Summary

Sam:

In our investigation of the endling lane, we devised a optimization model to suggest possible improvements on the traffic conditions that could effectively eliminate the problem of road rage. To provide context for optimization, we proposed two models to analyze the traffic conditions: a **microscopic** model to analyze the drivers' behaviors and their actions and a **macroscopic** formula to evaluate the overall traffic conditions to determine the extent of optimization.

We first designed the microscopic model that updates the drivers' positions after each loop in the computer simulation program. The updating process is divided into two parts: changing position due to change in lanes and due to velocities of cars. In our model, we analyze the **geometric and kinematic** conditions that must be satisfied by a car to change lane, and determine the probability of changing lane by the approximation of **logistic function**. Based on the theoretical analysis, we created a computer program to simulate the process.

We then formulate an evaluation function to determine traffic condition on the road. Therefore, based on the evaluation function, we use **genetic algorithm** to optimize the traffic conditions by iterations of coefficients of probabilities and velocity that directly determine the driver's behaviors and thus the resultant traffic conditions.

According to the optimization of drivers' behaviors, we wrote a letter to the Director of the Department of Transportation to recommend a training course for

drivers to enhance their behaviors on the road. Although we have only investigated the situations required in the problem set, our computer simulation program can simulate every possible conditions of lanes by simply changing parameters.

Table of Contents

1. Introduction And Problem Restatement

1.1. Introduction

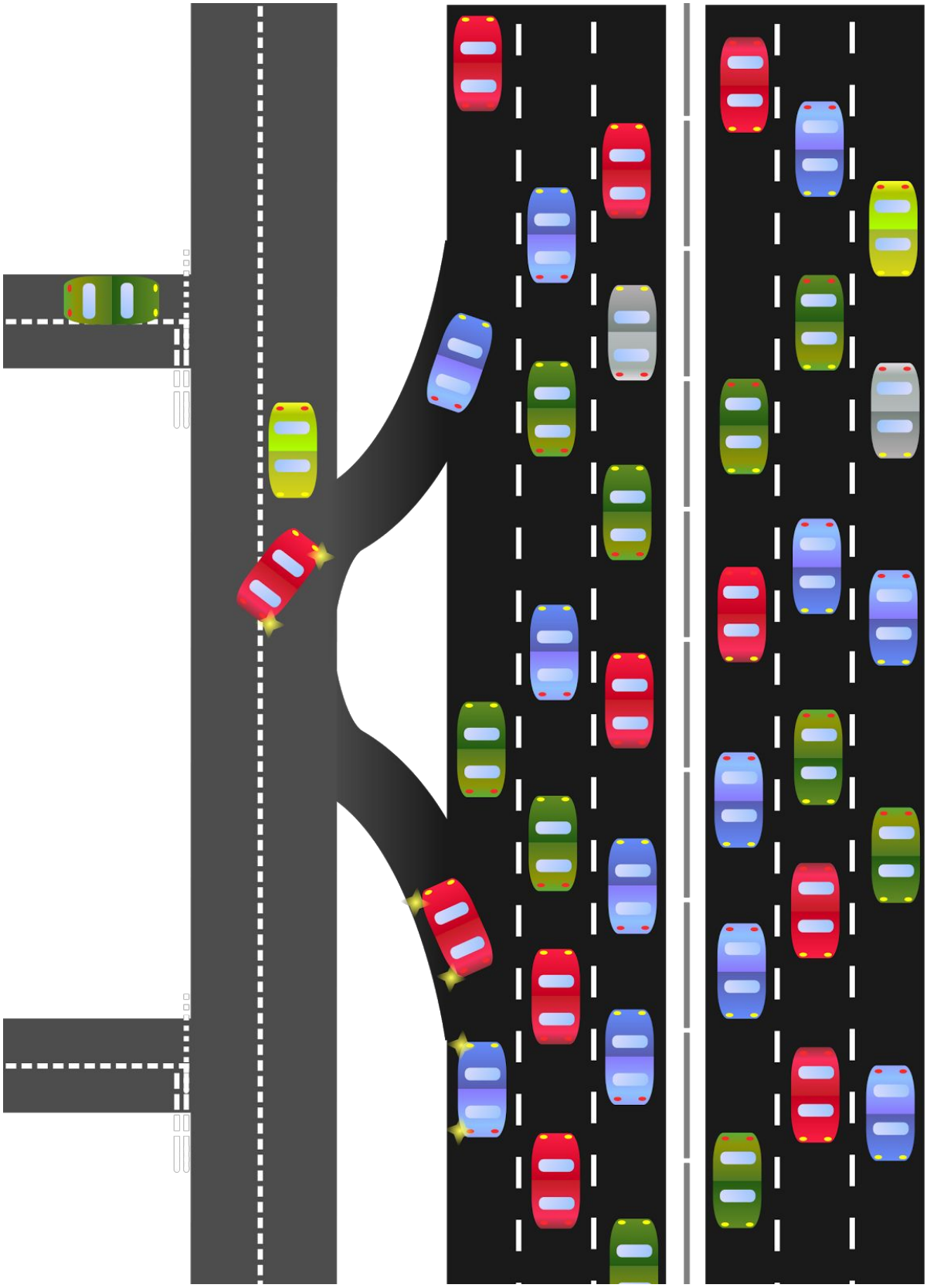
Freeways are important traffic infrastructures in urban and suburban area. However, traffic congestion is reported to be more and more severe in primary and secondary roads. The problem is most obvious when a lane of the road reaches an end and all driver on that try to change their lane. When there is a lane closure, multiple lanes will combine and there will be fewer numbers of lanes. For example, two lanes become one lane when closure occurs. However, in order to prevent serious traffic congestion, Department of Transportation devise a useful solution that there are always signals to remind the drivers of taking an early action to change lane into those common lanes, those which would not end. However, there are still problems. In spite of such hard works those decision makers have paid, the traffic system is not optimized still. Sometimes the situation becomes too chaotic because drivers' performance is irrational. These bad behaviors of drivers are known as "Road Rage".

We can all imagine a busy bottleneck section road with very few lanes. Many irritable drivers anxiously honk horns and frequently change lanes out of their own interest, trying to leave the chaotic road faster. Every individual tries to maximize their own interest. But just as the game theory states, if all the drivers try to maximize their individual interest, without considering the situation rationally, their total interest decreases. Their actions can harm the safety and efficiency of the road. Only when all the drivers consider rationally about the condition, the interest

can be maximized. Therefore, in our model, we assume that drivers are educated to be rational so that we can further optimize the traffic condition. That is goal of our model.

https://upload.wikimedia.org/wikipedia/commons/2/24/Trafficjamdelhi.jpg

https://en.wikipedia.org/wiki/Frontage_road



http://www.panoramio.com/photo/60589013

1.2. Problem Restatement

We aimed to solve the problem of road rage in a more fundamental way: improving the traffic conditions to prevent driver's dissatisfaction. We furnished our definitions for "fairness", "efficiency", and "evenhanded" criterions, optimized the driver behavior under the requirement of "fairness", and "efficiency" respectively and compared the macroscopic flow of traffic to that under the requirement of "evenhanded". We then simulated and optimized the traffic conditions with varying lanes configurations or maximum speed. Finally, we determined the best strategy for drivers and proposed the improvement plan to the Department of Transportation.

2. Assumptions and Basic Analysis

2.1. Assumptions

Assumption and Analysis

Assumptions and Justification

1. All the vehicles are identical.

   Justification: Compare to the whole lane, the shape of all the vehicles can be estimated to a fixed value. Therefore we assume that they have the same length and width.

2. After the driver decided to change lane, cars shift immediately to the target position.

   Justification: To simplify our microscopic model, we ignore the time for cars to change between lanes. When drivers are changing lanes, they will change as fast as they can to prevent traffic congestion because only when they finish the process their cars begin to move normally.

3. We assume that when a car changes lane, it turns and enters the target lane at an angle of 45 degree.

   Justification: The behavior of changing lane has a common pattern. Therefore, the rotation angle can be considered as a constant. In order to simplify our assumption, we adopt 45 degree as the angle.

4. Besides the analysis of irrational driver behavior in part 1, we assume that all the drivers are rational.

Justification: We admit that some drivers may have irrational behavior, however, in our model, those irrational drivers are eliminated by the model because their rude behaviors can be modified by education.

5. The only motivation for drivers on common lane is faster velocity.

   Justification: All drivers are rational. So they will maximize their efficiency.

6. A car can change to any speed in no time.

   Justification: The speed of the car is only influenced by the factors we study and introduced about in the following passage. The acceleration and deceleration of the car is ignored.

7. Drivers can judge whether his choice is rational.

   Justification: Drivers can judge the situation by their experience. This assumption further simplifies our model.

8. All lanes have same width.

   Justification: All the highways follow a fixed scale when they were built. Therefore, lane's width is identical.

9. Roads are flat with no height difference.

   Justification: Freeways are often flat to be safe for cars with different velocities. The inclination of roads cannot be very large. So we ignore it to apply model easier.

10. The scale of car includes the safety distance.

    Justification: Driver would not drive too close to a car in front of it. This assumption make sure cars do not collide with others after change in lane.

11. No unexpected and accidental conditions that interrupts the normal car fluid such as extreme weather, or disasters.

Justification: These unexpected conditions will completely destroy the traffic system. Any attempts to avoid them such as optimization of traffic rules are useless.

