor. We also used a Java program to confirm visually how are data was changing.
- Traffic fatality ratings are not determined
- Drivers are assumed to not crash, all follow the same uniform rules, and all are one unit
- Freeway lanes are spatially straight

Description of the Nagel-Schreckenberg Model

The Nagel-Schreckenberg model (NaSh model) is a single-lane representation of freeway traffic using a circular track. The circular track is divided into discrete cells, and each cell is either empty or contains a single vehicle. All vehicles behave according to the same acceleration and velocity rules. Each vehicle has a non-negative integer value for velocity up to a v_max. The track is updated in discrete time steps where all vehicle positions are updated in parallel. For each time-step, the NaSh model uses the following four steps to create a collision free update [1]:

- Acceleration: All velocities below v_max are incremented by 1.
- Slowing Down: If vehicle x is behind vehicle y, and distance(x, y) is less than velocity(x), then the velocity of x becomes min{distance(x, y), velocity(x)}
- Randomization: The velocity of each vehicle with velocity greater than 0 is decremented by 1 with probability p, a constant probability determined before the simulation begins.
- Car Motion: Each vehicle x is shifted forward by velocity(x)

The Acceleration phase provides a linear acceleration to each vehicle. The Slowing Down phase ensures that no vehicles collide. The Randomization phase creates a non-deterministic distribution of velocities. In free-flow conditions this velocity distribution is centered around v_max – p, where p is the probability to slow down. The Car Motion phase produces the state of the next time step.

Description of Model

Our model is a multi-lane extension of the NaSh model. Like the NaSh model, our track is discretized into cells which are either empty or contain a single vehicle. Our vehicles have integer velocity values, with a maximum velocity of 5. Each lane is a circular track updated in parallel. Vehicles have an additional parameter for passing memory called "pass reset". The pass reset of each vehicle is initialized to 3 and is decremented by 1 during the Acceleration phase, down to a minimum of 0. Vehicles can only pass if their pass reset is 0, and passing restores their pass reset value to 3. A global "look back" variable is used to determine how far vehicles look back in adjacent lanes before passing, and is adjusted to either 2 or 4 during simulation.

Passing between lanes is implemented as two additional phases to the NaSh model, one for passing to the left, and another for passing to the right. The sensitivity of the right hand rule to passing direction necessitates this division of the pass phase. The passing phase is dependent on the passing rule chosen for the simulation. When a vehicle passes between lanes, it is shifted by 1 cell in a direction perpendicular to its lane. We implemented three different passing rules: the right hand rule, where traffic passes to the right unless to overtake, a "fast" rule where each vehicle attempts to maximize its velocity through local information and indiscriminate passing, and a "no" rule, where each vehicle passes indiscriminately between lanes when possible. The decision to pass in each of the three passing rules is controlled by discrete random variables for each rule. A description of each of the passing rules is given in the table below.

Passing Rules and Behaviors

Passing Rule	Left Passing Behavior	Right Passing Behavior
Right Hand Rule	Faster vehicles have a higher probability to pass to the left if passing will increase their velocities.	Vehicles will pass to the right with constant probability.
Fast Rule	Vehicles have constant probability to pass to the left when passing will increase their velocities.	Vehicles have a constant probability to pass to the right if passing will increase their velocities.
No Rule	Vehicles will pass to the left with constant probability.	Vehicles will pass to the right with constant probability.

The phases for our model are described below. The passing phase is broken into left and right phases for each passing rule.

- Acceleration: For each vehicle x, velocity(x) = min{velocity(x)+1, max velocity}, and pass_reset(x) = max{pass_reset(x)-1, 0}
- Passing:
  - If Right Hand Rule:
    - Left Phase: For each vehicle x in lane L0, if x is not in the leftmost lane, and the below items hold, where L1 is the lane to the left of x, then x is shifted by 1 to the left, and pass_reset(x) is set to 3. Lanes are evaluated from the second to leftmost lane to the rightmost lane.
      - pass_reset(x) == 0
      - $\text{distance}_{L0}(x) < \text{velocity}(x)$, if x would have to slow down to avoid collision in L0
      - $\text{distance}_{L1}(x) >= \text{velocity}(x)$, if x would not have to slow down in the lane to the left
      - $\text{lookback}_{L1}(x) >=$ look_back, if there are look_back unoccupied spaces behind the position x would take in L1
      - $P_L(\text{velocity}(x)) = 1$, where $P_L(\text{velocity}(x))$ is a discrete binary random variable which is a function of velocity. $P_L(\text{velocity}(x)) = 1$ is higher for high velocities and lower for low velocities.
    - Right Phase: For each vehicle x in L1, if x is not in the rightmost lane, and the below items hold, where L0 is the lane to the right of x, then x is shifted by 1 to the right, and pass_reset(x) is set to 3. Lanes are evaluated from the second to rightmost lane to the leftmost lane.
      - pass_reset(x) == 0
      - $\text{lookback}_{L0}(x) >=$ look_back, if there are look_back unoccupied spaces behind the position x would take in L0
      - $P_R() = 1$, where P_R is a discrete binary random variable which is a not a function of velocity.
  - If Fast Rule: The fast rule has symmetrical update rules for the left and right phases. However, left passing is still evaluated before right passing.
    - Left Phase: For each vehicle x in lane L0, if x is not in the leftmost lane, and the

below items hold, where L1 is the lane to the left of x, then x is shifted by 1 to the left, and pass_reset(x) is set to 3. Lanes are evaluated from the second to leftmost lane to the rightmost lane.
        - pass_reset(x) == 0
        - $\text{distance}_{L0}(x)$ < velocity(x), if x would have to slow down to avoid collision in L0
        - $\text{distance}_{L1}(x)$ >= velocity(x), if x would not have to slow down to avoid collision in L1
        - $\text{lookback}_{L1}(x)$ >= look_back, if there are look_back unoccupied spaces behind the position x would take in L1
        - $P_{Fast}() = 1$, where P_{Fast} is a discrete binary random variable.
    - Right Phase: For each vehicle x in L1, if x is not in the rightmost lane, and the following hold, where L0 is the lane to the right of x, then x is shifted by 1 to the right, and pass_reset(x) is set to 3. Lanes are evaluated from the second to rightmost lane to the leftmost lane.
        - pass_reset(x) == 0
        - $\text{distance}_{L1}(x)$ < velocity(x), if x would have to slow down to avoid collision in L1
        - $\text{distance}_{L0}(x)$ >= velocity(x), if x would not have to slow down to avoid collision in L0
        - $\text{lookback}_{L1}(x)$ >= look_back, if there are look_back unoccupied spaces behind the position x would take in L0
        - $P_{Fast}() = 1$, where P_{Fast} is a discrete binary random variable.
  - If No Rule: The no rule has symmetrical update rules for the left and right phases. However, left passing is still evaluated before right passing.
    - Left Phase: For each vehicle x in lane L0, if x is not in the leftmost lane, and the following hold, where L1 is the lane to the left of x, then x is shifted by 1 to the left, and pass_reset(x) is set to 3. Lanes are evaluated from the second to leftmost lane to the rightmost lane.
    - pass_reset(x) == 0
    - $\text{lookback}_{L1}(x)$ >= look_back, if there are look_back unoccupied spaces behind the position x would take in L1.
    - $P_{No}() = 1$, where P_{No} is a discrete binary random variable.
  - Right Phase: For each vehicle x in L1, if x is not in the rightmost lane, and the following hold, where L0 is the lane to the right of x, then x is shifted by 1 to the right, and pass_reset(x) is set to 3. Lanes are evaluated from the second to rightmost lane to the leftmost lane.
    - pass_reset(x) == 0
    - $\text{lookback}_{L1}(x)$ >= look_back, if there are look_back unoccupied spaces behind the position x would take in L1.
    - $P_{No}() = 1$, where P_{No} is a discrete binary random variable.
- Slow Down: If vehicle x is behind vehicle y, and distance(x, y) is less than velocity(x), then the velocity of x becomes min{distance(x, y), velocity(x)}
- Randomization: The velocity of each vehicle with velocity greater than 0 is decremented by 1 with probability p, a constant probability determined before the simulation begins.
- Car Motion: Each vehicle x is shifted forward by velocity(x).