3. Model

3.1. Microscopic Analysis of Driver Behavior

3.1.1. Definitions and Symbols

Equations:

$$P_1 = k_{P,d}\left(\frac{2}{1+e^{-\lambda_P \Delta d}} - 1\right) \quad P_2 = k_{P,d}\left(\frac{2}{1+e^{-\lambda_P \Delta d}} - 1\right) + k_{P,t}\frac{1}{1+e^{\lambda_{P,t}}}$$

$$v = v_{max}\left[k_V\left(\frac{2}{1+e^{-\lambda_{v,d}(d-l-d_s)}} - 1\right) + (1-k_V)\frac{1}{1+e^{\lambda_{v,\rho\rho}}}\right]$$

Table: Variables

Symbol	Type	Meaning
Δd	Probability	Scale probability of changing lane due to Δd
t		Minimum distance between cars for safety
d	Velocity	Distance of two centers of cars
l		Length of a car
d_s		Minimum distance between cars for safety

Table: Constants

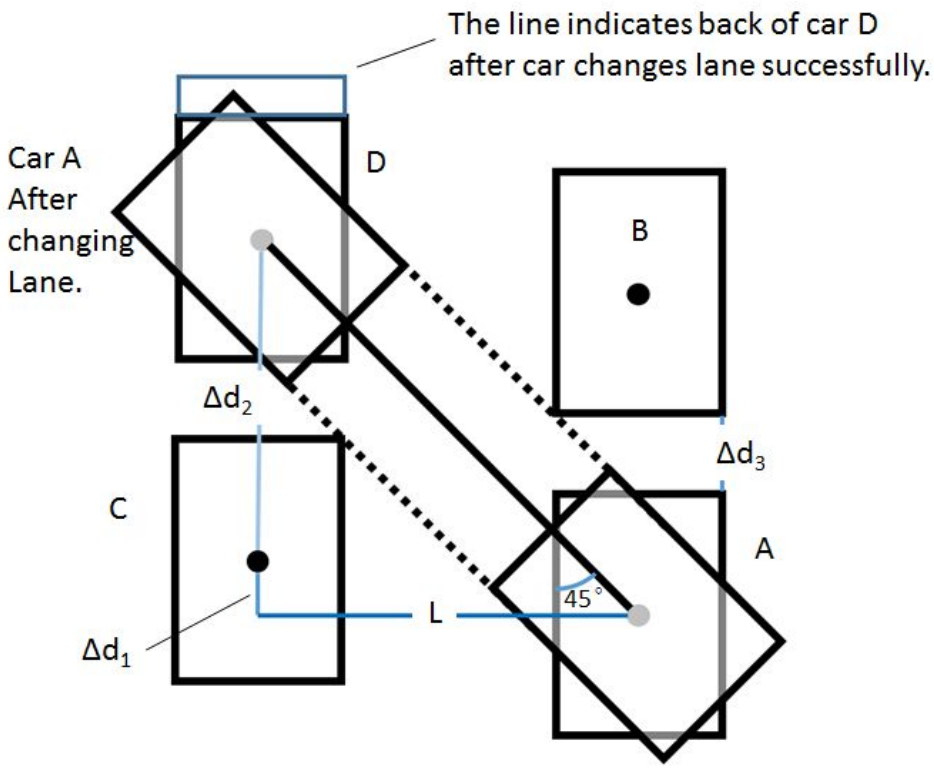
Symbol	Type	Usage
$k_{P,d}$	Probability	Scale probability of changing lane due to Δd
$k_{P,t}$		Scale probability of changing lane due to t
$\lambda_{P,d}$		Scale Δd due to difference in units
$\lambda_{P,t}$		Scale t due to difference in units
k_V	Velocity	Scale the weights of determinants: d and ρ
$\lambda_{v,d}$		Scale d due to difference in units
$\lambda_{v,\rho}$		Scale ρ due to difference in units

3.1.2. Changing Lane

Sam Hu:

Geometric deduction of changing lanes

For the simplification of our analysis, we only select four cars that may be involved in the changing lane problem. The graph of cars A, B, C, and D is shown below.



We assume, without losing generalization, the Car A, positioned after Car B, is about to change lane to its left before Car C and after Car D. The graph shown on the top showed the minimum requirement for avoiding collision in the four-car system.

Car A, rotating around its geometric center of an angle 45º, shifts to the expected position on the other lane and rotates 45º in the opposite direction. This process of changing lane is assumed to finish instantaneously.

In this part of deduction, we will find three minimum distances Δd_1, Δd_2, and Δd_3, defined with the illustration of graph below, that makes possible for Car A to change lane without collision.

[Graph 1 should be put here.]

To assist the geometric deduction, some assistive lines are drawn on the graph. Two lines starting from center O_1 are perpendicular to the length side of the car, with feet H_1 and H_2. A line starting from center O_2 is perpendicular to the length side of the car with foot H_3.

A line passing O_1 is parallel to the width side of cars and intersect at B with the line that passes O_2 and parallel to the length side of the cars. Line O_1B intersects the inclined length side at A. Line DH_3 extends to intersect the width side at H_4, and O_1B at C. Finally, we define $\Delta d_1 = O_2B$.

Since Car A rotates around its central point of angle 45º, there are two isosceles right triangles $\triangle ACD$ and $\triangle O_1AH_2$. Therefore, $AC = CD$, $\sqrt{2}OH_2 = AO_1$. Since

$CD = DH_3 + H_3C = \frac{1}{2}l + \Delta d_1$ and

$AC = BO_1 - BC - AO_1 = L - \frac{1}{2}w - \sqrt{2}\frac{1}{2}w = L - \frac{\sqrt{2}+1}{2}w$,

$\frac{1}{2}l + \Delta d_1 = L - \frac{\sqrt{2}+1}{2}w \Rightarrow \Delta d_1 = L - \frac{\sqrt{2}+1}{2}w - \frac{1}{2}l$.

[Graph 2 should be put here.]

Points O_2, O_1', O_3 are on the same line perpendicular to line O_1B. It is also obvious that $\triangle BO_1O_1'$ is also an isosceles triangle. We define $\Delta d_2 = O_2O_3$.

Since $\triangle BO_1O_1'$ is an isosceles triangle, $O_1B = O_1'B = L$.

Since $O_1'O_2 = O_1'B - O_2B = L - \Delta d_1$,

$\Delta d_2 = O_2O_3 = O_1'O_2 + O_1'O_3 = L - \Delta d_1 + l = L - L + \frac{\sqrt{2}+1}{2}w + \frac{1}{2}l + l = \frac{\sqrt{2}+1}{2}w + \frac{3}{2}l.$

[Graph 3 should be put here.]

Points E, F are two edges of Car A and Car D, and G is the intersection point of a width side and length side of Car A. $\triangle EFG$ is also an isosceles triangle. We define $\Delta d_3 = EF$.

For small width and length of a car, G is approximately the midpoint of the width side. Therefore, $\Delta d_3 = EF = FG = \frac{1}{2}w$.

The result of our deduction is summarized below.

$\Delta d_1 = L - \frac{\sqrt{2}+1}{2}w - \frac{1}{2}l$;

$\Delta d_2 = \frac{\sqrt{2}+1}{2}w + \frac{3}{2}l$;

$\Delta d_3 = \frac{1}{2}w$.

Tim and Calin:

The possibility for drivers to change lanes：

For common lane: $P_1 = k_{P,d}(\frac{2}{1+e^{-\lambda_P \Delta d}} - 1)$

For ending lane: $P_2 = k_{P,d}(\frac{2}{1+e^{-\lambda_P \Delta d}} - 1) + k_{P,t}\frac{1}{1+e^{\lambda_{P,t}}}$

We used combination of two logistic functions with different coefficients to determine drivers' probability to change lane. In common lane which is not ending, we assume that the drivers change lane only because they can drive faster. We find that the relation between probability to change lane and change in velocity mostly satisfies the logistic function where at two ends change in probability is gradual and in the middle the relation can be considered as roughly linear. So we get the formula for common lane: $P_1 = k_{P,d}(\frac{2}{1+e^{-\lambda_P \Delta d}} - 1)$. Besides common lane, we define the closed lane as the ending lane. We assume that probability is influenced by

increase in velocity, as well as the time to end of lane closure. Because a driver in ending lane must change to common lane before lane closure, thus closer to the end the more anxious they are. In this case, we use two constant to adjust the weight of two factors. The formula for the closed lane: $p = \frac{k1}{1+e^{-\Delta d}} + \frac{k2}{1+e^{\Delta t}}$.

3.1.3. Changing Velocity

3.1.2.1 Symbols and Definitions

Symbol	Definition
Δ d	distance from center to center of one car and car before it
ρ	density of car flow on the lane
V	velocity of one car after every single update
V_0	maximum speed limit of the lane
λ_1	coefficient of delta d, decided by genetic algorithm
λ_2	coefficient of rou, decided by genetic algorithm
k	coefficient of two logistic functions, decided by genetic algorithm

In order to further analyze drivers' performance, we set up a model to calculate instantaneous velocity of one car.

Based on our assumption, velocity of a individual car changes over time. This change, instead of random, is based on some key factors.

$$V = k\left(\frac{V0}{1-e^{-\lambda 1 \Delta d}}\right) + (1-k)\left(\frac{V0}{1+e^{\lambda 2 \rho}}\right)$$

3.1.4. Road Rage

In the real world situation, a driver may sometimes have irrational actions. Although these actions are harmful to all drivers, they seem to be unavoidable in our life. People are always self-centred.

On a freeway where one lane is reaching the end (ending lane), the drivers on that lane may not change their lane as soon as they see the warning sign. For psychological reason, some irritable drivers on the ending lane would try to drive as fast as the car beside them, which made them unable or forget to change the lane until they reach the very end of the lane. Also drivers on the lane just next to the ending one may try to drive faster to avoid cars on the right to change lane and drive in front of them. These behavior is understandable when we think about the 'fairness' of those driver. However, the efficiency of the whole road decreases and it is unfair for the drivers blocked by them.

In further analysis of our model, however, we assume all drivers are rational. The goal of our model is to optimize the efficiency and fairness of the traffic system. Those irrational behaviors, certainly cause model to be inefficient, can be easily modified by education to the rude drivers. Therefore, in our model we will use mathematical analysis to find optimization for rational situation so that efficiency and fairness as well can be further optimized.

3.2. Macroscopic Analysis of Real Situations

3.2.1. Introduction

In this part of the thesis, we will summarize various types of behavior of a passenger into a simulation program. Instead of designing different programs for different situations, we use **one** single program, with variable constants, including the arrangement and number of lanes, and maximum speed on the highway, to simulate all the possible situations required in and even beyond the problem set. We will run a simulation of each situation with some randomly determined driver behaviors to demonstrate the major problem in each situations, then use genetic algorithm to optimize 6 coefficients (introduced in 3.2.3. Definitions of Symbols), which determine the velocity and probability of changing lane of each driver. We use the evaluation function defined in 3.2.2. to evaluate the outcome of simulation and then try to minimize the output of the evaluation function by genetic algorithm to find the optimized driver behavior and overall macroscopic image of the situation.

3.2.2. Detailed Illustration of Simulation Program

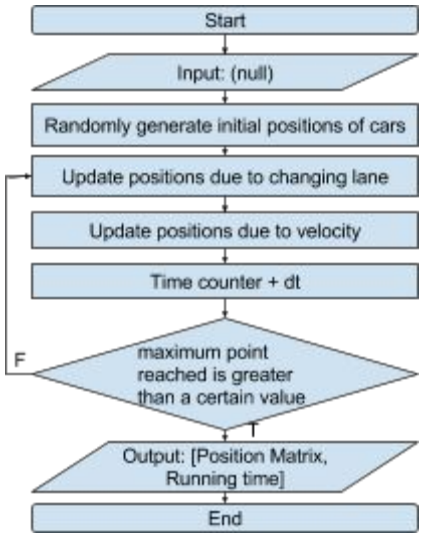
3.2.2.1. Introduction

The simulation program consists of two mode: visual mode (mode = 1) for debugging and demonstration and background mode (mode = 2) for data transition to genetic algorithm. These two modes are based on the common main function for

the simulation. The main function defines several sets of rules for the updating position of cars.

3.2.2.2. Mechanism of the Main Function for Simulation

The mechanism of the simulation program can be illustrated by the following flow diagram.



The function first initializes the matrix storing all the positions of cars. Then it enters a loop that updates the positions of cars according to the rules defined in 3.1. and stops when a first car drives a certain distance. After running the simulation, the function returns the matrix of positions and time (loops) required to reach to the final position.

3.2.3. Evaluation Function

Introduction and Restatement

density and velocity

When thinking about cars driving on a road, we draw connection between the car flow with the water flow. Our inspiration comes from the similarity between car flow and fluid dynamic. The road is like a vessel and cars can be seen as water molecules flow in the vessel. Several analogies are made to connect the physical model of fluid dynamics and the mathemical model of traffic conditions.

1. Pressure is most analogous to the danger of the road. From intuition, if pressure on the road is higher, then the road will be more crowded and chaotic, thus increasing the chance of accidient.
2. Density is most analogous to the closeness of the car on the road. From intuition, if density on the road is greater, then the road will be filled with more cars per unit length.
3. Velocity is most analogous to the average velocity of cars on the highway.

The most famous equation in fluid dynamic is the Bernoulli's equation. It is deduced from total work done by a fluid:

$$W = W_1 + W_2 + W_3$$

where W_1 is the work done by one layer of water in vessel: $W_1 = F_1 \Delta l_1 = P_1 A_1 \Delta l_1$

W_2 is the work done against first layer on the other part of vessel with different width:

$$W_2 = F_2 \Delta l_2 = - P_2 A_2 \Delta l_2$$

W_3 is the work done by gravity. This part is zero in our model because we assume our road is flat with no height difference.

So total work done by a fluid:

$$W = W_1 + W_2 = P_1 A_1 \Delta l_1 - P_2 A_2 \Delta l_2$$

According to work-energy principle in dynamic, net work done equals change in kinetic energy of the fluid:

$$\tfrac{1}{2} m v_2^2 - \tfrac{1}{2} m v_1^2 = P_1 A_1 \Delta l_1 - P_2 A_2 \Delta l_2$$

where u is the velocity at first layer and v is the velocity at second layer.

The volume of the fluid in regular vessel equals to cross sectional area multiplied by the length of it: $V = A\Delta l$

$$\tfrac{1}{2} m v_2^2 - \tfrac{1}{2} m v_1^2 = P_1 V_1 - P_2 V_2$$

$$\tfrac{1}{2}(v_2^2 - v_1^2) = \frac{P_1 V_1}{m} - \frac{P_2 V_2}{m}$$

$$\tfrac{1}{2}(v_2^2 - v_1^2) = \frac{P_1}{\rho_1} - \frac{P_2}{\rho_2}$$

$$v_2^2 - v_1^2 = \frac{2P_1}{\rho_1} - \frac{2P_2}{\rho_2}$$

$$\frac{2P_1}{\rho_1} + v_1^2 = \frac{2P_2}{\rho_2} + v_2^2$$

Since the equation indicates the equality of the combination of pressure, density and velocity, the equations could be further summarized into

$$\frac{2P}{\rho} + v^2 = c$$

$$P = c_1 \rho - c_2 \rho v^2$$

where c_1, c_2 are two positive constants

Inspired from the equation, we conclude that density ρ and negative square of velocity $-v^2$, are key determinants of the danger of the road. However, to evaluate the overall conditions on the road, we should consider factors, such as fairness and efficiency as well. The factor fairness has connections to the standard deviation of density since large discrepancy of the traffic conditions between lane would cause road rage; the factor efficiency has connection to velocity since the velocity

determines the time required for a driver to reach its destination; the factor danger has connection to density, and velocity, already in the above section, but also to standard deviation as well, since standard deviation is connected to road rage, which will increase the danger on the road.

To analyze the complex connections between all these qualitative criterions like fairness, efficiency and danger and quantitative data like density, velocity, and standard deviation of density, we use AHP (Analytical Hierarchy Process) to determine the assumed linear relation between those factors.

This equation connects the density and the velocity of water with the pressure in the vessel. Similarly, when cars driving on a road, the ‘pressure’ of the road can be deduced by the density and velocity of cars. We see the roads with higher ‘pressure’ are more dangerous. From the physical model of fluid dynamic, we can conclude that the road is more dangerous when the density of cars on the road is higher and the velocity is lower. Also velocity has greater effect since the degree of the velocity is two in the equation. Though it seems absurd that low velocity leads to danger, this conclusion is reasonable because we have assumed that velocity is proportional to the distance between two cars. Therefore, high velocity, meaning large distance, of the lane makes the traffic safer.

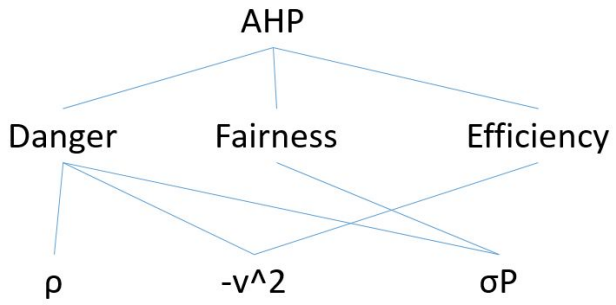
AHP

Analytic Hierarchy Process (AHP) is an approach to decision making that involves structuring multiple choice criteria into a hierarchy, assessing the relative importance of these criteria, comparing alternatives for each criterion, and determining an overall ranking of the alternatives.·

The access is based on our scores of each impact or project. It ranges from 1 to 9. The higher the score is, the more important the project is. For instance, if one impact is scored as 9, this impact is significant to the aim. Else if one is scored as 1, its impact to the aim is slight.

By organizing and assessing alternatives against a hierarchy of multifaceted objectives, AHP provides a proven, effective means to deal with complex decision making. In this model, we use AHP to allow a better, easier, and more efficient identification of factor criteria, their weights and analysis.

General AHP model consists of three hierarchies. The first hierarchy is the Aim Layer. It represents the final goal of the decision. The second hierarchy is Criterion Layer. It consists of some intuitive criterions of the traffic condition, including danger, unfairness, and inefficiency in our case. These criterions are closely related to Aim level and contribute to the aim to different extents. The third hierarchy is Alternatives Layer, It has some basic alternatives, including density, velocity, and standard deviation of density in our case, which share some criterions in the second hierarchy.



(Modification needed: AHP → Traffic condition, Fairness → Unfairness, Efficiency → Inefficiency)

A. Criteria Layer

The first step is to set up a comparison matrix by scoring each impact in each layer. We design this score by ourselves. In the criterion layer, the possible impacts are danger of the traffic, fairness and efficiency. Besides efficiency and fairness mentioned in the question, we add a new factor, danger, which relates to pressure of the lane if we consider the car flow as ideal fluid model. We also assume that danger is always the most important impact of traffic system because in a dangerous system, neither fairness nor efficiency can be high. Therefore, we score 'danger' impact as 7. However, we cannot decide the order of significance of fairness and efficiency. Both of the impacts are important. Therefore, we devise three groups of AHP model of efficiency-biased, fairness biased, and even-handed model. We let the score of the biased impact be 5 and the other be 3. In even-handed model, we let both efficiency and fairness be 5.

1. Efficiency biased model

Aim	Danger 1	Fairness 1	Efficiency 9
Danger 1	1	1	1/9
Fairness 1	1	1	1/9
Efficiency 9	9	9	1

2. Fairness biased model

Aim	Danger 1	Fairness 9	Efficiency 1
Danger 1	1	1/9	1
Fairness 9	9	1	9
Efficiency 1	1	1/9	1

3. Even-handed model

Aim	Danger 7	Fairness 5	Efficiency 5
Danger 7	1	7/3	7/5
Fairness 5	5/7	1	1
Efficiency 5	5/7	1	1

B. Alternative Layer

We use AHP model to evaluate the significance of the projects in this layer to the aim layer. But in this section we will first find the significance Here is the Alternative Layer:

ρ	the density of the lane
$-v^2$	negative square of velocity
σρ	standard deviation of density of each lane

Here we list three tables to show relatively the impact of three projects on every impacts.

1. B1: Danger

Danger	ρ 5	$-v^2$ 7	σρ 3
ρ 5	1	5/7	5/3
$-v^2$ 7	7/5	1	7/3
σρ 3	3/5	3/7	1

In the access of danger, we are convinced that all the three factors are related to this impact. We think that danger is related to pressure of car fluid and the higher the pressure is, the more potentially dangerous the traffic system is. Based on the fluid model, we have the formula: $p = c1\rho - c2\rho v^2$. density is positively proportional to pressure while square of velocity is negatively proportional to

pressure. Thus we give density 5 points and negative square of velocity 7 points. Besides these two projects, we also assume that deviation of density also affects the safety. If deviation of density, meaning the distribution of car density on each lane, is large, uneven distribution makes the fluid dangerous. Deviation is not closely related and we give it 3 points.

2. B2: Unfairness

Unfairness	$\sigma\rho$
$\sigma\rho$	1

Based on our definition of fairness, it is only related to deviation of density because more even distribution shows that the chance to change lane is equally distributed to every car driver all the time.

3. B3: Inefficiency

Efficiency	$-v^2$
$-v^2$	1

[Plus some calculations of AHP]

End your conclusion like:

We therefore find an even-handed evaluation function

$f \to 0.1527\rho + 0.4847(-v^2) + 0.3626\sigma$,

another biased to fairness $f \to 0.0303\rho + 0.1333(-v^2) + 0.8364\sigma$,

and the other biased towards efficiency $f \to 0.0303\rho + 0.4471(-v^2) + 0.1091\sigma$.