The Acceleration phase is used to manage the pass reset value of each vehicle. The other phases from the NaSh model remain unchanged in our implementation. Because the Passing phase is before the Slow Down phase, passing vehicles can cause other vehicles to slow down by passing. Other models (Wagner) have used the velocities of vehicles in adjacent lanes to prevent vehicles from slowing other vehicles by passing between lanes. Here a look back value is used to compensate for this behavior. During look back checks, passing is aborted if the location a vehicle would pass to have less than look back number of unoccupied cells behind it. With a look back value of 4 and a maximum velocity of 5, only the fastest vehicles will be slowed by vehicles passing between lanes. For our simulation we allow the look back value to be either 4 for minimal interference from passing, or 2 for a high interference with oncoming traffic when passing.

The right hand rules specifies that traffic passes to the right unless to overtake. We have simulated this logic into our passing phase for the right hand rule by implementing a greedy, stochastic left passing rule and a stochastic right passing rule. When evaluating the left passing phase, vehicles who will have to slow down in their current lane check the left lane and pass with a probability which increases with their velocity after looking back. When evaluating the right passing phase, vehicles will pass to the right with a constant probability after looking back. In this way, faster traffic will tend to pass to the left and avoid slowing down, and slower traffic has a lower probability of passing to the left, but a constant probability of passing to the right. It is important to note that although individual vehicles may violate the right hand rule on occasion, in the results section this stochastic process is shown to produce a good simulation of the right hand rule. The fast rule for passing is the greedy left passing rule from the right hand rule applied in both passing directions. Therefore, vehicles try to maximize their velocity with local information by passing in either direction. Similarly, the no rule is the right passing rule from the right hand rule applied in both passing directions. This rule simulates random passing and is used for control.

Results

In our experiment, we handled over 10 variables to vary results. These variables manipulated microscopic features, each individual vehicle, but in general conducted macroscopic effects. The variables had either a constant value or were varied slightly. Here we will talk about the ones that we found when varied had the most interesting effects. One of the major factors that we impose is a burn in period where we start the vehicles in a clustered area and we don't start measuring statistics after a specific number of time steps which in our experiment were always 200 time steps. This allows the model to settle into an equilibrium state so that unnatural initial configurations do not harm the statistics.

Table A.1's results are plotted across the diagrams in Appendix B. Figure B.1.1 and Figure B.2.2 display the how the lane's densities vary with a difference in lookback. For 4 lanes, we measured the density of the i-th lane ρ_i as such:

$$\rho_i = \frac{1}{l * (N - m)} \sum_{l_i \in L} x_{l_i}$$

$$l: lane\ size, L: Set\ of\ all\ lanes, N: Number\ of\ cycles, m: record\ start\ cycle$$

For the Right Hand rule (RH rule) if you observe Figure B.1.1 below the Lane Density increases as we start heading to the right; which we expected from the right hand rule. If you compare this Figure to Figure B.1.2 there doesn't appear to be much difference which suggests that lookback has little or no impact on Lane Density. In the other figures we can also observe this same difference. While the RH Lane Densities are skewed left, the No Rule (NR) and the Fast Rule (FR) have more uniform densities across all lanes. All Rules have an increase in density as the population density increases; this is to be expected as the population density controls the probability of spawning a vehicle thus controlling the population size. The greater the population the more congestion and thus the greater the lane density.

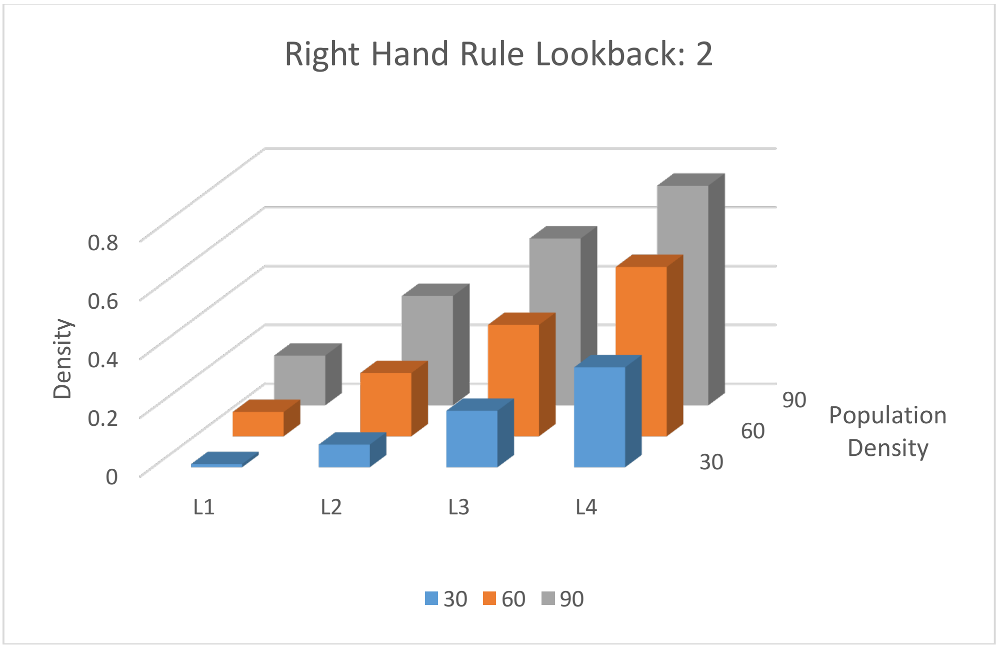

Figure B.1.1

v

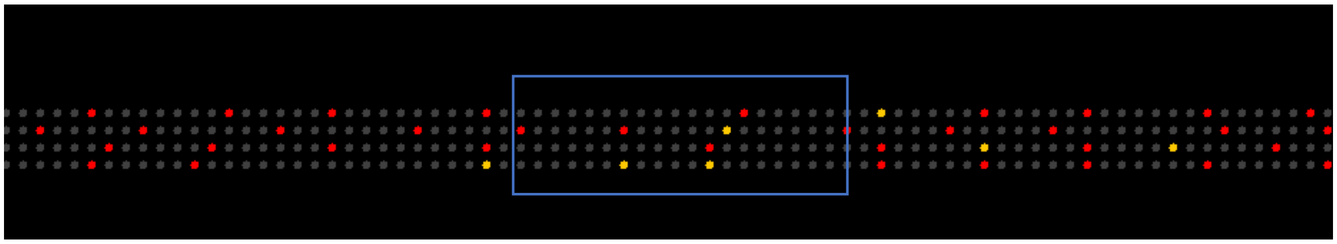

Figure 1

Illner and Bohun assert that flow-rate is a product of density and average velocity [8]. In our experiment we measured flow by monitoring how many vehicles were passing a block in the center and capturing their velocities as depicted in Figure 1 above. We use the following formula to measure the overall average flow-rate φ, this formula is a product in our model of both density and average velocities. Observe that Figure C.4 and Figure C.2 have the same shapes but vary in size, this comes as a result of this relationship and holds as evidence of it.

$$\varphi = \frac{\sum \frac{(vel)}{2(lane\_size)(maximum\_vel)}}{N - M}$$

N Garber and A. Ehrhar found that speed is not a major factor affecting safety that instead variability in speed spawns more accidents [7]. We measured the variability and speeds and frequency in lane changes in different rules so we can measure how safe vehicles are driving in each rule. We don't parameterize the two rules because we do not want to define safety but instead just assess it from two dimensions.

Upon viewing the Figures in Appendix C it will be quickly noticeable what effects these rules have on speed variability, passing frequency and flow-rate. Figure C.1 shows the Standard Deviation of Velocity measured for all vehicles through all time steps. The NR is displaying low deviations at high densities in Figure C.1 and, from Figure C.2, high flow at high densities, with the converse conditions being opposite. This shows us that when applying No Rule in high or medium densities, the vehicles have the most flow. Also at high densities passing frequency increases while at low it is slightly lower. A similar instance can be observed with the Fast Rule: at low densities there is both more flow and less variability while also maintaining little if no passing frequency. In general the RH rule, as seen in these figures tend to mediate variability and flow-rate between these two rules.

Conclusions

The results show conditions according to Garber (et al) and Illner (et al) that the best driving conditions that maximize both flow and safety vary at different speeds. The No Rule shows more flow and less variability at high densities while the Fast Rule shows more flow and less variability at low densities and high velocities. Both the Fast and the No rules show more flow and less variability than the RH rule at high and low velocities respectively. In general, the Fast Rule works better in higher velocity, low density conditions and the No Rule works better in lower velocity, high density conditions. The RH rule is the medium between the two in either case, and we witness high variability only because the cars are divided in lanes by order of velocity. Finally, we feel that the Right Hand Rule is the one that works the best under all conditions, while the other rules can work great under specific conditions.

Appendix A

Rule	Lookback	Population Density	L1	L2	L3	L4
RH	2	30	0.010675	0.077114	0.19224	0.339971
RH	2	60	0.082664	0.215254	0.379276	0.57614
RH	2	90	0.169422	0.372142	0.567504	0.747599
RH	4	30	0.014418	0.087249	0.174103	0.274229
RH	4	60	0.122973	0.23731	0.34708	0.519303
RH	4	90	0.213734	0.369739	0.536909	0.706285
NO	2	30	0.163674	0.215591	0.127587	0.056481
NO	2	60	0.292498	0.37478	0.316113	0.216609
NO	2	90	0.429325	0.499714	0.474081	0.420214
NO	4	30	0.147969	0.193103	0.162742	0.112853
NO	4	60	0.273228	0.312557	0.305215	0.292333
NO	4	90	0.462424	0.45273	0.442586	0.455593
FH	2	30	0.124328	0.135897	0.146676	0.159766
FH	2	60	0.259183	0.296347	0.330031	0.374439
FH	2	90	0.365089	0.414465	0.465106	0.528673
FH	4	30	0.139853	0.145091	0.150186	0.15487
FH	4	60	0.274789	0.287128	0.30537	0.33938
FH	4	90	0.418344	0.439202	0.469861	0.502593