3.2.4. Applications And Results

According to the theoretical deduction introduced in 3.1., we would use the realistic computer simulation explained in 3.2.1. to 3.2.3 to simulate different scenarios. We divided the scenarios into three different categories: scenarios with different requirements, scenarios with different lanes conditions, and scenarios with different maximum allowed speed, which are discussed in Section 3.2.4.1., Section 3.2.4.2., and Section 3.4.2.3. respectively.

The variables that are kept constant or changed are summarized in the following table.

constants	k_{ρ}	k_{-v2}	k_{sigma}	n(lanes)	isEndingLane	v_{max}
changing requirements	changing			constant		
changing configuration of	constant			changing		constant

lanes			
changing maximum speed		constant	changing

The genetic algorithm, of average running time 20 minutes, requires a huge number of iterations; therefore, some measures must be taken to reduce the running time of the program.

The optimization programs are running on group members' computers simultaneously and only time to save time.

For seven constants $k_{P,d}$, $k_{P,t}$, $\lambda_{P,d}$, $\lambda_{P,t}$, k_V, $\lambda_{v,d}$, $\lambda_{v,\rho}$ mentioned in the beginning of Section 3, only five of the constants $k_{P,d}$, $k_{P,t}$, $\lambda_{P,d}$, $\lambda_{P,t}$, k_V are automatically determined by the program.

For constant k_V that scales the velocity, it must be big enough to enable the program to run fast enough. Therefore, we decomposed k_V into $k_V = 0.9 + k_V' \times 0.1$, where k_V' is the constant to be determined by the program.

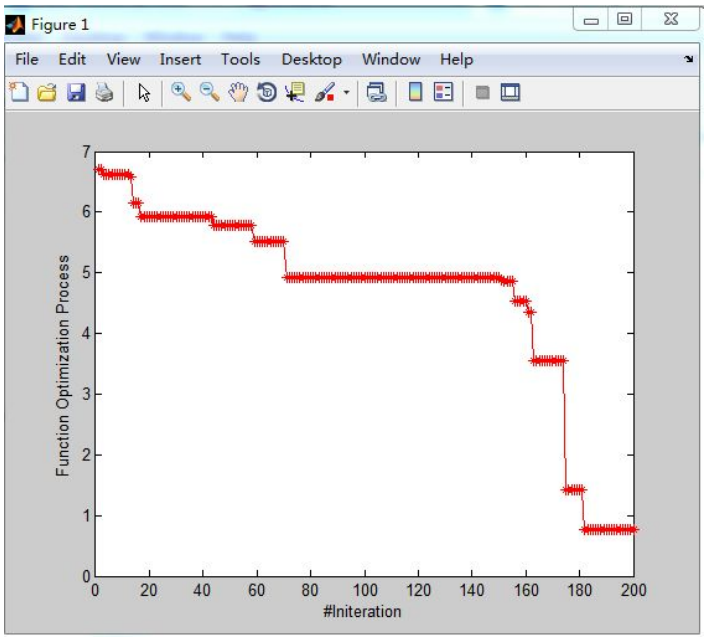
Constants $\lambda_{v,d}$ and $\lambda_{v,\rho}$ are set to 0.1 and 1 respectively to give the velocity due to distance between cars.

The number optimized in the program are integers between 0 to 1000, which would be divided by 1000 in the problem to obtain the coefficients needed in the simulation.

We run the optimization program with different configurations, and the results are summarized below.

Standard Condition: Requirement is evenhanded; maximum speed is 65 mph; three lanes merge into two lane.

Graph:



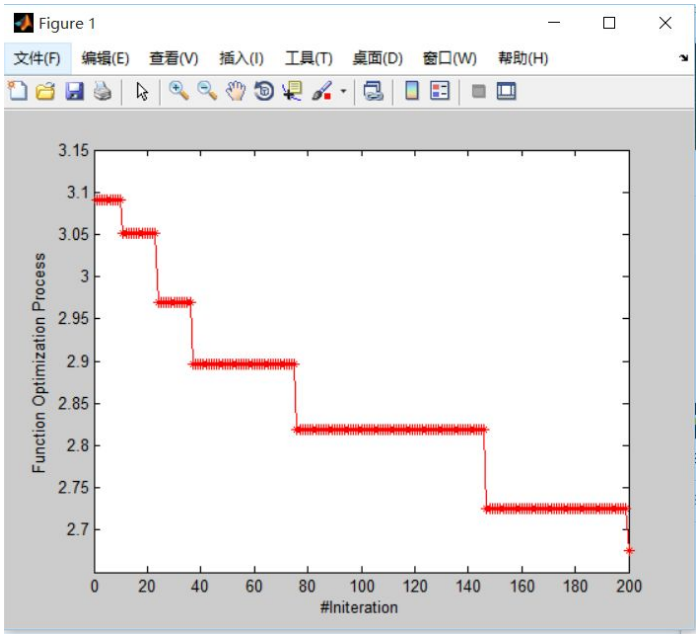
Data:

```
optimizedValue =

    0.7678


ans =

   190   611     7    80   869
```

Simulation:

Condition 1-2: Requirement is fairness; maximum speed is 65 mph; three lanes merge into two lane.

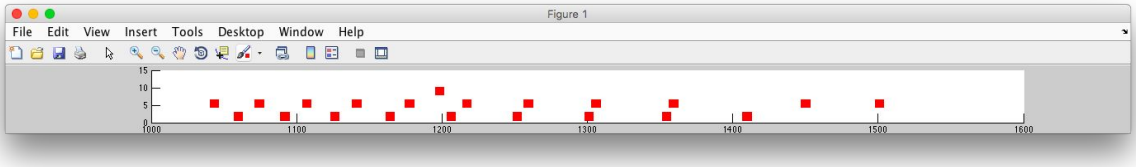
Graph:



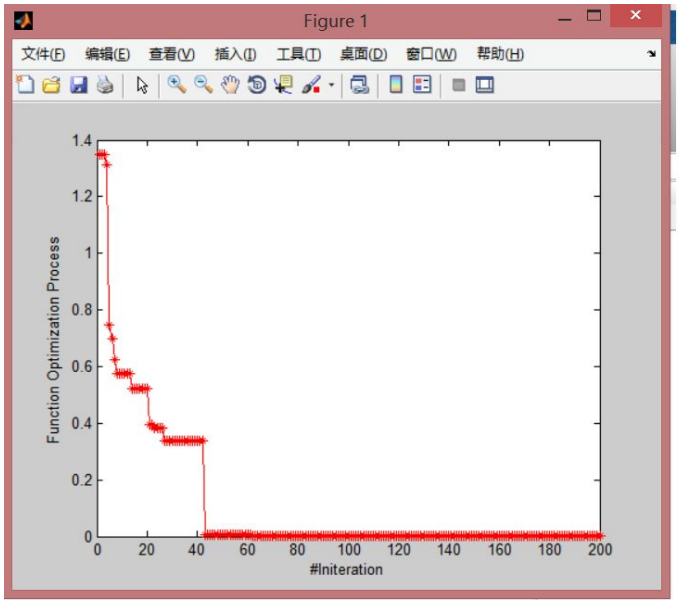
Data:

```
optimizedValue =

    2.6763


ans =

   145   392   370   414   689
```

Simulation:

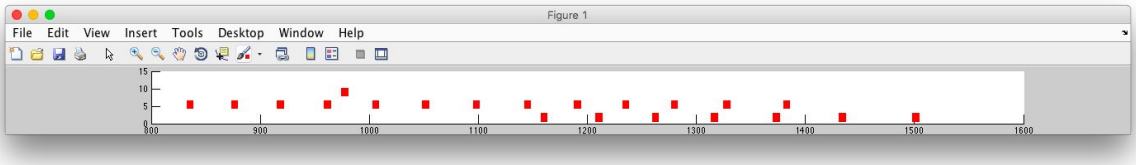


Condition 1-3: Requirement is efficiency; maximum speed is 65 mph; three lanes merge into two lane.

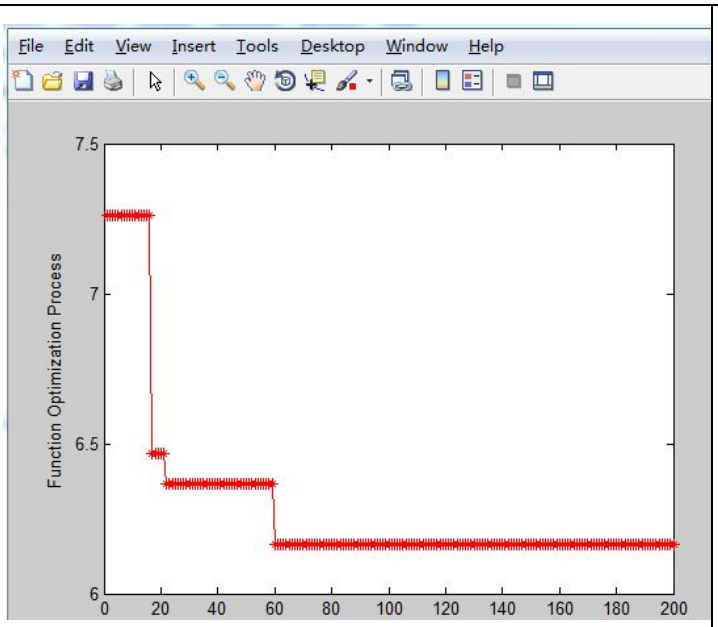



```
optimizedValue =

    0.0041

ans =

   109     7     0   942   639
```

Simulation:

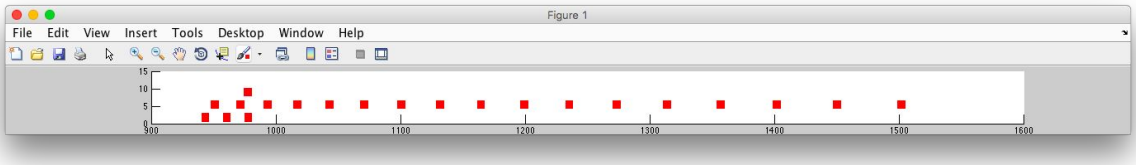


Condition 2-2: Requirement is evenhanded; maximum speed is 65 mph; three lanes merge into one lane.