Table A.1

Appendix B: Lane Densities

Right Hand Rule Lane Densities

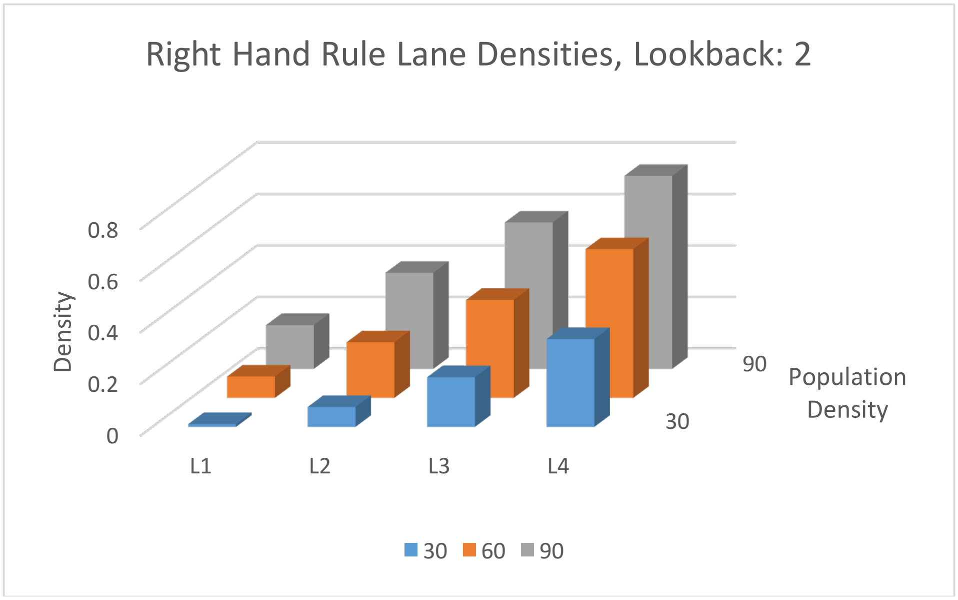

Figure B.1.1

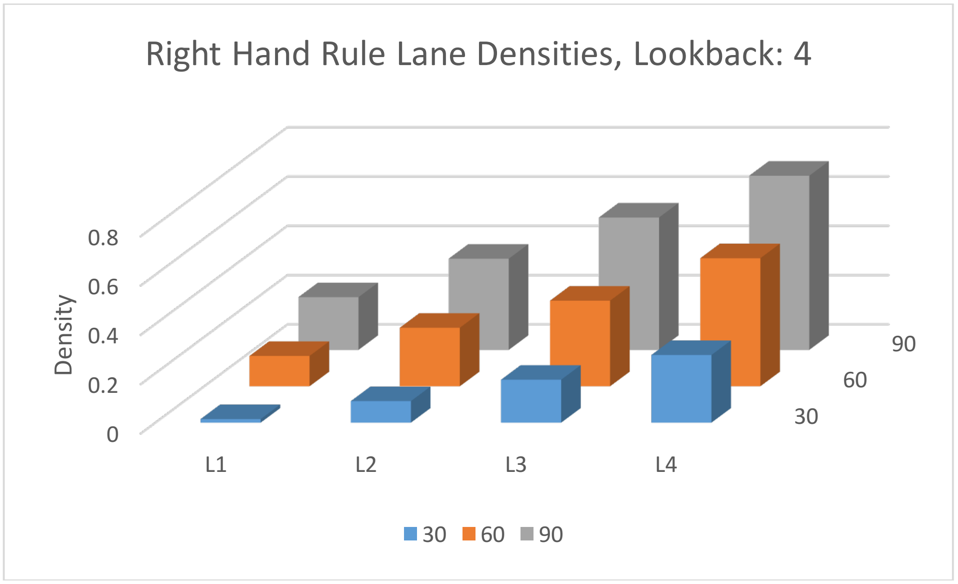

Figure B.1.2

No Rule Lane Densities

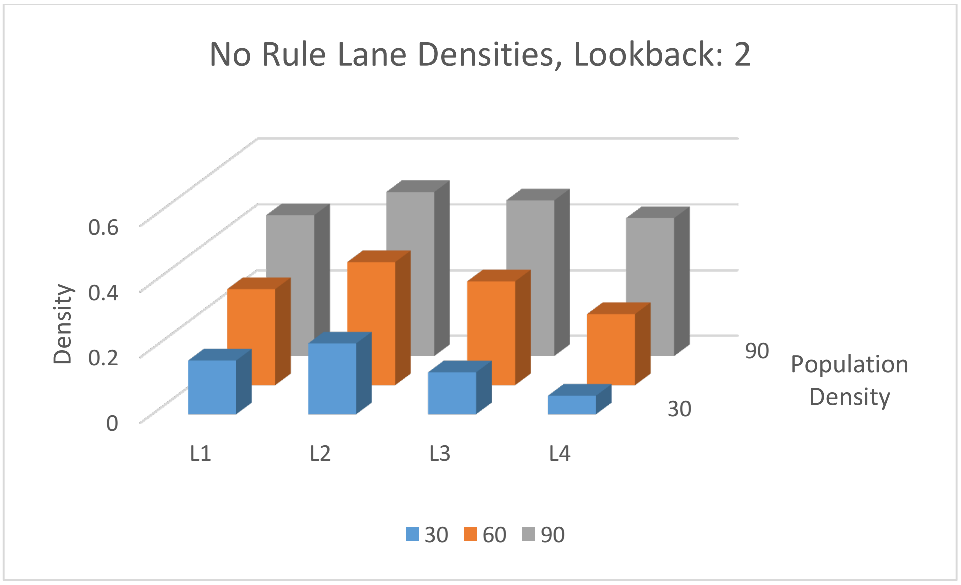

Figure B.2.1

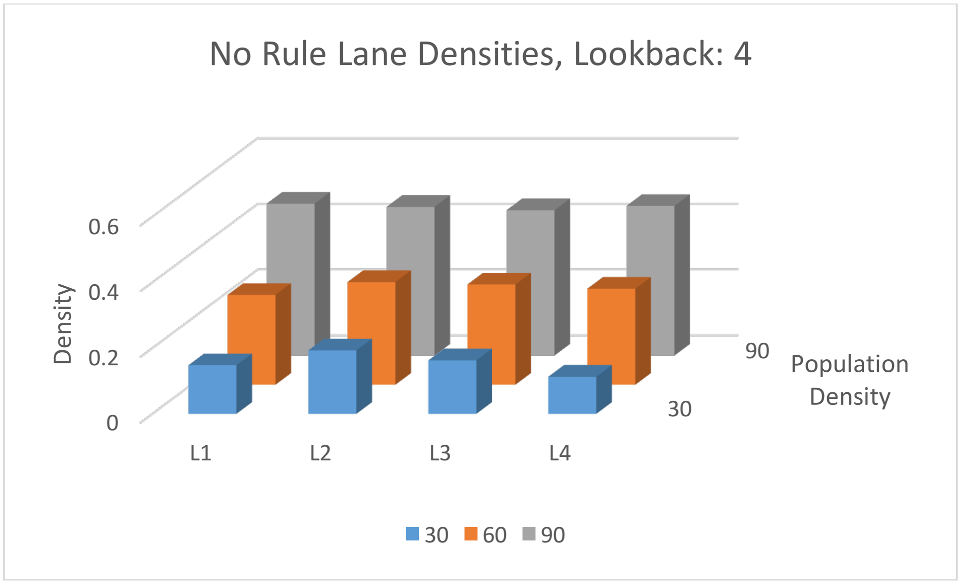

Figure B.2.2

Fast Rule Densities

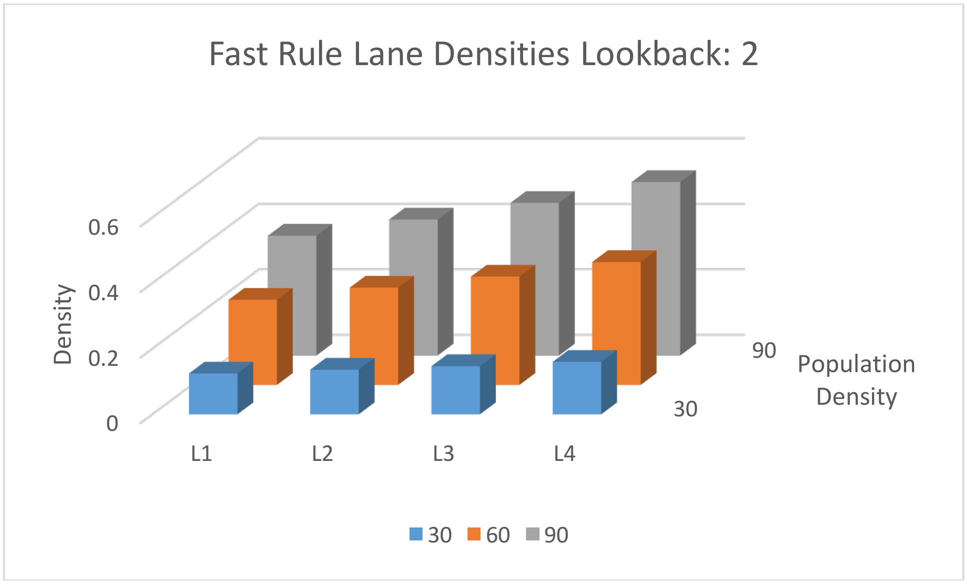

Figure B.3.1

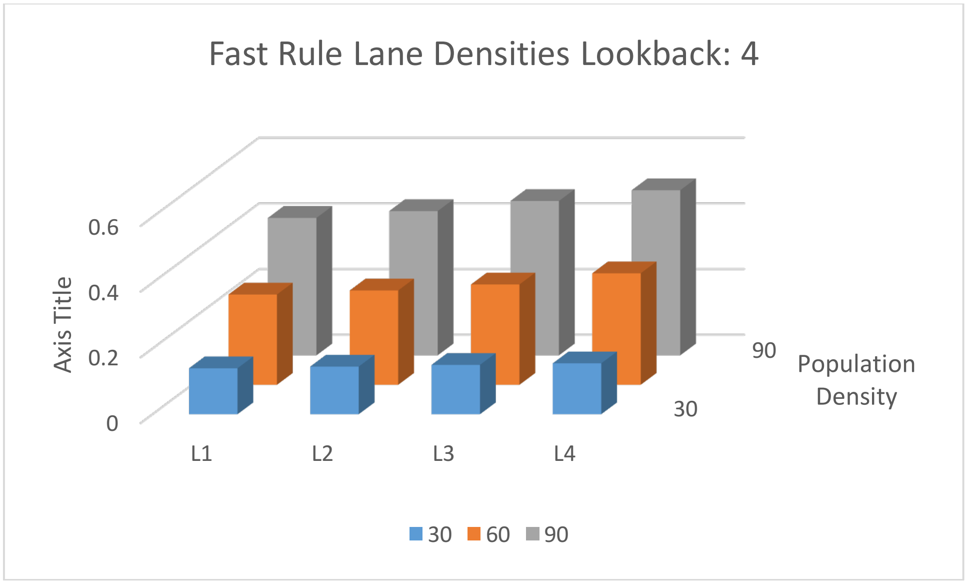

Figure B.3.2

Appendix C

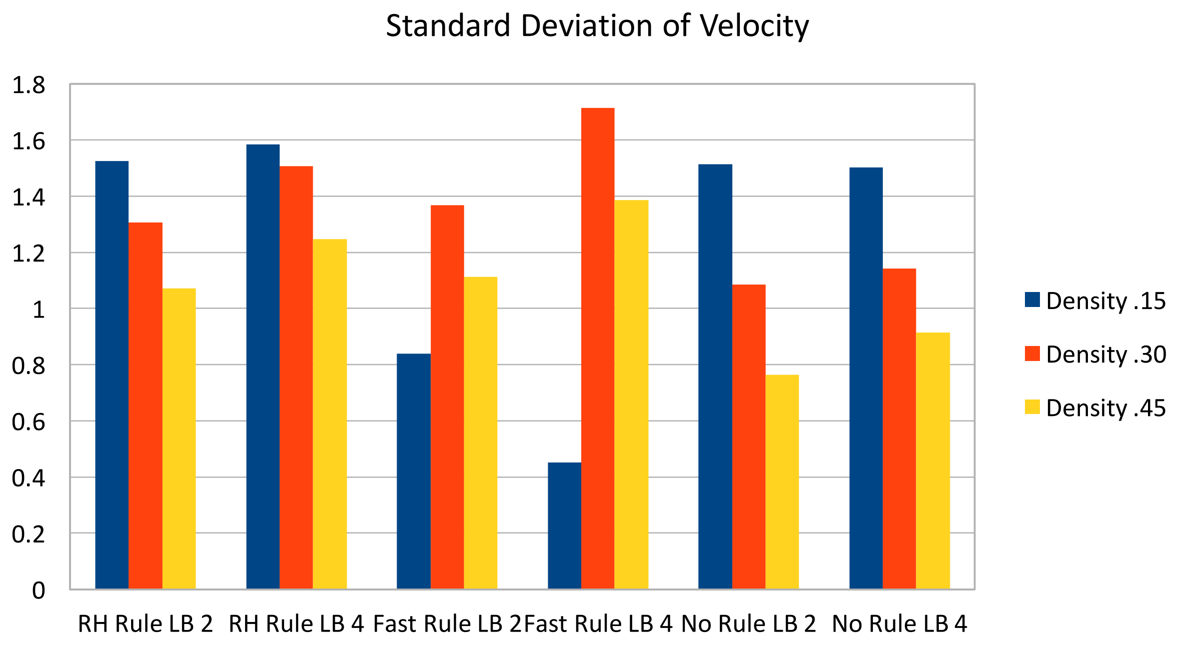

Figure C.1

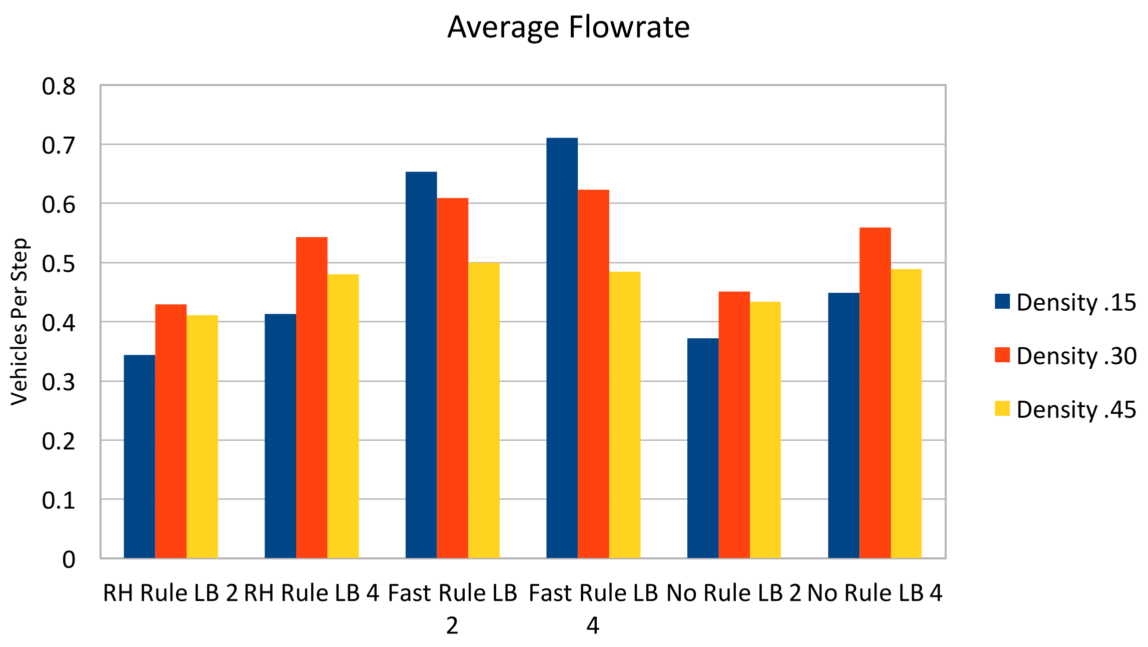

Figure C.2

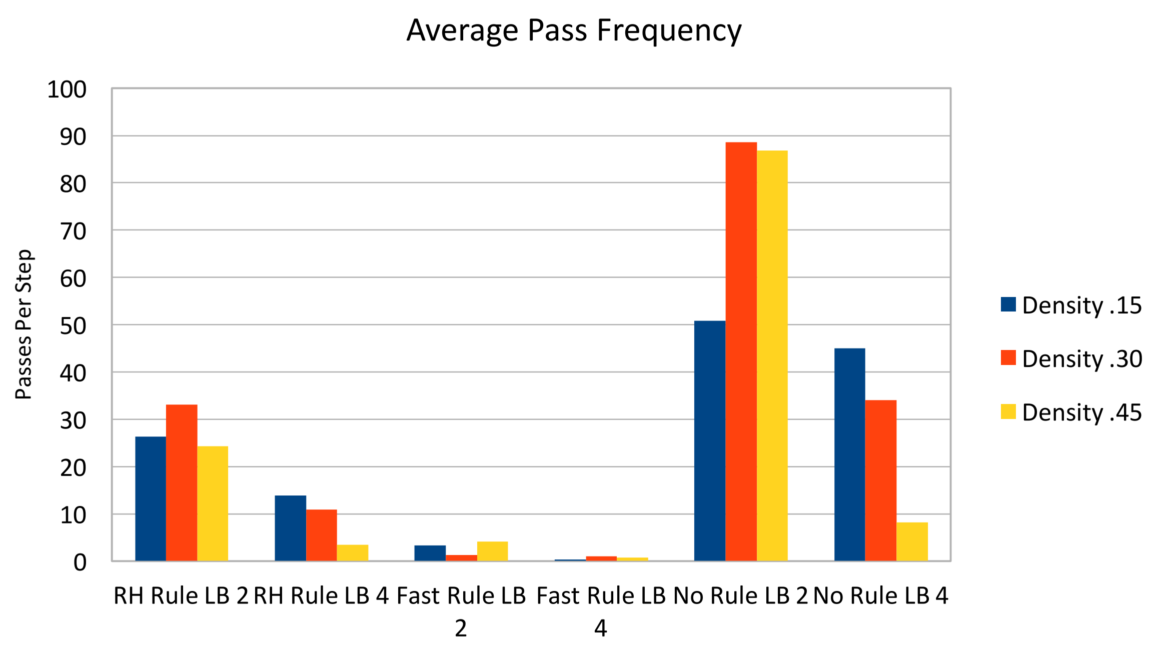

Figure C.3

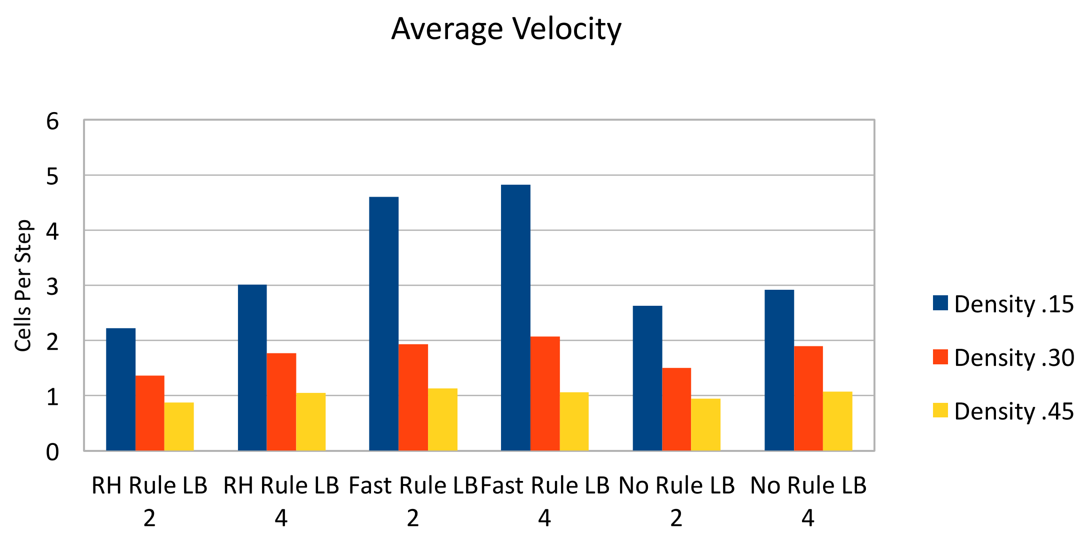

Figure C.4

Sources

[1] K. Nagel and M. Schreckenberg. J. Physique I, 2:2221 (1992)

[2] M Rickert, K. Nagel, M. Schreckenberg, A. Latour, 2008, Two Lane Traffic Simulations using Cellular Automata, http://arxiv.org/pdf/cond-mat/9512119.pdf (February 10, 2014)

[3] G. Bham, R. Benekohal, 2004, A high fidelity traffic simulation model based on cellular automata and car-following concepts,http://transportation.mst.edu/media/research/transportation/documents/Bham_2004_Transportation-Research-Part-C-Emerging-Technologies.pdf (February 10, 2014)

[4] B. Eisenblatter, L. Santen, A. Schadschneider, M. Schreckenberg, 2008, Jamming transition in a cellular automaton model for traffic flow,http://arxiv.org/pdf/cond-mat/9706041v1.pdf (February 10, 2014)

[5] C. Daganzo, 2004, In Traffic Flow, Cellular Automata = Kinematic Waves,http://www.its.berkeley.edu/publications/UCB/2004/RR/UCB-ITS-RR-2004-5.pdf (February 10, 2014)

[6] P. Wagner, K. Nagel, D. Wolf, 1996. Realistic Multi Lane Traffic Rules for Cellular Automata, http://ac.els-cdn.com/S0378437196003081/1-s2.0-S0378437196003081-main.pdf?_tid=06702468-92a8-11e3-b9f8-00000aab0f01&acdnat=1392073791_96aa92a83c765857367fa293b546c057(February 10, 2014)

[7] N Garber, A. Ehrhart, 2000. The Effect of Speed, Flow, and Geometric Characteristics on Crash Rates for Different Types of Virginia Highways,http://www.virginiadot.org/vtrc/main/online_reports/pdf/00-r15.pdf (February 10, 2014)

[8] Illner, Reinhard, C. Bohun, Samantha McCollum, and Thea Roode. *Mathematical Modelling, A Case Studies Approach*. N.p.: American Mathematical Society, 2005. Print.