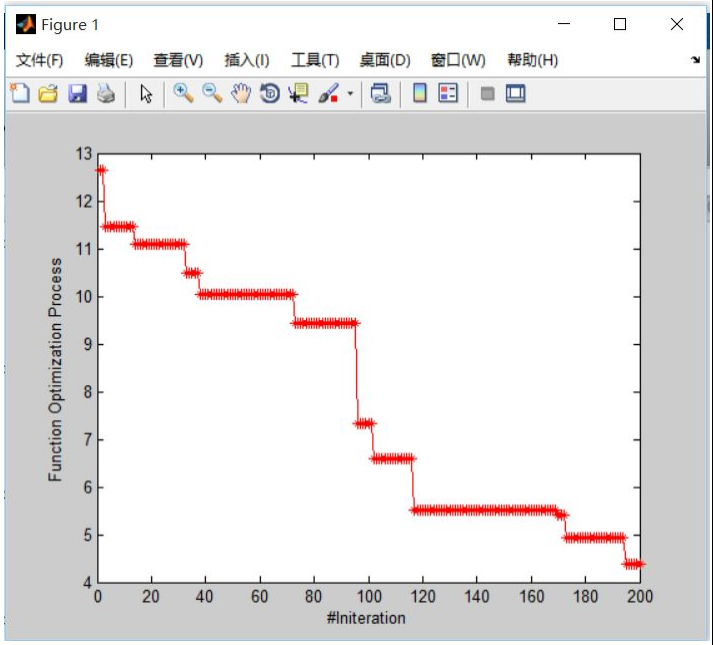



```
optimizedValue =

    6.1635


ans =

   750    63   631   607   381
```

Simulation:

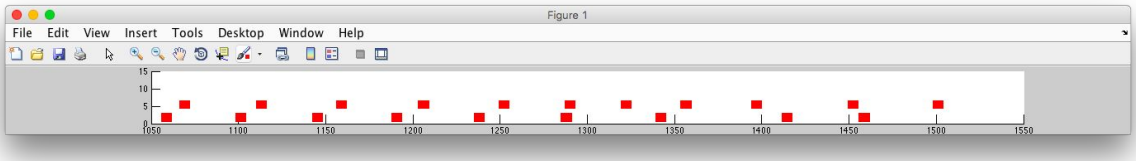


Condition 3-2: Requirement is evenhanded; maximum speed is 35 mph; three lanes merge into two lane.




```
optimizedValue =

    4.3840


ans =

     4   296   646    98   839
```

Simulation:



The rescaled coefficients are then summarized below in a table.

Type of Changing	Change	$k_{P,d}$	$k_{P,t}$	$\lambda_{P,d}$	$\lambda_{P,t}$	k_V'
Changing Requirement	Fairness	0.145	0.392	0.370	0.414	0.689
	Efficiency	0.109	0.007	0.000	0.942	0.639

	Evenhanded	0.190	0.611	0.007	0.080	0.869
Changing Configuration of Lane	Three lanes into two lanes	0.190	0.611	0.007	0.080	0.869
	Three lanes into one lane	0.750	0.063	0.631	0.607	0.381
Changing Maximum Speed	65 mph	0.190	0.611	0.007	0.080	0.869
	35 mph	0.004	0.296	0.646	0.098	0.839

Equations to determines their probability to change lane and velocity are summarized below.

Type of Changing	Change	Equations
Changing Requirement	Fairness	$P_1 = 0.145(\frac{2}{1+e^{-0.370\Delta d}} - 1)$ $P_2 = 0.145(\frac{2}{1+e^{-0.370\Delta d}} - 1) + 0.392\frac{1}{1+e^{0.942t}}$ $v = v_{max}[0.969(\frac{2}{1+e^{-0.1(d-l-d_S)}} - 1) + 0.031\frac{1}{1+e^{\rho}}]$
	Efficiency	$P_1 = 0$ $P_2 = 0.007\frac{1}{1+e^{0.942t}}$ $v = v_{max}[0.964(\frac{2}{1+e^{-0.1(d-l-d_S)}} - 1) + 0.036\frac{1}{1+e^{\rho}}]$

	Evenhanded	$P_1 = 0.190(\frac{2}{1+e^{-0.007\Delta d}} - 1)$ $P_2 = 0.190(\frac{2}{1+e^{-0.007\Delta d}} - 1) + 0.611\frac{1}{1+e^{0.080t}}$ $v = v_{max}[0.987(\frac{2}{1+e^{-0.1(d-l-d_s)}} - 1) + 0.013\frac{1}{1+e^{\rho}}]$
Changing Configuration of Lane	Three lanes into two lanes	$P_1 = 0.190(\frac{2}{1+e^{-0.007\Delta d}} - 1)$ $P_2 = 0.190(\frac{2}{1+e^{-0.007\Delta d}} - 1) + 0.611\frac{1}{1+e^{0.080t}}$ $v = v_{max}[0.987(\frac{2}{1+e^{-0.1(d-l-d_s)}} - 1) + 0.013\frac{1}{1+e^{\rho}}]$
	Three lanes into one lane	$P_1 = 0.750(\frac{2}{1+e^{-0.631\Delta d}} - 1)$ $P_2 = 0.750(\frac{2}{1+e^{-0.631\Delta d}} - 1) + 0.063\frac{1}{1+e^{0.607t}}$ $v = v_{max}[0.938(\frac{2}{1+e^{-0.1(d-l-d_s)}} - 1) + 0.062\frac{1}{1+e^{\rho}}]$
Changing Maximum Speed	65 mph	$P_1 = 0.190(\frac{2}{1+e^{-0.007\Delta d}} - 1)$ $P_2 = 0.190(\frac{2}{1+e^{-0.007\Delta d}} - 1) + 0.611\frac{1}{1+e^{0.080t}}$ $v = v_{max}[0.987(\frac{2}{1+e^{-0.1(d-l-d_s)}} - 1) + 0.013\frac{1}{1+e^{\rho}}]$
	35 mph	$P_1 = 0.004(\frac{2}{1+e^{-0.646\Delta d}} - 1)$ $P_2 = 0.004(\frac{2}{1+e^{-0.646\Delta d}} - 1) + 0.296\frac{1}{1+e^{0.098t}}$ $v = v_{max}[0.984(\frac{2}{1+e^{-0.1(d-l-d_s)}} - 1) + 0.016\frac{1}{1+e^{\rho}}]$

3.2.4.1. Changing Requirement

In the simulation, we let the speed of car be 65mph and the bottle head section road change for 3 lanes to 2. We assume the optimized situation is the evenhanded

condition and use the genetic algorithm to run the three situations we mentioned in 3.2.3.

From the results, we can see that in the ‘Efficiency’ situation, there is no possibility for cars on normal lanes to change and only very few cars on the ending lane can change. In this situation, the car on the ending lane sacrifice their efficiency to cross the bottle neck road to maximize the efficiency of whole road. This is really unfair for the drivers on ending lane and is unlikely to happen in real world.

In the ‘fairness’ situation, the possibility for all lanes to change increase significantly compared to ‘efficiency’. Also, the scalar of time factor for the drivers on the ending lane is very high (0.942), which means the cars on the ending lane would change lane very early. Drivers on the ending lane would very likely to shift as soon as they see the sign to create a fair environment for all drivers.

In the evenhanded situation, the possibility for drivers on normal lane to change lane is greater than efficiency situation. On the other hand, the scalar of time factor is lower than that of the fairness situation. Drivers on the ending would neither change lane very unlikely nor change lane very readily. The evenhanded situation is optimized to take into account both characteristic of ‘efficiency’ and ‘fairness’.

3.2.4.2. Changing Configuration of Lanes

In this simulation, we compare the relationship between probability of changing lanes and number of final lanes. We list out two conditions: three to two

and three to one. Before evaluating the result, we need to mention prerequisites of this simulation: In this part, changing requirement is even handed, which means it finds a balance between efficiency and fairness.

1. P2

According to functions of P2, it is obvious that probability of cars on ending lanes to change lanes in three to one condition is much bigger than cars in three to two conditions. This is quite reasonable because for cars on ending lanes in three to one condition must change lanes as soon as possible for lanes will end and there is just one common lane.

2. P1

As P1 is determined by P2, so P1 of three to one condition is bigger than P1 of three to two conditions

3.Velocity

Maximum velocity of three to two condition is higher than maximum velocity of three to one condition according to formula we get. This is also reasonable, for three to one conditions, probability of changing lane is much higher, which will reduce maximum speed of cars.

3.2.4.3. Changing Maximum Speed

The most apparent difference between two situations is the coefficients of logistic functions and coefficients of variables. We let k be coefficient of logistic function and c be coefficient of variables. In situation 1 with 65 mph limit, k_1, equal to 0.19, is far larger than k_2 in situation2, which equals to 0.004. As k represents the maximum probability of changing lanes, based on the data from genetic algorithm, when maximum speed limit is higher, total probability is far

higher. Especially when logistic reaches a maximum value, for instance, the probability for every car in situation 1 to change lane is 19%, about 2.4 cars will change lane in one second, with 0.08 second as the basic unit of update. However, in situation 2, even when probability reaches its maximum, probability is still 0.4% only, representing that 0.05 cars change per second. Indeed, maximum speed impact so much. Because we assume that velocity of cars are proportional to the distance between cars, and since cars will try to reach highest level of velocity, which is the speed limit, to achieve most interest, a high speed restriction leads to a high level of velocity and thus means farther distance between cars. Since the farther cars are, the easier to change lanes, probability is largely depended on the maximum speed limit.

However, the magnitude of coefficient of variable c is negatively proportional to speed limit. Lower limit leads to more negative value of c. The value of c represents how fast probability changes with variables. When speed limit is low in situation 2, average velocity of cars is low and distance between cars are short. Therefore, in this case, change in distance between cars is relatively more significance, while the denominator, distance, is small, compared to situation 1 where denominator is large. Therefore, in situation 2, only a small change in distance can make the probability increase fast. Mention that the change is **percentage change of probability**. However, considering its extremely low total probability, though p changes with distance change fast, the **magnitude of probability change** is still too low to be significant.

In spite of this great difference of probability related to change in distance changing with maximum speed limit, the probability related to distance from a car on ending lane to end of lane does not vary much with changing speed limit. The

reason is because the car must change lane when it comes to end of lane closure. The gradual change in probability with time also suggests that the probability related to distance to ending is relatively inelastic.

3.3. Implications of the Results

Neither “fairness” nor “efficiency” is a perfect indicator for the traffic conditions on the highway. Overly emphasis on “fairness” would result in frequent changing in lanes, thus increasing the probability of collision on the road, while overly emphasis on “efficiency” would result in the congestion at the end of the ending lane. The emphasis will significantly increase the dissatisfaction of the drivers on that lane, thus increasing the probability of road rage. Our evenhanded criterion has the advantages of both “fairness” and “efficiency” criterions. The optimized drivers’ behavior according to evenhanded criterion tends to change lanes in a reasonable pace.