Appendix E: Code

```
#include <iostream>
#include <math.h>
#include <cstdlib>
#include <fstream>

using namespace std;

//Constants
#define LOOKBACK 4   //greater than 0!
#define PASSRESET 3
#define LANE 4
#define LANESIZE 300
#define CYCLES 10000
#define START_METRICS 200
#define MAX_VEL 5
//Probabilities
#define P_SPAWNING 30
#define P_SLOWING 6
#define P_PASS_L 15
#define P_PASS_R 80
#define P_PASS_FAST 80
#define P_PASS_NO 80
//Switches
#define RH_RULE 0
#define NO_RULE 0
#define FAST_RULE 1


//Data Unit
struct block {
   bool occupied;
   int velocity;
   int passReset;
};
typedef struct block block;




int main(int argc, char const *argv[]) {
```

```
//Stream opener and and random seed generator
srand((unsigned int)time(NULL));
ofstream f;
f.open("traffic.in");

block freeway[LANE][LANESIZE];
int i,j;
int runCycles = CYCLES;
int numCars[LANE];
float passLeftVector[MAX_VEL+1];
double avgFlowRate = 0;
double avgVelocity = 0;
double avgVelocityPerCycle[CYCLES+1];
float passRightFreq = 0;
float passLeftFreq = 0;
float density[LANE];
long double velSumSquare = 0;
long double velSquareSum = 0;
long double variance = 0;
double populationSize = 0;

//Statistics
for (i = 0; i < LANE; i++) {
   density[i] = 0;
}
for (i = 0; i < CYCLES+1; i++) {
   avgVelocityPerCycle[i] = 0;
}

passLeftVector[0] = 0;
for (i =1; i < MAX_VEL+1; i++) {
   passLeftVector[i] = i * P_PASS_L + P_PASS_L;
}

//Track is populated with vehicles
for(i = 0; i < LANE; i++){
   numCars[i] = 0;
   for(j = 0; j < LANESIZE; j++){
      if(j < LANESIZE/2 && P_SPAWNING > (rand() % 100)){
         freeway[i][j].occupied = true;
         freeway[i][j].velocity = (rand() % MAX_VEL) + 1;
         freeway[i][j].passReset = PASSRESET;
         numCars[i]++;
      }
      else{
         freeway[i][j].occupied = false;
         freeway[i][j].velocity = 0;
         freeway[i][j].passReset = 0;
```

```
            }

         }
      }

      //////////////////////////////////////////////////////////////////////////////////////////////////////
      //////////////////////////////////////////////////////////////////////////////////////////////////////
      //////////////////////////////////////////////////////////////////////////////////////////////////////
      f << LANE << endl;
      f << LANESIZE << endl;
      //////////////////////////////////////////////////////////////////////////////////////////////////////
      //////////////////////////////////////////////////////////////////////////////////////////////////////
      //////////////////////////////////////////////////////////////////////////////////////////////////////

      while(runCycles-- > 0){
         //Step 0: Increase each cars velocity by 1 unit
         for(i = 0; i < LANE; i++){
            for(j = 0; j < LANESIZE; j++){
               if(freeway[i][j].occupied){
                  freeway[i][j].velocity = (int)fmin(freeway[i][j].velocity+1, MAX_VEL);
                  freeway[i][j].passReset = (int)fmax(freeway[i][j].passReset-1, 0);
               }
            }
         }

         int dist;
         int k;
         //Step 1: Passing Phase
         if (FAST_RULE) { //if fast rule being applied
            if (LANE > 1 && P_PASS_FAST > 0) { //if there are multiple lanes
               //pass to the left phase
               for(i = 1; i < LANE; i++) { //begin on the second to leftmost lane and proceed to the right
most lane
                  for(j = 0; j < LANESIZE; j++){
                     if (freeway[i][j].occupied && freeway[i][j].passReset == 0) {
                        dist = 0;
                        k = (j+1)%LANESIZE;
                        while(!freeway[i][k].occupied && dist < freeway[i][j].velocity) { //determine gap
                           k = (k+1)%LANESIZE;
                           dist++;
                        }
                        if (dist < freeway[i][j].velocity) { //if needs to slow down
                           dist = 0;

                           k = (j+1)%LANESIZE;
                           while(!freeway[i-1][k].occupied && dist < freeway[i][j].velocity) { //determine
gap
                              k = (k+1)%LANESIZE;
                              dist++;
                           }

                           if (dist == freeway[i][j].velocity) {
                              dist = 0;
                              k = j;
                              while(!freeway[i-1][k].occupied && dist < LOOKBACK) { //check lefter lane
by lookback distance
                            k = ((k-1)%LANESIZE < 0)?( (k-1)%LANESIZE + LANESIZE ):( (k-
1)%LANESIZE );
                            dist++;
                              }

                              if (dist == LOOKBACK && P_PASS_FAST > rand()%100) {
                            freeway[i-1][j].velocity = freeway[i][j].velocity;
                            freeway[i-1][j].passReset = PASSRESET;
                            freeway[i-1][j].occupied = true;

                            passLeftFreq++;

                            freeway[i][j].occupied = false;
                            freeway[i][j].passReset = 0;
                            freeway[i][j].velocity = 0;
                              }
                           }
                        }
                     }
                  }
               }
               //pass to the right phase
               for(i = LANE-2; i >= 0; i--) {
                  for(j = 0; j < LANESIZE; j++){
                     if (freeway[i][j].occupied && freeway[i][j].passReset == 0) {
                        dist = 0;
                        k = (j+1)%LANESIZE;
                        while(!freeway[i][k].occupied && dist < freeway[i][j].velocity) { //determine gap
                           k = (k+1)%LANESIZE;
                           dist++;
                        }
                        if (dist < freeway[i][j].velocity) { //if needs to slow down
                           dist = 0;
                           k = j;
                           while(!freeway[i+1][k].occupied && dist < freeway[i][j].velocity) { //determine
gap
                              k = (k+1)%LANESIZE;
                              dist++;
                           }
                    if (dist == freeway[i][j].velocity) {//if at least more velocity in left lane
                        dist = 0;
                        k = j;
                        while(!freeway[i+1][k].occupied && dist < LOOKBACK) { //check lefter lane
by lookback distance
                            k = ((k-1)%LANESIZE < 0)?( (k-1)%LANESIZE + LANESIZE ):( (k-
1)%LANESIZE );
                            dist++;
                        }

                        if (dist == LOOKBACK && P_PASS_FAST > rand()%100) {
                            freeway[i+1][j].velocity = freeway[i][j].velocity;
                            freeway[i+1][j].passReset = PASSRESET;
                            freeway[i+1][j].occupied = true;

                            passRightFreq++;

                            freeway[i][j].occupied = false;
                            freeway[i][j].passReset = 0;
                            freeway[i][j].velocity = 0;
                        }
                    }
                }
            }
        }
    }
}
}
else if (RH_RULE) { //if rh rule is being used
    //Step 1.1: Passing Phase Left
    if (LANE > 1 && P_PASS_L > 0) { //if there are multiple lanes
        for(i = 1; i < LANE; i++) { //begin on the second to leftmost lane and proceed to the right
most lane
            for(j = 0; j < LANESIZE; j++){
                if (freeway[i][j].occupied && freeway[i][j].passReset == 0) {
                    dist = 0;
                    k = (j+1)%LANESIZE;
                    while(!freeway[i][k].occupied && dist < freeway[i][j].velocity) { //determine gap
                        k = (k+1)%LANESIZE;
                        dist++;
                    }
                    if (dist < freeway[i][j].velocity) { //if needs to slow down
                        dist = 0;

                        k = (j+1)%LANESIZE;
                        while(!freeway[i-1][k].occupied && dist < freeway[i][j].velocity) { //determine
gap
                            k = (k+1)%LANESIZE;
                        dist++;
                    }

                    if (dist == freeway[i][j].velocity) {
                        dist = 0;
                        k = j;
                        while(!freeway[i-1][k].occupied && dist < LOOKBACK) { //check lefter lane
by lookback distance
                            k = ((k-1)%LANESIZE < 0)?( (k-1)%LANESIZE + LANESIZE ):( (k-
1)%LANESIZE );
                            dist++;
                        }

                        if (dist == LOOKBACK && passLeftVector[freeway[i][j].velocity] >=
rand()%100) {

                            freeway[i-1][j].velocity = freeway[i][j].velocity;
                            freeway[i-1][j].passReset = PASSRESET;
                            freeway[i-1][j].occupied = true;

                            passLeftFreq++;

                            freeway[i][j].occupied = false;
                            freeway[i][j].passReset = 0;
                            freeway[i][j].velocity = 0;
                        }
                    }
                }
            }
        }
    }
}
//Step 1.2: Passing Phase Right
if (LANE > 1 && P_PASS_R > 0) { //if there are multiple lanes
    for(i = LANE-2; i >= 0; i--) { //begin on the second to rightmost lane and proceed to the