From the analysis in Section 3.2.4.2., more ending lane would result in a more frequent changing in lane, thus allowing the drivers’ to switch to the non-ending lane in time. From the analysis in Section 3.2.4.2., the probability of changing lane decreases significantly when maximum speed decreases, allowing a more stable flowing of traffic with less chance of sudden speed change.

4. Conclusions

4.1. Strength and Weakness

- Strength
    - Our computer simulation is suitable for almost all possible conditions of the highway.
    - Our microscopic model successfully explain various behaviors of drivers.
    - Our macroscopic evaluation is successfully applied with genetic algorithm to optimize the driver behavior.
- Weakness
    - We do not have enough time allowing the genetic algorithm to iterate with more generations; therefore, our optimization result may not be convincing.
    - We assume the instant changing of lane of cars due to the difficulty of dynamic analysis.

4.2. Possible Policies and Practice

1. The Department of Transportation should establish enough signs to remind drivers early when a lane is reach to an end.
2. The Department of Transportation should not focus only on the the efficiency of the road.

3. The Department of Transportation should set up signs on the lanes to indicate that only the cars on the ending lane can change to its adjacent lane.
4. The Department of Transportation should provide courses for drivers with the following content:
    a. Drivers should not compete with cars on any line.
    b. Drivers on common lane should keep on their lane unless they can drive faster if they change.
    c. Drivers should not change lane at the last moment before colliding into end of lane.
    d. Drivers on ending lane should not change line if there are already a few of cars changing lanes. Nor should they keep in ending lane if no car is changing lane.

5. Proposal

Dear Director of the Department of Transportation:

We devised a mathematical model to investigate the traffic flows on the highway that has an ending lane. In the model, we have an analysis on drivers' behavior microscopically, and build a computer simulation to visualize the traffic on the load. In addition, we also evaluate the overall traffic flow macroscopically by an evaluation function that takes account of fairness, efficiency, and security of the highway traffic.

According to our model, the Department of Transportation and ordinary drivers on the highway should cooperate together to solve the problem of road rage in a more fundamental way: improving the traffic conditions. The Department should add more signs on the road to help drivers to make more rational decisions, while the drivers should attend courses set up by the Department to learn the rational response to different traffic conditions.

We strongly recommend you to apply our model to improve the road conditions. Our computer simulator has an inherent advantage of precision simulation of microscopic behaviors and macroscopic situations. In addition, it can simulate various conditions of the highways only by changing some constants defined in the program. We are confident that our model is a representation of the reality on the highway and our optimization based on the simulation is promising.

Sincerely Yours,

HIMCM 2015 Team Anonymous

Appendix: Code

```
function main
    mode = 2;

    global coefficientOfProbScaling_Distance;
    global coefficientOfProbScaling_TimeUntilDeadline;
    global coefficientOfDistanceScaling;
    global coefficientOfTimeScaling;

    global coefficientFactorScalingOfVelocity;
    global coefficientDistanceScalingOfVelocity;
    global coefficientDensityScalingOfVelocity;

    coefficientOfProbScaling_Distance = 0.1;
    coefficientOfProbScaling_TimeUntilDeadline = 0.3;
    coefficientOfDistanceScaling = 0.5;
    coefficientOfTimeScaling = 0.4;

    coefficientFactorScalingOfVelocity = 0.9;
    coefficientDistanceScalingOfVelocity = 0.1;
    coefficientDensityScalingOfVelocity = 1;

    coefficientOfProbScaling_Distance = 0.419;
    coefficientOfProbScaling_TimeUntilDeadline = 0.621;
    coefficientOfDistanceScaling = 0.963;
    coefficientOfTimeScaling = 0.005;

    coefficientFactorScalingOfVelocity = 0.668;

    p1Matrix = [1 5/7 5/3;7/5 1 7/3;3/5 3/7 1];
    p2Matrix = [1];
    p3Matrix = [1];
    cArray{1} = p1Matrix;
    cArray{2} = p2Matrix;
    cArray{3} = p3Matrix;
    % relation between hierarchies

    DecisionTree = [1 0 0;1 0 1;1 1 0];
    % connection of elements

    DecisionTreeScoreEfficiencyBiased = [1 1 1/9;1 1 1/9; 9 9 1];
    DecisionTreeScoreFairnessBiased = [1 1/9 1;9 1 9;1 1/9 1];
    DecisionTreeScoreEvenHanded = [1 7/3 7/5;5/7 1 1;5/7 1 1];

    coefficientOfEvaluationEfficiencyBiased = AHP(DecisionTree, DecisionTreeScoreEfficiencyBiased, cArray)
    coefficientOfEvaluationFairnessBiased = AHP(DecisionTree, DecisionTreeScoreFairnessBiased, cArray)
    coefficientOfEvaluationEvenHanded = AHP(DecisionTree, DecisionTreeScoreEvenHanded, cArray)

    coefficientOfEvaluation = coefficientOfEvaluationEvenHanded;

    if mode == 1
        mainFunctionOfUpdatingPosition(mode);
    else
        numberOfPop = 10;
        lengthGene = 5;
        precisionOfGene = 0.001;

        baseOfGene = floor(1 / precisionOfGene);
        baseV = ones(1, lengthGene) * baseOfGene;

        maxGeneration = 1; % maximum number of initeration
        gap = (numberOfPop - 1)/numberOfPop; % generation gap

        chromo = crtbp(numberOfPop, lengthGene, baseOfGene);

        functionValue = evaluateChromos(chromo, precisionOfGene, coefficientOfEvaluation);

        recorder = zeros(maxGeneration, 2);
        % The recorder is for the graph showing the process of
        % optimization.

        generationCounter = 0;
        finalChromo = zeros(numberOfPop, lengthGene);
        while generationCounter < maxGeneration
            fitValue = ranking( functionValue ); % deal with minimize
```

```
            selectionChromo = select('rws', chromo, fitValue, gap); % selection
            selectionChromo = recombin('xovsp', selectionChromo); % crossover
            selectionChromo = mut(selectionChromo, 0.7/lengthGene, baseV); % mutation

            functionValueSelection = evaluateChromos(selectionChromo, precisionOfGene, coefficientOfEvaluation);
            [chromo, functionValue] = reins(chromo, selectionChromo, 1, 1, functionValue, functionValueSelection);
            % According to the function, insert new child set into the new
            % population

            [f, id] = min(functionValue);
            generationCounter = generationCounter + 1

            recorder(generationCounter, 1) = f;
            recorder(generationCounter, 2) = id;
            finalChromo = chromo;
            % output
        end

        plot(recorder(:, 1), 'r-*');
        xlabel('#Initeration');
        ylabel('Function Optimization Process');
        optimizedValue = recorder(generationCounter, 1) % max
        finalChromo(recorder(generationCounter, 2), :) % gene
    end
end

function [sumD, sumV, sumS] = mainFunctionOfUpdatingPosition(mode)
% mode == 1 ==> display
% mode == 2 ==> for genetic algorithm

    global widthOfCar;
    global lengthOfCar;
    global widthOfLane;
    global maxVelocity;
    global lengthOfAdditionalRightLane;
    global safeDistanceBetweenCars;

    global numberOfCars;
    global numberOfLanes;

    global isRightLane;

    widthOfCar = 2.19;
    lengthOfCar = 5.82;
    widthOfLane = 3.6;
    maxVelocity = 65 * 0.44704; % mile/hour * 0.44704 = meter/second
    lengthOfAdditionalRightLane = 1000;
    safeDistanceBetweenCars = 5;

    numberOfCars = 21;
    numberOfLanes = 3;

    isRightLane = [0 0 1];

    dt = 0.08;

    positionMatrix = randomInitialPosition();

    bool = 1;
    tCounter = 0;

    sumDensity = 0;
    sumVelocity = 0;
    sumSTD = 0;

    while bool == 1
        % updateData
        positionMatrix = changeLane(positionMatrix);
        [positionMatrix, velocity, density, stdOfDensity] = updatePosition(positionMatrix, dt);


        if density == inf
            density = 0;
        end
        if isnan(stdOfDensity)
            stdOfDensity = 0;
        end
```

```
        sumDensity = sumDensity + density;
        sumVelocity = sumVelocity + velocity;
        sumSTD = sumSTD + stdOfDensity;

        tCounter = tCounter + dt;

        if max(positionMatrix(:, 2)) > (lengthOfAdditionalRightLane * 1.5);
           bool = 0;
        end

        if mode == 1
            % Clearing figure
            clf;
            % Drawing frame

            for i = 1:numberOfCars;
                laneNum = positionMatrix(i, 1);
                posX = positionMatrix(i, 2);
                posY = (laneNum - 1/2) * widthOfLane;
                drawCar(posX, posY);
            end

            axis auto;
            % The axis is in the metric of meter.
            pause(dt);
        end
    end

    sumD = sumDensity;
    sumV = sumVelocity;
    sumS = sumSTD;

end

% Specific Algorithms STARTS
function y = randomInitialPosition()
    global numberOfLanes;
    global numberOfCars;
    global lengthOfCar;

    positionMatrix = zeros(numberOfCars, 2);

    yPos = zeros(1, numberOfLanes);

    for i = 1:floor(numberOfCars/numberOfLanes)
        yDisplacementAdditional = lengthOfCar+rand(1, numberOfLanes)*50;
        yPos = yPos + yDisplacementAdditional;
        numLane = 1:numberOfLanes;

        for ii = 1:numberOfLanes
            positionMatrix((i-1)*numberOfLanes + ii, :) = [numLane(ii), yPos(ii)];
        end
    end
    positionMatrix = sortrows(positionMatrix, [1 2]);
    y = positionMatrix;
end

function y = findNearestDistance(positionOfCAR, positions, laneNumOfPositions)
    global lengthOfAdditionalRightLane;
    global safeDistanceBetweenCars;
    global widthOfLane;
    global isRightLane;

    nearest = inf;
    if and(positionOfCAR < (lengthOfAdditionalRightLane - widthOfLane - safeDistanceBetweenCars), isRightLane(laneNumOfPositions)
== 0)
        positions = [positions inf];
    end

    for i = 1: length(positions)
        positionOfAnotherCar = positions(i);
        distance = positionOfAnotherCar - positionOfCAR;
        if distance > 0
           if (distance < nearest)
              nearest = distance;
           end
        end
    end
```

```
    y = nearest;
end

function y = findNearestDistances(positionsOfCARs, positions, laneNumOfPositions)
    returnVector = [];
    for i = 1: length(positionsOfCARs)
       positionOfCAR = positionsOfCARs(1, i);
       returnVector = [returnVector findNearestDistance(positionOfCAR, positions, laneNumOfPositions)];
    end
    y = returnVector;
end

function y = findPositionArray(positionMatrix)
    global numberOfLanes;
    global numberOfCars;

    for i = 1:numberOfLanes
        positionArray{i} = [];
    end

    for i = 1:numberOfCars
        numLane = positionMatrix(i, 1);
        yPos = positionMatrix(i, 2);
        positionArray{numLane} = [positionArray{numLane} yPos];
    end

    y = positionArray;
end

function y = densityCalculation(positionArray)
    len = length(positionArray);
    answerVector = zeros(1, len);
    for i = 1:len
        positionOfOneLane = positionArray{i};
        density = 0;
        numberOfCarsInOneLane = length(positionOfOneLane);
        if numberOfCarsInOneLane > 0
           density = numberOfCarsInOneLane / (positionOfOneLane(numberOfCarsInOneLane) - positionOfOneLane(1));
        end
        answerVector(1, i) = density;
    end

    y = answerVector;
end

function y = changeLane(positionMatrix)
    global numberOfLanes;
    global isRightLane;
    global widthOfCar;
    global lengthOfCar;
    global lengthOfAdditionalRightLane;
    global safeDistanceBetweenCars;

    newPositionMatrix = zeros(size(positionMatrix));

    positionArray = findPositionArray(positionMatrix);

    counter = 1;
    for i = 1:numberOfLanes
       positionsOfOneLane = positionArray{i};
       numberOfCarsInOneLane = length(positionsOfOneLane);

       if not(numberOfCarsInOneLane == 0)
           diff = positionsOfOneLane(1, 2: numberOfCarsInOneLane) - positionsOfOneLane(1, 1: numberOfCarsInOneLane - 1);

           if isRightLane(i) == 1
               diffFinal = lengthOfAdditionalRightLane - positionsOfOneLane(numberOfCarsInOneLane);
               if diffFinal < safeDistanceBetweenCars
                   diffFinal = 0;
               end
               diff = [diff diffFinal];
           else
               diff = [diff inf];
           end
       else
           diff = [];
       end
```

```
        changeToLane = zeros(size(diff));
        notChange = zeros(size(diff));
        changeMatrix = zeros(4, length(diff));
        idOfCars = counter:(counter+length(diff)-1);
        deltaDistances = zeros(size(diff));
        rightLaneDistance = positionsOfOneLane;


        if or(i == 1, i == numberOfLanes)
            if i == 1
                changeToLaneUndecided = 2;
            elseif i == numberOfLanes
                changeToLaneUndecided = numberOfLanes - 1;
            end
            nearestDistances = findNearestDistances(positionsOfOneLane, positionArray{changeToLaneUndecided},
changeToLaneUndecided);
            changeToLane = ones(size(diff)) * changeToLaneUndecided;

            deltaDistances = nearestDistances - diff;
            notChange = 0;
            if isRightLane(i) == 0
                notChange = diff >= nearestDistances;
            end

            notChangeDueEndOfLane = 0;
            if isRightLane(changeToLaneUndecided) == 1
                notChangeDueEndOfLane = nearestDistances == inf;
            end

            notChangeSinceCollision = diff < widthOfCar/2;
            changeToLane = changeToLane .* not(notChangeDueEndOfLane) .* not(notChangeSinceCollision);
        else
            leftNearestDistances = findNearestDistances(positionsOfOneLane, positionArray{i-1}, i-1);
            rightNearestDistances = findNearestDistances(positionsOfOneLane, positionArray{i+1}, i+1);

            nearestDistances = min(leftNearestDistances, rightNearestDistances);

            isToRight = rightNearestDistances == nearestDistances;
            changeToLane = i - 1 + isToRight*2;
            % if it is changing to the right, isToRIght == 1, then
            % changeToLane = i - 1 + 1*2, else changeToLane = i - 1

            deltaDistances = nearestDistances - diff;
            scaleDownCoefficient = 0.65;
            deltaDistances = deltaDistances * scaleDownCoefficient;
            % For the lane that has two adjacent lanes, it has more
            % probability to change lane, so the deltaDistance is divided by
            % two to scale down its chance.


            notChangeDueEndOfLane = 0;
            if isRightLane(changeToLane) == 1
                notChangeDueEndOfLane = nearestDistances == inf;
            end

            notChangeSinceCollision = diff < widthOfCar/2;
            changeToLane = changeToLane .* not(notChangeDueEndOfLane) .* not(notChangeSinceCollision);

            notChange = 0;
            if isRightLane(i) == 0
                notChange = diff >= nearestDistances;
            end
        end

        changeToLane = changeToLane .* not(notChange);
        % if do change, then NOT(notChange) = 1, else == 0, if the value
        % is 0, it implies that this car will not change lane.

        rightLaneDistance = rightLaneDistance * isRightLane(1, i);

        if not(numberOfCarsInOneLane == 0)
            changeMatrix(1, :) = idOfCars;
            changeMatrix(2, :) = changeToLane;
            changeMatrix(3, :) = deltaDistances;
            changeMatrix(4, :) = rightLaneDistance;
        end

        changeMatrixWithProbAndExec = kickOutFailedChange(calculateProbabilityAndResultAccordingToChangeMatrix(changeMatrix));
```

```
        newSize = size(changeMatrixWithProbAndExec);
        for ii = 1: newSize(2)
            id = changeMatrixWithProbAndExec(1, ii) - counter + 1;
            changeMatrixWithProbAndExec(6, ii) = positionsOfOneLane(1, id);
        end

        sizeOf = size(changeMatrixWithProbAndExec);
        if sizeOf(1) == 0
            changeMatrixArray{i} = [];
        else
            changeMatrixForConflictCheck = calculateNewPosition(changeMatrixWithProbAndExec);
            changeMatrixArray{i} = changeMatrixForConflictCheck;
        end

        counter = counter+length(diff);
    end

    % conflict check start
    for i = 1:length(changeMatrixArray)
        changeMatrix = changeMatrixArray{i};

        newChangeMatrix = [];

        sizeOfMatrixForCheck = size(changeMatrix);
        if not(sizeOfMatrixForCheck(2) == 0)
            for ii = 1:sizeOfMatrixForCheck(2)
                position = changeMatrix(3, ii);
                probability = changeMatrix(4, ii);

                for iii = 1:length(changeMatrixArray)
                    if not(iii == i)
                        changeMatrixForRef = changeMatrixArray{iii};
                        sizeOfChangeMatrixForRef = size(changeMatrixForRef);
                        if (sizeOfChangeMatrixForRef(1) == 0)
                            newChangeMatrix = [newChangeMatrix changeMatrix(:, ii)];
                        else