leftmost lane.
        for(j = 0; j < LANESIZE; j++) {
            if(freeway[i][j].occupied && freeway[i][j].passReset == 0) { //if hasn't passed in a while
                dist = 0;
                k=j;
                while(!freeway[i+1][k].occupied && dist < LOOKBACK) {
                    k = ((k-1)%LANESIZE < 0)?( (k-1)%LANESIZE + LANESIZE ):( (k-
1)%LANESIZE );
                    dist++;
                }
                if(dist == LOOKBACK && P_PASS_R >= rand()%100) {
                    freeway[i+1][j].velocity = freeway[i][j].velocity;
                    freeway[i+1][j].passReset = PASSRESET;
                    freeway[i+1][j].occupied = true;
```

```
                        passRightFreq++;

                        freeway[i][j].occupied = false;
                        freeway[i][j].passReset = 0;
                        freeway[i][j].velocity = 0;
                    }
                }
            }
        }
    }
}
else if (NO_RULE) {
    if (LANE > 1 && P_PASS_NO > 0) { //if there are multiple lanes
        //pass to the left phase
        for(i = 1; i < LANE; i++) { //begin on the second to leftmost lane and proceed to the right
most lane
            for(j = 0; j < LANESIZE; j++){
                if (freeway[i][j].occupied && freeway[i][j].passReset == 0) {
                    dist = 0;
                    k = j;

                    while(!freeway[i-1][k].occupied && dist < LOOKBACK) { //check lefter lane by
lookback distance
                        k = ((k-1)%LANESIZE < 0)?( (k-1)%LANESIZE + LANESIZE ):( (k-
1)%LANESIZE );
                        dist++;
                    }

                    if (dist == LOOKBACK && P_PASS_NO >= rand()%100) {
                        freeway[i-1][j].velocity = freeway[i][j].velocity;
                        freeway[i-1][j].passReset = PASSRESET;
                        freeway[i-1][j].occupied = true;

                        passLeftFreq++;

                        freeway[i][j].occupied = false;
                        freeway[i][j].passReset = 0;
                        freeway[i][j].velocity = 0;
                    }
                }
            }
        }
        //pass to the right phase
        for(i = LANE-2; i >= 0; i--) { //begin on the second to leftmost lane and proceed to the right
most lane
            for(j = 0; j < LANESIZE; j++){
                if (freeway[i][j].occupied && freeway[i][j].passReset == 0) {
                    dist = 0;
                    k = j;

                    while(!freeway[i+1][k].occupied && dist < LOOKBACK) { //check lefter lane by
lookback distance
                       k = ((k-1)%LANESIZE < 0)?( (k-1)%LANESIZE + LANESIZE ):( (k-
1)%LANESIZE );
                       dist++;
                    }

                    if (dist == LOOKBACK && P_PASS_NO >= rand()%100) {
                       freeway[i+1][j].velocity = freeway[i][j].velocity;
                       freeway[i+1][j].passReset = PASSRESET;
                       freeway[i+1][j].occupied = true;

                       passRightFreq++;

                       freeway[i][j].occupied = false;
                       freeway[i][j].passReset = 0;
                       freeway[i][j].velocity = 0;
                    }
                 }
              }
           }
        }
     }
     else {}




     //Step 2: Slow down if some one is in the way
     for(i = 0; i < LANE; i++){
        for(j = 0; j < LANESIZE; j++){
           if(freeway[i][j].occupied){

              dist = 0;
              k = (j + 1)%LANESIZE;
              while(!freeway[i][k].occupied && dist < freeway[i][j].velocity) {
                 k = (k+1)%LANESIZE;
                 dist++;
              }

              //for (k = j+1; !freeway[i][k].occupied; k = (k + 1)%LANESIZE )
              //    dist++;


              freeway[i][j].velocity = (int)fmin(freeway[i][j].velocity, dist);
        }
    }
}




//Step 3: Randomization of slowing down
for(i = 0; i < LANE; i++){
    for(j = 0; j < LANESIZE; j++){
        if(freeway[i][j].occupied && freeway[i][j].velocity > 0){
            if(P_SLOWING > ( rand() % 100) ){
                freeway[i][j].velocity--;
            }
        }
    }
}




int vel;
int numCars;
//Step 4: Update Position
for(i = 0; i < LANE ; i++){
    numCars = 0;
    for (j = 0; j < LANESIZE; j++) {
        if (freeway[i][j].occupied) {
            numCars++;
        }
    }
    j = 0;
    while(numCars > 0){
        if(!freeway[i][j].occupied) j = (j+1)%LANESIZE;
        else if(freeway[i][j].velocity > 0){  //occupied
            vel = freeway[i][j].velocity;

            freeway[i][ (j+vel)%LANESIZE ].velocity = freeway[i][j].velocity;
            freeway[i][ (j+vel)%LANESIZE ].passReset = freeway[i][j].passReset;
            freeway[i][ (j+vel)%LANESIZE ].occupied = true;


            freeway[i][j].occupied = false;
            freeway[i][j].passReset = 0;
            freeway[i][j].velocity = 0;
            j=(j+vel+1)%LANESIZE;
            numCars--;
         }
         else {//occupied but velocity 0
            j = (j+1)%LANESIZE;
            numCars--;
         }
      }
   }

   //Step 5: Statistics
   //calc density and average velocity
   if (CYCLES - runCycles > START_METRICS) {
      int velocity = 0;
      for (i = 0; i < LANE; i++) {
         for (j = LANESIZE/2-MAX_VEL; j < LANESIZE/2+MAX_VEL; j++) {
            if (freeway[i][j].occupied)
               velocity += freeway[i][j].velocity;
         }
      }

      double flowRate = (float)velocity / (LANE * 2 * MAX_VEL);
      avgFlowRate += flowRate;

      //calculate global average velocity
      velocity = 0;
      int numberCars = 0;
      for (i = 0; i < LANE; i++) {
         for (j = 0; j < LANESIZE; j++) {
            if (freeway[i][j].occupied) {
               velocity += freeway[i][j].velocity;
               numberCars++;
               density[i]++;
               velSumSquare += freeway[i][j].velocity*freeway[i][j].velocity;
               velSquareSum += freeway[i][j].velocity;
               populationSize++;
            }
         }
      }
      avgVelocityPerCycle[CYCLES-runCycles] = (float)velocity / numberCars;
      avgVelocity += (((float)velocity) / numberCars);
      // passLeftFreq += passFreqLeft[CYCLES-runCycles];
      // passRightFreq += passFreqRight[CYCLES - runCycles];
   }

   ////////////////////////////////////////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
    //print to file
    for(i = 0; i < LANE; i++){
        for(j = 0; j < LANESIZE; j++){
            f << freeway[i][j].occupied;
            f << " ";
            f << freeway[i][j].velocity;
            f << " ";
        }
        f << endl;
    }
    ////////////////////////////////////////////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////////////////////////////////////////////////
}

////////////////////////////////////////////////////////////////////////////////////////////////////
//Calculations

//Calculates the population variance for velocities over all cycles and all vehicles
velSquareSum = velSquareSum * velSquareSum / populationSize;
variance = (velSumSquare - velSquareSum) / populationSize;

//Calculates the right and left shift frequency
passRightFreq /= (CYCLES - START_METRICS);
passLeftFreq /= (CYCLES -START_METRICS);

//Calculates the flowrate and the average cycle-velocity across all cycles
avgFlowRate /= (CYCLES - START_METRICS);
avgVelocity /= (CYCLES - START_METRICS);

//Calculates the density of each lane
for (i =0; i < LANE; i++) {
    density[i] /= (float)((LANESIZE)*(CYCLES - START_METRICS));
}

//Calculates the system's variance in velocities for all vehicles at all cycles
float varianceVel = 0;
for (i = 0; i < CYCLES-START_METRICS; i++) {
    varianceVel += (avgVelocityPerCycle[i] - avgVelocity)*(avgVelocityPerCycle[i] - avgVelocity);
}
varianceVel /= (float)(CYCLES-START_METRICS);


if (RH_RULE)
    cout << "RH Rule" << endl;
else if(NO_RULE) {
    cout << "No Rule" << endl;
} else {
        cout << "Fast Rule" << endl;
    }
    cout << "Spawn Rate " << P_SPAWNING << endl;
    //cout << "Slow Rate " << P_SLOWING << endl;
    //cout << "Num Lanes " << LANE << endl;
    //cout << "Lane Length " << LANESIZE << endl;
    //cout << "Num Cycles " << CYCLES << endl;
    //cout << "Max Velocity " << MAX_VEL << endl;
    cout << "Look back " << LOOKBACK << endl;
    cout << "Pass Reset " << PASSRESET << endl;
    if (RH_RULE) {
        //cout << "Pass Left " << P_PASS_L << endl;
        //cout << "Pass Right " << P_PASS_R << endl;
    } else if (NO_RULE) {
        //cout << "Pass rate " << P_PASS_NO << endl;
    } else {
        //cout << "Pass rate " << P_PASS_FAST << endl;
    }
    cout << "Average Flow rate " << avgFlowRate << endl;
    cout << "Average Velocity " << avgVelocity << endl;
    //cout << "Variance of Velocity " << varianceVel << endl;
    //cout << "Standard Deviation of Cycle Average Velocity " << sqrt(varianceVel) << endl;
    // cout << "Population velocity Average " << variance << endl;
    cout << "Standard Deviation of Velocities " << sqrt((double)variance) << endl;

    //cout << "Pass Right Freq " << passRightFreq << endl;
    //cout << "Pass Left Freq " << passLeftFreq << endl;
    cout << "Pass Freq " << passRightFreq + passLeftFreq << endl;
    for (i = 0; i < LANE; i++) {
        cout << "Density Lane " << i << " " << density[i] << endl;
    }

    f.close();
    return 0;
}
```