                            positionsForRef = changeMatrixForRef(3, :);
                            probabilityForRef = changeMatrixForRef(3, :);
                            if willCollideDet(position, positionsForRef) == 0
                                diffDistance = abs(positionsForRef - position);
                                successByDistance = diffDistance > lengthOfCar;

                                success = 0;
                                if (sum(successByDistance) == length(successByDistance))
                                    % pass confirmation, since:
                                    % if the above equality is confirmed, it means the
                                    % car is safe from any other car
                                    success = 1;

                                else
                                    diffProbability = probability - probabilityForRef;
                                    successByProbability = diffProbability < 0;
                                    if (sum(successByProbability) == length(successByProbability))
                                        % pass confirmation according to the same
                                        % principle
                                        success = 1;
                                    end
                                end

                                if success == 1
                                    newChangeMatrix = [newChangeMatrix changeMatrix(:, ii)];
                                end
                            end
                        end
                    end
                end
            end
        end

        newChangeMatrixArray{i} = newChangeMatrix;
    end
    % conflict check end

    y = newPositionMatrixByChangeMatrixArray(positionMatrix, newChangeMatrixArray);
end

function y = calculateProbabilityAndResultAccordingToChangeMatrix(changeMatrix)
    global lengthOfAdditionalRightLane;
```

```
    global maxVelocity;

    sizeOfMatrix = size(changeMatrix);

    probLine = zeros(1, sizeOfMatrix(2));
    resultLine = rand(1, sizeOfMatrix(2));
    for i = 1: sizeOfMatrix(2)
        prob = 0;
        oneCarVector = changeMatrix(:, i);
        changeTo = oneCarVector(2, 1);
        deltaDistance = oneCarVector(3, 1);

        rightLaneDistance = oneCarVector(4, 1);
        if (rightLaneDistance == 0)
            prob = probabilityCalculationGeneral(deltaDistance);
        else
            time = (lengthOfAdditionalRightLane - rightLaneDistance) / maxVelocity;
            prob = probabilityCalculationIsRightMostLane(deltaDistance, time);
        end
        if changeTo == 0
            prob = 0;
        end
        probLine(1, i) = prob;
        resultLine = probLine > resultLine;
        % generate a random number between 0 and 1. Success rate is equal
        % to the probability that the number generated is smaller than the
        % numerical value of probability.
    end

    newMatrix = zeros(6, sizeOfMatrix(2));
    newMatrix(1:4, :) = changeMatrix;
    newMatrix(5, :) = probLine;
    newMatrix(6, :) = resultLine;

    y = newMatrix;
end

function y = kickOutFailedChange(changeMatrixWithResult)
    newMatrix = [];
    sizeOfMatrix = size(changeMatrixWithResult);
    for i = 1:sizeOfMatrix(2)
        if not(changeMatrixWithResult(6, i) == 0)
           newMatrix = [newMatrix changeMatrixWithResult(:, i)];
        end
    end
    y = newMatrix;
end

function y = calculateNewPosition(changeMatrix)
    global widthOfLane;
% In this change matrix, it only contains
%   - idOfTheCar
%         - changeToLand
%         - newPosition
%         - probability

    sizeOfMatrix = size(changeMatrix);
    newMatrix = zeros(4, sizeOfMatrix(2));

    newMatrix(1, :) = changeMatrix(1, :);
    newMatrix(2, :) = changeMatrix(2, :);
    newMatrix(4, :) = changeMatrix(5, :);
    newMatrix(3, :) = changeMatrix(6, :) + widthOfLane;

    y = newMatrix;
end

function y = newPositionMatrixByChangeMatrixArray(positionMatrix, newChangeMatrixArray)
    for i = 1: length(newChangeMatrixArray)
        changeMatrix = newChangeMatrixArray{i};
        sizeOf = size(changeMatrix);
        if sizeOf(1) == 0
            % DO NOTHING
        else
            id = changeMatrix(1, :);
            positionMatrix(id, 1) = changeMatrix(2, :);
            positionMatrix(id, 2) = changeMatrix(3, :);
        end
```

```
    end
    y = positionMatrix;
end

function [positionMat, vel, d, stdOfD] = updatePosition(positionMatrix, dt)
    global numberOfLanes;
    global lengthOfAdditionalRightLane;
    global safeDistanceBetweenCars;
    global isRightLane;

    newPositionMatrix = zeros(size(positionMatrix));

    positionArray = findPositionArray(positionMatrix);
    densities = densityCalculation(positionArray);

    d = sum(densities);
    stdOfD = std(densities);

    velocityCounter = 0;
    counter = 1;
    for i = 1:numberOfLanes
       positionsOfOneLane = positionArray{i};
       numberOfCarsInOneLane = length(positionsOfOneLane);

       if not(numberOfCarsInOneLane == 0)
           density = 0;

           if or(i == 1, i == numberOfLanes)
               if i == 1
                   density = densities(2);
               elseif i == numberOfLanes
                   density = densities(numberOfLanes - 1);
               end
           else
               density = densities(i+1) + densities(i-1);
           end
           %density = numberOfCarsInOneLane / (positionsOfOneLane(numberOfCarsInOneLane) - positionsOfOneLane(1));

           diff = positionsOfOneLane(1, 2: numberOfCarsInOneLane) - positionsOfOneLane(1, 1: numberOfCarsInOneLane - 1);

           if isRightLane(i) == 1
               diffFinal = lengthOfAdditionalRightLane - positionsOfOneLane(numberOfCarsInOneLane) - safeDistanceBetweenCars;
               if diffFinal < safeDistanceBetweenCars
                   diffFinal = 0;
               end
               diff = [diff diffFinal];
           else
               diff = [diff inf];
           end
           velocities = zeros(1, length(diff));
           for ii = 1: length(diff)
               stopCarCompensation = 1;
               if isRightLane(i) == 1
                   stopCarCompensation = ( lengthOfAdditionalRightLane - positionsOfOneLane(ii) ) > safeDistanceBetweenCars;
               end

               velocities(1, ii) = velocityCalculation(diff(1, ii), density) * stopCarCompensation;
           end
           velocityCounter = velocityCounter + sum(velocityCounter);
           displacements = velocities * dt;
           positionsOfOneLane = positionsOfOneLane + displacements;

           newPositionMatrix(counter: (counter + length(displacements) - 1), 1) = i;
           positionsOfOneLane = positionsOfOneLane';
           newPositionMatrix(counter: (counter + length(displacements) - 1), 2) = positionsOfOneLane;
           counter = counter + length(displacements);
       end
    end

    newPositionMatrix = sortrows(newPositionMatrix, [1 2]);
    positionMat = newPositionMatrix;
    vel = velocityCounter;
end

function y = drawCar(x, y)
    global widthOfCar;
    global lengthOfCar;
    y = rectangle('Position', [x - lengthOfCar/2, y - widthOfCar/2, lengthOfCar, widthOfCar],'EdgeColor', 'r', 'Facecolor', 'r');
```

```
end


function y = velocityCalculation(distance, density)
    global maxVelocity;
    global lengthOfCar;
    global safeDistanceBetweenCars;

    global coefficientFactorScalingOfVelocity;
    global coefficientDistanceScalingOfVelocity;
    global coefficientDensityScalingOfVelocity;

    k = coefficientFactorScalingOfVelocity;
    v0 = maxVelocity;
    lambda1 = coefficientDistanceScalingOfVelocity;
    lambda2 = coefficientDensityScalingOfVelocity;
    v = v0 * (k * ( 2 / ( 1 + exp(-lambda1 * (distance - lengthOfCar - safeDistanceBetweenCars)) ) - 1) + (1 - k) * ( 1 / ( 1 +
exp(lambda2 * density))) );

    if v < 10
        v = 0;
    end

    y = v;
end

function y = probabilityCalculationGeneral(distance)
    global coefficientOfProbScaling_Distance;
    global coefficientOfDistanceScaling;

    prob = 2 * coefficientOfProbScaling_Distance * (1 / (1 + exp(-coefficientOfDistanceScaling*distance) ) - 1/2);
    % coefficientOfProbScaling / ( 1 + exp(-coefficientOfDistanceScaling * distance));
    if (prob < 0)
       prob = 0;
    end
    y = prob;
end

function y = probabilityCalculationIsRightMostLane(distance, time)
    global coefficientOfProbScaling_Distance;
    global coefficientOfProbScaling_TimeUntilDeadline;
    global coefficientOfDistanceScaling;
    global coefficientOfTimeScaling;

    k1 = coefficientOfProbScaling_Distance;
    k2 = coefficientOfProbScaling_TimeUntilDeadline;
    lambda1 = coefficientOfDistanceScaling;
    lambda2 = coefficientOfTimeScaling;

    prob = 2 * k1 * (1 / (1 + exp(-lambda1*distance) ) - 1/2) + k2 / (1 + exp(lambda2*time));
    if (prob < 0)
       prob = 0;
    end
    y = prob;
end

function y = willCollide(distance1, distance2)
    global widthOfLanes;
    global widthOfCar;
    global lengthOfCar;
    % CAR: the car that changes lane
    % car0, car1, car2: other cars.
    % All distance is the displacement in y component
    % distance 1 is the distance between CAR and car0 that is on the left/right of CAR.
    % distance 2 is the distance between car0 and car1 that is before car1.
    bool = 0;
    if (distance1 < (widthOfLanes - (sqrt(2)+1)/2 * widthOfCar - lengthOfCar/2))
        bool = 1;
    elseif (distance2 < ((sqrt(2)+1)/2 * widthOfCar + 3*lengthOfCar/2))
        bool = 1;
    end
    y = bool;
end

function y = willCollideDet(position, positions)
    deltaD = [positions inf] - position;

    secondsmallest = 12000;
```

```
    smallest = 10000;
    index = 1;


    for i = 1: length(deltaD)
        dD = deltaD(i);
        tempSmallest = 0;

        if and(dD < smallest, dD > 0)
            tempSmallest = dD;
            index = i;
        end
        smallest = dD;
    end
    distance1 = deltaD(index);
    distance2 = deltaD(index+1);

    y = willCollide(distance1, distance2);
end

function y = processRawDataAndEvaluate(positionMatrix, time, coefficientOfEvaluation)
    global lengthOfAdditionalRightLane;

    positionMatrix = sortrows(positionMatrix, [1 2]);

    positionArray = findPositionArray(positionMatrix);
    densities = densityCalculation(positionArray);
    density = max(densities);
    velocity = lengthOfAdditionalRightLane * 1.5 / time;

    y = evaluationFunction(density, velocity, std(densities), coefficientOfEvaluation);
end

function y = evaluateChromos(chromos, precision, coefficientOfEvaluation)
    global coefficientOfProbScaling_Distance;
    global coefficientOfProbScaling_TimeUntilDeadline;
    global coefficientOfDistanceScaling;
    global coefficientOfTimeScaling;

    global coefficientFactorScalingOfVelocity;

    sizeOf = size(chromos);
    results = zeros(sizeOf(1), 1);
    for i = 1: sizeOf(1);
        oneLineOfChromo = chromos(i, :) * precision;

        coefficientOfProbScaling_Distance = oneLineOfChromo(1, 1);
        coefficientOfProbScaling_TimeUntilDeadline = oneLineOfChromo(1, 2);
        coefficientOfDistanceScaling = oneLineOfChromo(1, 3);
        coefficientOfTimeScaling = oneLineOfChromo(1, 4);

        coefficientFactorScalingOfVelocity = 0.9 + oneLineOfChromo(1, 5) * 0.1;

        [sumDensity, sumVelocity, sumSTD] = mainFunctionOfUpdatingPosition(2);

        results(i, 1) = evaluationFunction(sumDensity, sumVelocity, sumSTD, coefficientOfEvaluation);
    end

    y = results;
end

function y = evaluationFunction(density, velocity, stdOfDensity, coefficientOfEvaluation)
    dataVector = [density, (- velocity^2), stdOfDensity];
    y = dataVector * coefficientOfEvaluation;
end
% Specific Algorithms ENDS


% General Algorithms STARTS
function y = AHP(decisionTree, decisionLayerScore, methodLayerScoreArray)
% The final output is U*w, where U is the combination of the standardized
% method layer eigenvector, and w is the standardized eigenvector of
% decision layer.

% Decision Tree has the structure of follows: (example)
%     c1    c2   c3
% p1   1    0     1
% p2   1    1     1
```

```
% p3   0    1     0
% 1 ==> c has connection to p, 0 is the opposite

    [eigenVectorDLS, eigenValueDLS] = eig(decisionLayerScore);
    w = eigenVectorDLS(:, 1) / sum(eigenVectorDLS(:, 1));
    % Calculate w

    U = zeros(size(decisionTree));
    % Initialize U.

    % Since the score matrix of the method layer may not have the same
    % dimension, they are stored in a array(matrix) to be fetched in this
    % function.
    for i = 1: length(methodLayerScoreArray)
       oneMethodLayerScore = methodLayerScoreArray{i};
       % Fetch one method layer score from the big array.

       ui = oneMethodLayerScore(:, 1) / sum(oneMethodLayerScore(:, 1));
       % Calculate the i-th u.

       oneColumnOfDecisionTree = decisionTree(:, i);
       % Fetch the structure of the decision tree of this column.

       number = size(oneColumnOfDecisionTree, 1);
       % the number of zeros to be filled by the new i-th u generated by
       % the following code.
       NewUi = zeros(number, 1);
       % Initialized new i-th u.

       counter = 1;
       % set a counter, since the empty connection in decisionTree is
       % omitted in the score matrix.

       % Fill the new i-th u according to the structure of the decision
       % tree.
       for ii = 1: number
           oneOrZero = oneColumnOfDecisionTree(ii, 1);
           % Just One or Zero.

           if oneOrZero == 1
               % If it is not empty, then ...
               NewUi(ii, 1) = ui(counter, 1);
               % Fill in the value.
               counter = counter + 1;
               % counter++
           else
              % DO NOTHING
           end
       end

       U(:, i) = NewUi;
       % Plug in the value.
    end

    % THE FINAL STEP!
    y = U*w;
end
% General Algorithms ENDS
```